3rd Annual
Austin Conference on Integrated Systems & Circuits
2008 Publication

KEYNOTE ADDRESS **Dr. Necip Sayiner,** ***President, CEO, and Board Member*** ***Silicon Laboratories***	
Track 1: DIGITAL DESIGN-1 **Session Chair:** ***Mike Seningen***	Track 2: ANALOG/BIO **Session Chair:** ***Ka Leung***
1-1 Design Automation and Verification Methodology Challenges of the Intel Atom Processor ***Rajesh Gupta***	***2-1*** ***Invited Paper:*** Biosensor Microarrays in CMOS ***Arjang Hassibi***
1-2 Reducing Flip-Flop Power for DSP Design ***Bassam Mohd,*** *Martin Saint-Laurent****, Paul Bassett, and Shahid Imam***	2-2 Rail to Rail Fully Differential Sample and Hold Based on Clocked Diffrential Difference Amplifier using Resistive Local Common Mode Feedback ***Jaime Ramirez-Angulo, Clara Lujan-Martinez, Carlos Rubia-Marcos, Ramon G. Carvajal, and Antonio Lopez-Martin***
1-3 Adaptive Voltage Tuning for Dual-Vdd ASICs ***Stephen Bijansky and Adnan Aziz***	2-3 Buck-Boost Converter Based Power Conditioning Circuit for Low Excitation Vibrational Energy Harvesting ***Arvindh Rajasekaran, Abhiman Hande, and Dinesh Bhatia***
Track 3: CAD-1 **Session Chair:** ***Michael Solka***	Track 4: COMPUTER ARITHMETIC **Session Chair:** ***Earl Swartzlander***
3-1 Modeling of NBTI-Induced PMOS Degradation under Arbitrary Dynamic Temperature Variation ***Bin Zhang and Michael Orshansky***	***4-1*** ***Invited Paper:*** Automated Multiplier Design ***K'Andrea C. Bickerstaff and Earl Swartzlander***

3-2 ELIAD: Efficient Lithography Aware Detailed Router with Compact Post-OPC Printability Prediction ***Minsik Cho, Kun Yuan, Yongchan Ban, and David Z. Pan***	4-2 Full Adder Evaluation and Selection for a Parallel Multiplier ***Mike Spear, Gaurav Tuteja, and Earl Swartzlander***
3-3 COOLER - A Fast Multiobjective Fixed-Outline Thermal Floorplanner ***Debarshi Chatterjee, Theodore Manikas, and Igor Markov***	4-3 Dual Vdd Design Optimization of Fast Multipliers ***Eun Jung Jang, Earl Swartzlander, and Jebediah Keefe***
3-4 Dynamic Compaction with Recursive Learning for Delay Test ***Zheng Wang and Duncan M. H. Walker***	4-4 A Hybrid Approach to Last Stage Addition for Wallace and Dadda Multipliers ***James Haydn Nelson, Eiman Ebrahimi, Mahnaz Sadoughi, and Earl Swartzlander***
TUTORIAL Fractional-N PLL ***Dr. Axel Thomsen,*** ***Silicon Labs***	
KEYNOTE ADDRESS **Dr. Jason Rhode,** ***President and CEO*** ***Cirrus Logic***	
Track 5: DIGITAL DESIGN-2 **Session Chair:** ***Adnan Aziz***	Track 6: ANALOG AND RF **Session Chair:** ***TR Viswanathan***
5-1 ***Invited Paper:*** A Sub 1W to 2W Low Power IA Processor for Mobile Internet Devices in 45nm Hi-K Metal Gate CMOS ***Gianfranco Gerosa, Steve Curtis, Mike D'Addeo, Bo Jiang, Belliappa Kuttanna, Feroze Merchant, Binta Patel, Mohammed Taufique, Haytham Samarchi, and Christopher Weaver***	***6-1*** ***Invited Paper:*** Receivers Design: Case Studies ***Hesam Amir-Aslanzadeh and Edgar Sanchez-Sinencio***
5-2 A High Speed 128-Point Fast Fourier Transform Circuit for OFDM Systems ***Tung-Yeh Wu and Jacob Abraham***	6-2 Feed-Forward Interference Suppression for Broadband Systems ***Xin Wang and Ranjit Gharpurey***

5-3 Hardware Trojan Modeling and Detection Techniques ***Daniel G. Saab, Fatih Kocan, and Jacob Abraham***	6-3 A Linear Transconductor using Series-Connected CMOS Quad ***Venkatesh Acharya, Bhaskar Banerjee, and TR*** *Viswanathan*
5-4 ***Invited Paper:*** Migration of Cell Broadband Engine from 65nm SOI to 45nm SOI ***Osamu Takahashi*** ***(Scott Cottier presenting)***	6-4 Indirect Compensation Techniques for Three-Stage CMOS Op-Amps ***Vishal Saxena, Jacob Baker, and TR Viswanathan***
Track 7: CAD-2 **Session Chair:** ***Michael Orshansky***	Track 8: SYSTEMS & ARCHITECTURES **Session Chair:** ***Mattan Erez***
7-1 3D Resistance Extraction with Lithographic and Scattering Effect ***Ying Zhou, Zhuo Li, and Weiping Shi***	8-1 Power Analysis of a Path-Based Perception Branch Predictor ***Justin Friesenhahn, Lizy Kurian John, and Mark McDermott***
7-2 Accelerating Statistical Static Timing Analysis using Graphics Processing Units ***Kanupriya Gulati and Sunil Khatri***	8-2 Hardware / Software Tradeoffs in Multicore Architectures ***Steven Guccione***
7-3 Autonomous Optical Proximity Correction: The New Frontier of Design for Manufacturing? ***Shanhu Shen, Peng Yu, and David Z. Pan***	8-3 Workload Slicing for Detailed Pre-Silicon Power Estimation ***Hassan Al-Sukhni, James Holt, David Lindberg, and Michele Reese***
TUTORIAL Holistic Coupling of Manufacturing and Design ***Dr. Sani Nassif,*** **IBM**	
TUTORIAL Out of Order Superscalar Architecture ***Dr. Derek Chiou,*** **University of Texas at Austin**	
TUTORIAL Constraint Solving for Functional Verification ***Dr. Andreas Kuhlmann,*** **Cadence Berkeley Labs**	

Automated Multiplier Design

K'Andrea C. Bickerstaff

KenQuest LLC
Austin, TX 78735
Tel: 512-944-6377
Email: kbickerstaff@austin.rr.com

Earl E. Swartzlander, Jr.

Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712
Tel: 512-471-5923
Email: eswartzla@aol.com

Abstract—With delay proportional to the logarithm of the multiplier word length, column compression multipliers are the fastest multipliers. Unfortunately, since the design community has assumed that fast multiplication can only be realized through custom design and layout, column compression multipliers are often dismissed as too time consuming and complex to implement. This research demonstrates that an automated multiplier generation and layout process with current CAD tools makes the column compression multiplier a viable option for application specific CMOS products. Techniques for optimal multiplier designs are identified through analysis of area and delay of Wallace and Dadda multipliers.

I. Introduction

Column compression multipliers are of special interest due to their high speed performance. With delay proportional to the logarithm of the operand word length, column compression multipliers are generally faster than array multipliers, whose delay grows linearly with operand word length, especially for large word sizes. They were first introduced by Wallace [1], and later refined by Dadda [2]. With the advent of VLSI, this type of multiplier was difficult to layout and exhibited high interconnect overhead. However, advances in computer-aided design have helped to mitigate these problems. In the literature, reports of fast CMOS implementations, alternative design schemes, and strategies for the pipelining [3]–[5] of column compression multipliers have begun to appear with increasing frequency.

Designs of column compression multipliers have mostly been recommended based on improved speed performance. The issues of power consumption, inter-connect, and layout have not received as much attention, although simulations suggest that they are much more power efficient than array multipliers especially for large operand sizes [6].

In the literature, most high speed multipliers are implemented as custom designs. In such cases, design engineers expend significant time and effort manually constructing layouts to minimize routing loads and optimize timing. A relatively minor change in the design, such as changing the word size or using a different process technology, can require a time-consuming, major redesign of the multiplier.

This research identifies techniques for optimal computer aided designs of column compression multipliers. Practical, realizable multipliers have been developed, using industry standard design and layout tools.

II. Multiplier Design

Thirty two multipliers were developed to examine their area, delay, and power characteristics. These multipliers are realizations of Wallace and Dadda multipliers for operand sizes from 8-bits by 8-bits to 64-bits by 64-bits in 250 nm to 90 nm process technologies. A multiplier generator was developed in Perl to output the multiplier netlists in gate-level Verilog or spice formats.

A. *Column Compression Multipliers*

Column compression multipliers use three steps to multiply two numbers: 1) the bit product matrix is formed, 2) the columns of the bit product matrix are "compressed" to an equivalent two row matrix by the strategic application of counters or compressors, and 3) the two rows are summed with a carry propagating adder (usually assumed to be a carry lookahead adder) to produce the product.

Fig. 1 shows the block diagram of an N-bit by N-bit unsigned multiplier. D flip-flops are used to hold the primary inputs. The flip-flops drive multiple buffers to distribute input signals to the N^2 AND gates that form the bit products.

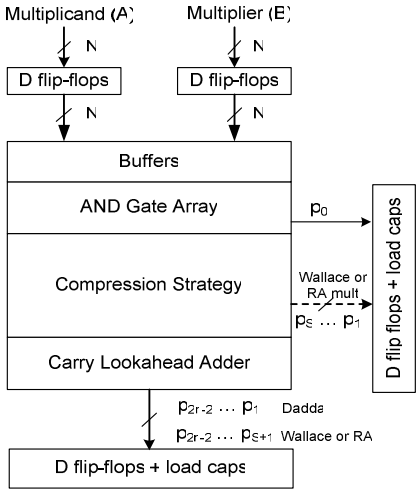


Fig. 1: Block Diagram for N-bit by N-bit Multiplier

B. *Wallace Multipliers*

The Wallace column compression step involves examining three rows at a time of the bit product matrix for the application of (3,2) and (2,2) counters. Using a (3,2) counter, three partial product bits of the same binary weight can be combined to generate a sum bit of the same binary weight and a carry bit of a binary weight increased by one significant position. If there are only two bits with the same binary weight in the three row group of partial products, then a (2,2) counter is used to add the two bits, generating a sum output of the same binary weight and a carry output with binary weight increased by one significant position. Single bits are passed to the next reduction stage. This reduction process is then repeated until there are only two rows of terms remaining. The Wallace column compression step uses the maximum number of full and half adders possible.

The column compression process is illustrated by means of a dot diagram in Fig. 2 for a 6-bit by 6-bit Wallace multiplier. The dot diagram starts with the trapezoidal bit product matrix at the top. Then there is a succession of progressively more compressed matrices (Matrix 1, Matrix 2, etc.). Each matrix shows the results of compressing the previous matrix. Each (3,2) counter takes in three inputs from the preceding matrix and shows the two outputs as two dots connected by a line. Similarly the outputs of a (2,2) counter are shown connected by a line with a slash. In each case the rightmost dot of the pair that is connected by a line is in the column from which the counter inputs were taken in the preceding matrix.

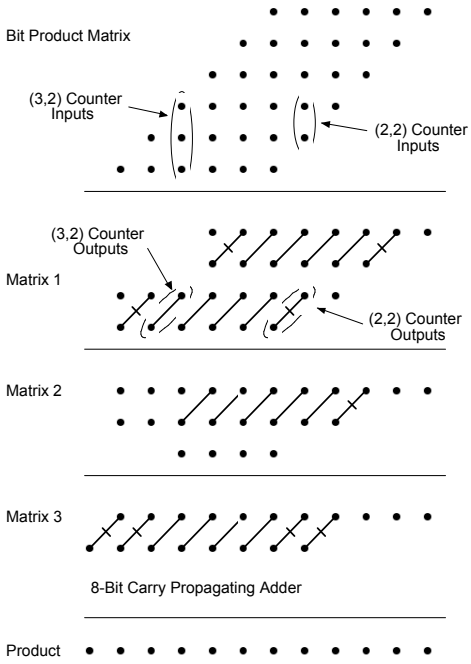


Fig. 2: 6-bit by 6-bit Wallace Multiplier

The compression process for a 12-bit by 12-bit Wallace multiplier is shown in Fig. 3. There are five compression matrices, each of which has a single (3,2) counter delay. This Wallace multiplier requires 144 AND gates, 102 (3,2) counters, 34 (2,2) counters and finally an 18-bit carry propagating adder.

C. *Dadda Multipliers*

The Dadda column compression process is similar to the Wallace process, but performs the least compression necessary on each bit product matrix. The height of the matrices in the reduction process is determined by working back from the final (two row) matrix and limiting the height of each matrix to the largest integer that is no more than 1.5 times the height of its successor. The first terms of the matrix height sequence are 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, 94, etc.

A 12-bit by 12-bit Dadda multiplier is shown in Fig. 4. Since the height of the initial bit product matrix is 12, any column having more than nine bits (or that will grow to more than nine bits due to carries) is reduced so that no column in the next matrix will have more than nine bits. Succeeding matrix heights are 6, 4, 3 and finally 2. As with the Wallace multiplier, there are five compression matrices, each of which has a single (3,2) counter delay. This Dadda multiplier requires 144 AND gates, 99 (3,2) counters, 11 (2,2) counters, and finally a 22-bit carry propagating adder.

In comparing the Wallace and Dadda 12-bit by 12-bit multipliers, the Wallace compression uses a few more (3,2) counters, many more (2,2) counters, and a slightly smaller carry propagating adder. The number of compression stages is the same, so the delay is expected to be approximately the same.

D. *Multiplier Implementation*

The most critical logic cell of the Wallace and Dadda multipliers is the (3,2) counter. Fig. 5 shows the schematic of the (3,2) counter that is a standard cell within each of the standard cell libraries. For this (3,2) counter the slowest paths are from the *a* and the *b* inputs to the carry out, c_{out}.

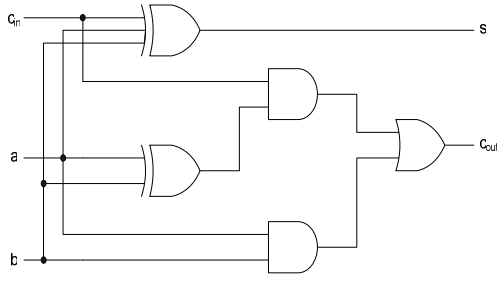


Fig. 5: Schematic of (3,2) Counter

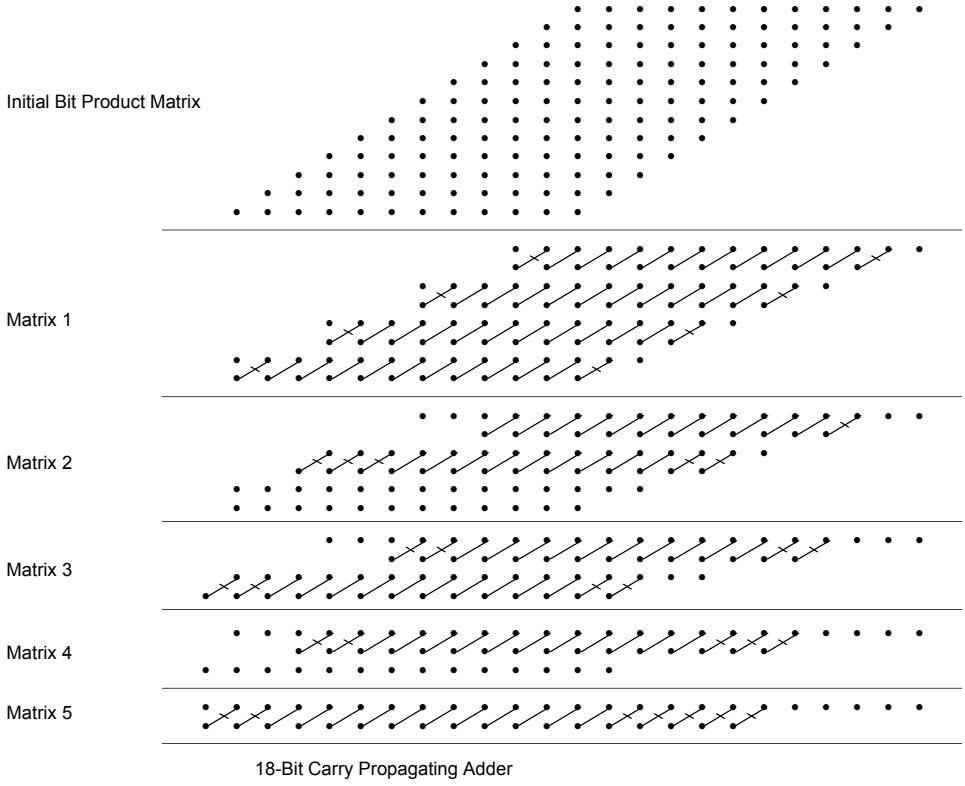


Fig. 3: 12-bit by 12-bit Wallace Multiplier

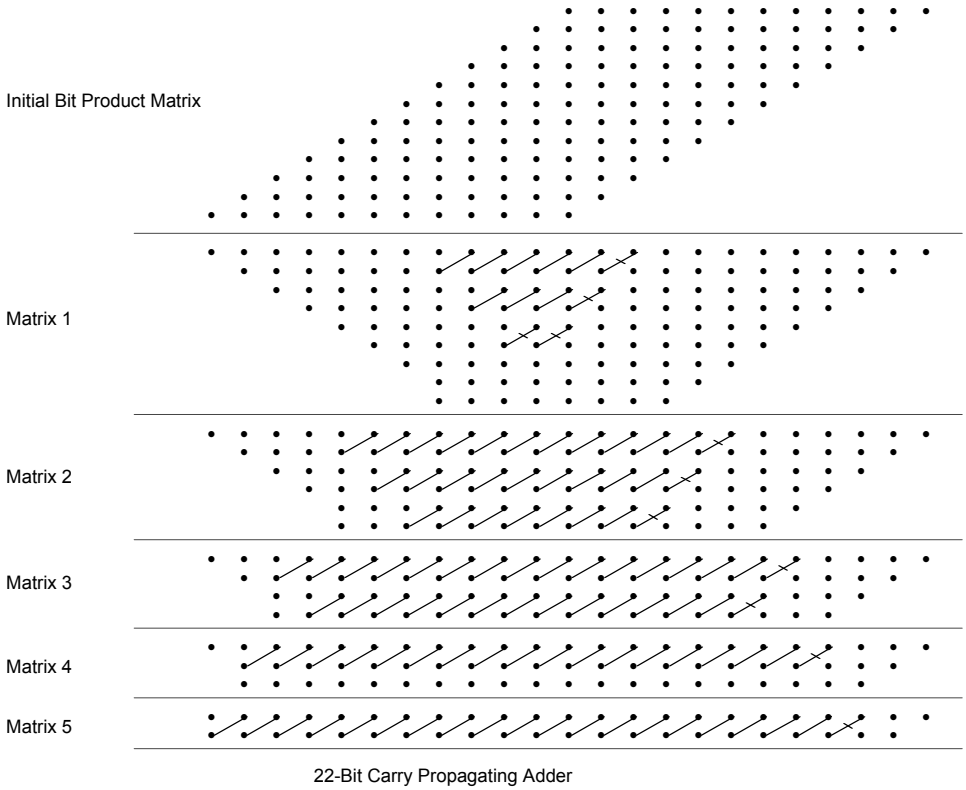


Fig. 4: 12-bit by 12-bit Dadda Multiplier

III. Process Technologies and Cell Libraries

The process technologies used for this research are described in Table 1. All are from a single foundry. The multipliers were implemented using standard cells from state-of-the-art libraries. These libraries were created for mainstream applications with optimizations for speed and density.

TABLE 1

PROCESS TECHNOLOGIES

Feature Size	Metal Layers	Supply Voltage
250 nm	5	2.5 V
180 nm	6	1.8 V
130 nm	8	1.2 V
90 nm	9	1.0 V

The design kit for each standard cell library includes Library Exchange Format (LEF) files and timing files. The LEF file contains the physical information for a process technology as well as geometric abstracts of all of the cells. All of the cell timing files are for the nominal temperature, voltage, and process corner, often named "typical.lib."

IV. Multiplier Implementation and Verification

Premier tools from Cadence Design Systems, Inc., form the backbone of the design environment. Fig. 6 illustrates the design and tool flow for the development and verification of the column compression multipliers. The multiplier netlists were checked using formal verification techniques and then placed and routed. The parasitic resistances and capacitances were then extracted from the layouts and used to back-annotate the netlists for delay analysis and power simulations. Scripting languages, like Perl and C shell, were used to automate often repeated tasks and streamline information extraction.

Column compression multipliers are sometimes described as "complex" and are thought to be "unwieldy" to design. Current computer aided design techniques offer the opportunity to make efficient multipliers. Modern process technologies offer five or more layers of metal for signal routing. Thus it is possible to place all of the multiplier cells without having to leave room for routing channels. The multiplier area is solely dependent on the area of the cells used in the design. Since, for N-bit by N-bit multipliers, the number of the largest cells, the (3,2) counters, grows as N^2, then multiplier area is expected to be roughly proportional to N^2.

To a first-order, each generation of process technology should make all MOS physical dimensions proportional to the minimum feature size, λ, of the process technology [7]. Since the height and width of each cell is proportional to λ, the area of any standard cell is proportional to λ^2, and the total area of an N-bit by N-bit column compression multiplier is expected to be roughly proportional to $\lambda^2 N^2$.

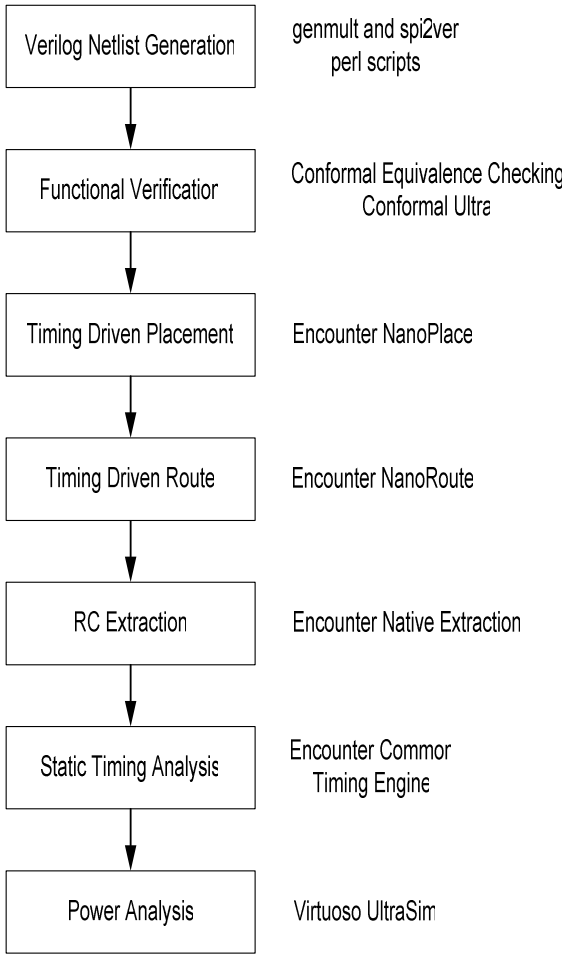


Fig. 6: Design and Tool Flow

V. Multiplier Area

A. *Area Estimation*

As noted previously, a first order area estimate is based on the number of (3,2) counters, which is proportional to N^2. A more exact area estimate can be developed based on the gate counts.

Table 2 summarizes expressions for calculating the number of (3,2) and (2,2) counters used in partial product reduction as well as the word length of the final carry propagate adder as a function of the input multiplier size, N [8]. The expressions for counters and adder word length assume $N \geq 6$ and are also based on the number of reduction stages, S, which is approximately equal to $\log_{1.4} N$. The number of (3,2) counters in Dadda multipliers comes simply from recognizing that there are N^2 bits in the original partial product matrix and $4N - 3$ bits in the final two row matrix.

TABLE 2

COMPONENTS FOR WALLACE AND DADDA MULTIPLIERS

Method	(3,2) Counters	(2,2) Counters	CPA Length
Wallace	$N^2 - 4N + 2 + S$	$\geq N$	$2N - 1 - S$
Dadda	$N^2 - 4N + 3$	$N - 1$	$2N - 2$

For Wallace multipliers, the number of (3,2) counters is approximately $N^2 - 4N + 2 + S$. The number of (2,2) counters is at least N, and is often much greater than N.

The carry lookahead adder for the multipliers uses a maximum lookahead logic block width of four. When needed, 1-bit, 2-bit, and 3-bit lookahead logic blocks were available, so that no additional, unused hardware was included. The complexity of an N-bit carry lookahead adder implemented with 4-bit lookahead logic blocks is roughly 1.4 times the complexity of N (3,2) counters [9].

Therefore, based on gate counts and the complexity of the multiplier components, the Wallace multipliers will have about a 10% larger area than the Dadda multipliers for a given process technology and word size. This is based on first examining the number of (3,2) and (2,2) counters, and finally the size of the carry lookahead adder.

B. *Area Measurements*

All of the multipliers were placed and routed using NanoPlace and NanoRoute of Cadence's Encounter platform. Power and ground were configured to abut in order to minimize the area. No additional routing channels were required. Filler cells were added as needed to extend power and ground lines. Row utilization for all of the multipliers was 95%. Visual inspection of each of the layouts showed dense, compact, very regular layouts. There were no empty sections of any significant size. Though five or more metal layers were available for each process, the 8-bit by 8-bit multipliers required only three metal layers and even the large 64-bit by 64-bit multipliers required only four metal layers.

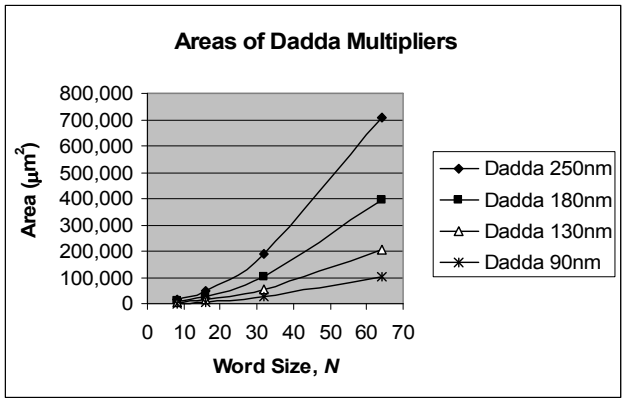


Fig. 7: Dadda Multiplier Areas

For a given process technology, standard cell library, and word size, the area differences among the Wallace and Dadda multipliers are small. For multipliers larger than 8 by 8, Wallace multipliers will be the larger. For 8 by 8 multipliers, Dadda multipliers were larger, due to the area of the carry lookahead adder. Figure 7 shows the layout areas for all of the Dadda multipliers developed in this research. For each process, all of the multiplier areas show slightly less than quadratic growth with increases in N.

The initial prediction was that the area of the column compression multipliers is proportional to $\lambda^2 N^2$, where λ is the process's minimum feature size and N is the input word size. The areas of the four sizes of multipliers (from 8-bits by 8-bits to 64-bits by 64-bits) with 250 nm to 90 nm technologies indicated that predicting area to grow as N^2 does not completely describe the smaller multipliers. An equation estimating area for column compression multipliers needs to include both an N^2 term and an N term.

A least squares method was applied to the measured multiplier areas to calculate approximations to the area. Equations (1) and (2) provide area estimates for the two types of multipliers. These estimates produce maximum errors of about ±7%.

$$\text{Area}_{Wallace} \cong 0.00283\ \lambda^2 N^2 + 0.015\ \lambda^2 N - 0.0472\ \lambda^2 \qquad (1)$$

$$\text{Area}_{Dadda} \cong 0.00275\ \lambda^2 N^2 + 0.0122\ \lambda^2 N - 0.0166\ \lambda^2 \qquad (2)$$

VI. MULTIPLIER DELAY

The automation of column compression multiplier design and layout yields not only compact layouts as discussed in Section V, but also small delay times. Though they differ slightly by the number and significantly by the method of application of (3,2) and (2,2) counters, intuitively, Wallace and Dadda multipliers should have approximately equal delay times for a given word size and process technology since they use the same number of reduction stages.

A. *Delay Estimation*

The total delay of a column compression multiplier is the sum of delays through 1) input signal buffering, 2) bit product formation, 3) compression, and 4) the final carry propagate adder. Since the (3,2) and (2,2) counters are applied in parallel in each stage, the delay of each stage is one (3,2) counter delay. Equation (3) offers a simple estimate for the multiplier delay

$$Delay_{NxN} = t_{\text{buffer}} + t_{\text{AND}} + S \cdot t_{(3,2)} + t_{\text{CPA}(2N-X)} \qquad (3)$$

Where: t_{buffer} is the delay of the input buffers, t_{AND} is the delay of a 2-input AND gate, S is the number of reduction stages ($S \cong \log_{1.4} N$), + $t_{(3,2)}$ is the delay of a (3,2) counter and $t_{\text{CPA}(Z)}$ is the delay of a Z-bit carry propagate adder. In Equation (3), $X = 2$ for Dadda multipliers and $X = S + 1$ for Wallace multipliers.

Since S is proportional to log N and since the delay of a carry lookahead adder also is proportional to log N, for a given process technology, the delays of the Wallace and Dadda multipliers are proportional to log N. Though different numbers of (3,2) and (2,2) counters are applied within each reduction stage, the number of reduction stages for each type of multiplier is the same. The number of reduction stages is proportional to the logarithm of the word size. Especially for large values of N, the delay through the reduction stages dominates the overall multiplier delay. Therefore, the overall multiplier delay is approximately proportional to the logarithm of the word size.

To a first order, with both channel length and supply voltage proportional to λ, the process feature size, a gate's propagation delay is proportional to λ [10]. Since the delay of a given size multiplier is the sum of the same number of gate delays, the multiplier delay is also proportional to λ.

Based on this understanding of the effects of the word size, N, and process geometry, λ, on delay, a rough prediction of column compression multiplier delay is

$$Delay \approx k\,\lambda\,(\log N) \qquad (4)$$

where k is a constant scaling factor.

B. *Delay Measurements*

Delays were estimated using the Common Timing Engine of Cadence's Encounter platform. The Common Timing Engine takes as inputs a design's netlist (Verilog), cell library process information, parasitic resistance and capacitance data, and simulation environment parameters such as temperature and voltage. All of the timing analysis is performed at the nominal voltage level for the specific process technology as given in Table 1. Temperature is set at 25°C. Typical process models are used.

Table 3 lists all of the back-annotated CTE delay data for the Wallace and Dadda multipliers developed in generic standard cell libraries. Overall, for the same word size and process technology, both multipliers show approximately equal delays. For the 64-bit by 64-bit multipliers, the Wallace multipliers are slightly slower than the Dadda multipliers. Figure 5 plots the back-annotated delays for the Dadda multipliers. Inspection of the data shows that the delay is roughly proportional to the logarithm of the word size N.

Equations (5) and (6) provide delay approximations for each type of column compression multiplier, with λ in nanometers.

$$Delay_{Wallace} \cong 0.0094\,\lambda \log_2(N) - 0.011\,\lambda \qquad (5)$$

$$Delay_{Dadda} \cong 0.0082\,\lambda \log_2(N) - 0.0066\,\lambda \qquad (6)$$

TABLE 3

BACK-ANNOTATED DELAYS FOR WALLACE AND DADDA MULTIPLIERS

Word Size	Process	Wallace (nsec)	Dadda (nsec)
8 by 8	250 nm	4.8	4.7
8 by 8	180 nm	3.2	3.2
8 by 8	130 nm	2.6	2.6
8 by 8	90 nm	1.5	1.5
16 by 16	250 nm	6.9	6.7
16 by 16	180 nm	4.7	4.5
16 by 16	130 nm	3.8	3.8
16 by 16	90 nm	2.2	2.2
32 by 32	250 nm	8.6	8.5
32 by 32	180 nm	5.9	5.8
32 by 32	130 nm	4.8	4.6
32 by 32	90 nm	2.9	2.8
64 by 64	250 nm	12.6	10.9
64 by 64	180 nm	8.0	7.4
64 by 64	130 nm	6.6	5.9
64 by 64	90 nm	3.9	3.9

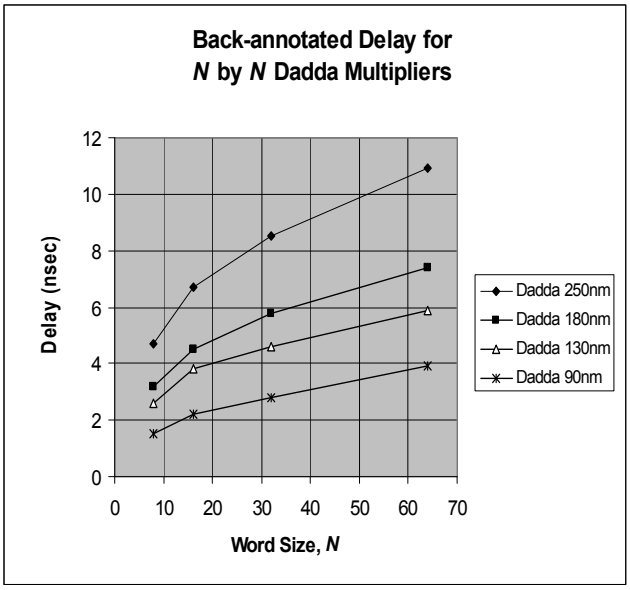


Fig. 8: Back-annotated delay for N by N Dadda multipliers

VII. CONCLUSIONS

During the past decade, many chip architects and IC designers have insisted that fast multipliers are only realized by custom design and layout, which often takes an engineer three months or more to design, layout, and verify. Column compression multipliers are dismissed as too time consuming and complex to design and layout because of their irregular structure. Unlike the two to three

year development periods allotted for large microprocessors, the time to market for application specific ICs is typically three to six months. This research demonstrates that an automated multiplier generation and layout process makes the column compression multiplier a viable option for such products.

The automated place and route of the multipliers yields extremely compact and regular layouts. All of the multipliers show very high row utilizations at 95%. Across different process technologies, doubling the operand size will increase the total area by approximately a factor of four for each type of multiplier. Area for column compression multipliers reduces by approximately 0.5 for each generational transition in the minimum feature size. This research has shown that the area of an N by N column compression multiplier is reasonably well estimated by an approximation that includes both an N^2 term and an N term.

The delay data of this research challenges the prediction that multiplier delay increases in proportion to the logarithm of the word size. A better approximation for delay includes an additional λ term.

Column compression multipliers should be used when fast multiplication is needed in CMOS products that are largely developed using automated design, layout, and verification practices. When the IC development schedule is short, these multipliers can be generated, placed and routed, and simulated in a matter of days instead of the two or three months required for a custom multiplier. With the automation of the development of the column compression multipliers, chip architects and designers can quickly accommodate changes in the word size or the selection of a different semiconductor foundry. Also, once developed, the multiplier's fully verified netlist can be easily reused for future products.

Finally, this research shows the critical importance of the (3,2) counters. Regardless of which type of column compression multiplier is selected, an IC designer can significantly improve the multiplier's performance by adjusting the (3,2) counter cell. For the 130 nm and smaller process geometries, many standard cell library vendors offer special (3,2) counter cells that have been tailored to be low power or high speed. If time is available for any custom design and layout, then develop a new (3,2) counter that fits the performance goals. Note that for a new, custom (3,2) counter cell to be inserted into a project's set of standard cells, the new cell would require significant simulation time to fully characterize it and create its timing file as well as the creation of multiple layout views.

REFERENCES

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

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

[3] P. R. Cappello and K. Steiglitz, "A VLSI Layout for a Pipelined Dadda Multiplier," *ACM Transactions on Computer Systems*, vol. 1, pp. 157-174, 1983.

[4] L. Breveglieri, L. Dadda, and V. Piuri, "Column Compression Pipelined Multipliers," *Proceedings 1995 International Conference on Application Specific Array Processors*, pp. 93-103, 1995.

[5] Jieh-Hwang Yen, Lan-Rong Dung, and Chi-Yuan Shen, "Design of Power-Aware Multiplier with Graceful Quality-Power Trade-Offs," *IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 1642-1645, May 2005.

[6] Thomas K. Callaway and Earl E. Swartzlander, Jr., "Optimizing Multipliers for WSI," *Proceedings of the 1993 International Conference on Wafer Scale Integration*, pp. 85-94, 1993.

[7] Robert H. Dennard, Fritz H. Gaensslen, Hwa-Nien Yu, V. Leo Rideout, Ernest Bassous, and Andre R LeBlanc, "Design of Ion-Implanted MOSFETs with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, vol. SC-9, pp. 256-268, 1974.

[8] K'Andrea C. Bickerstaff, Michael J. Schulte, and Earl E. Swartzlander, Jr., "Reduced Area Multipliers," *Proceedings of the 1993 International Conference on Application Specific Array Processors*, pp. 478-489, 1993.

[9] Earl E. Swartzlander, Jr., "High-Speed Computer Arithmetic," in Allen B. Tucker, ed., *The Computer Science and Engineering Handbook*, Boca Raton: CRC Press, pp. 462-481, 1997.

[10] Neil H. Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, 2nd Edition, Reading, MA: Addison-Wesley Publishing Co., 1993.

Automated Multiplier Design

K'Andrea C. Bickerstaff

and

Earl E. Swartzlander, Jr.

Purpose

For Wallace and Dadda Multipliers:

- Demonstrate automated design and layout
- Analyze area and delay characteristics
- Identify optimal design techniques

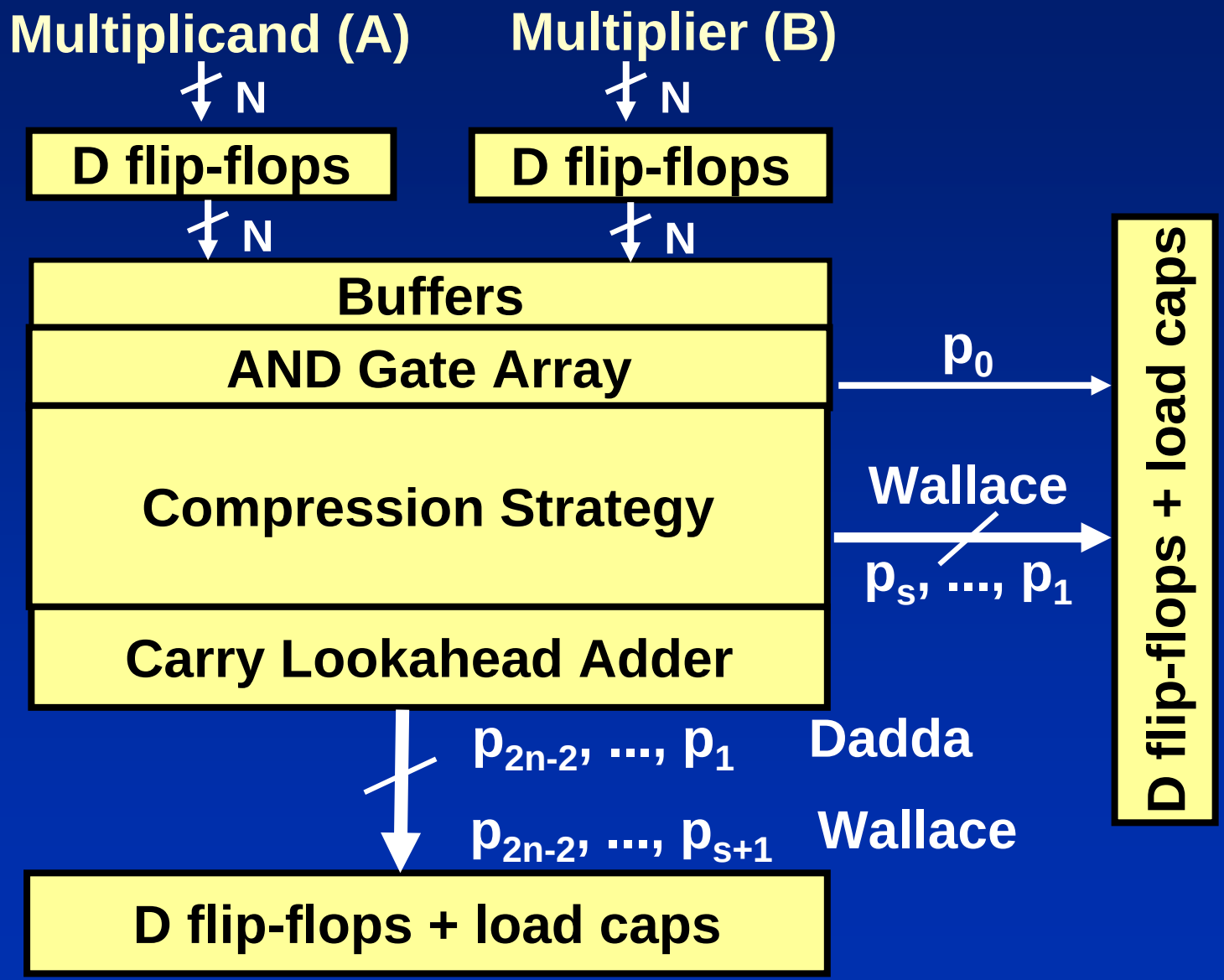
Outline

- Introduction to CC Multipliers
- Design and Implementation Details
- Area Estimation and Measurements
- Delay Estimation and Measurements
- Conclusions

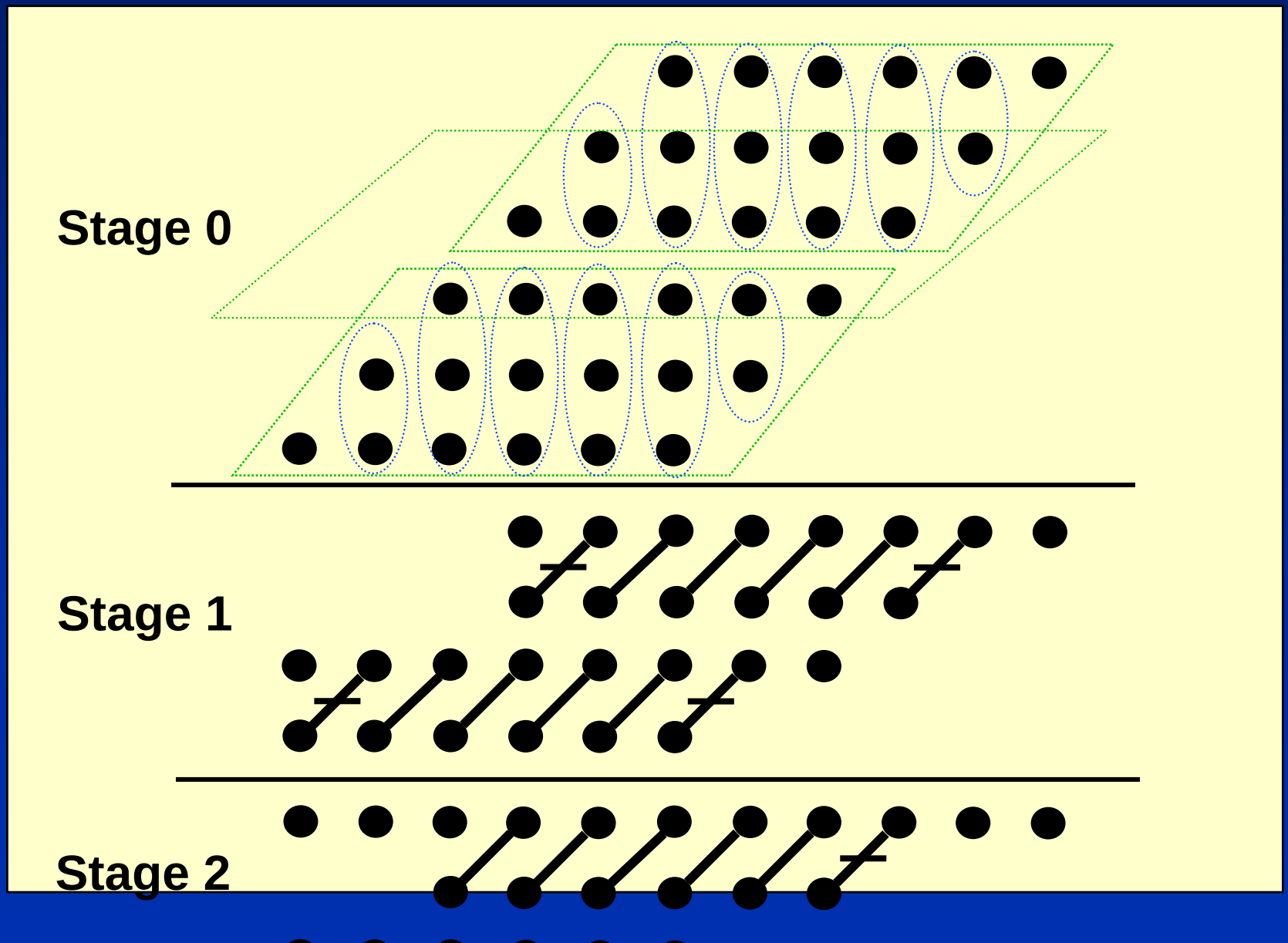
Introduction to CC Multipliers

- Introduced by Wallace in 1964
- Refined later by Dadda in 1965
- Delay proportional to log (N)
- In the literature,
  - alternative design schemes
  - strategies for pipelining
- Dismissed as too complex and unwieldy

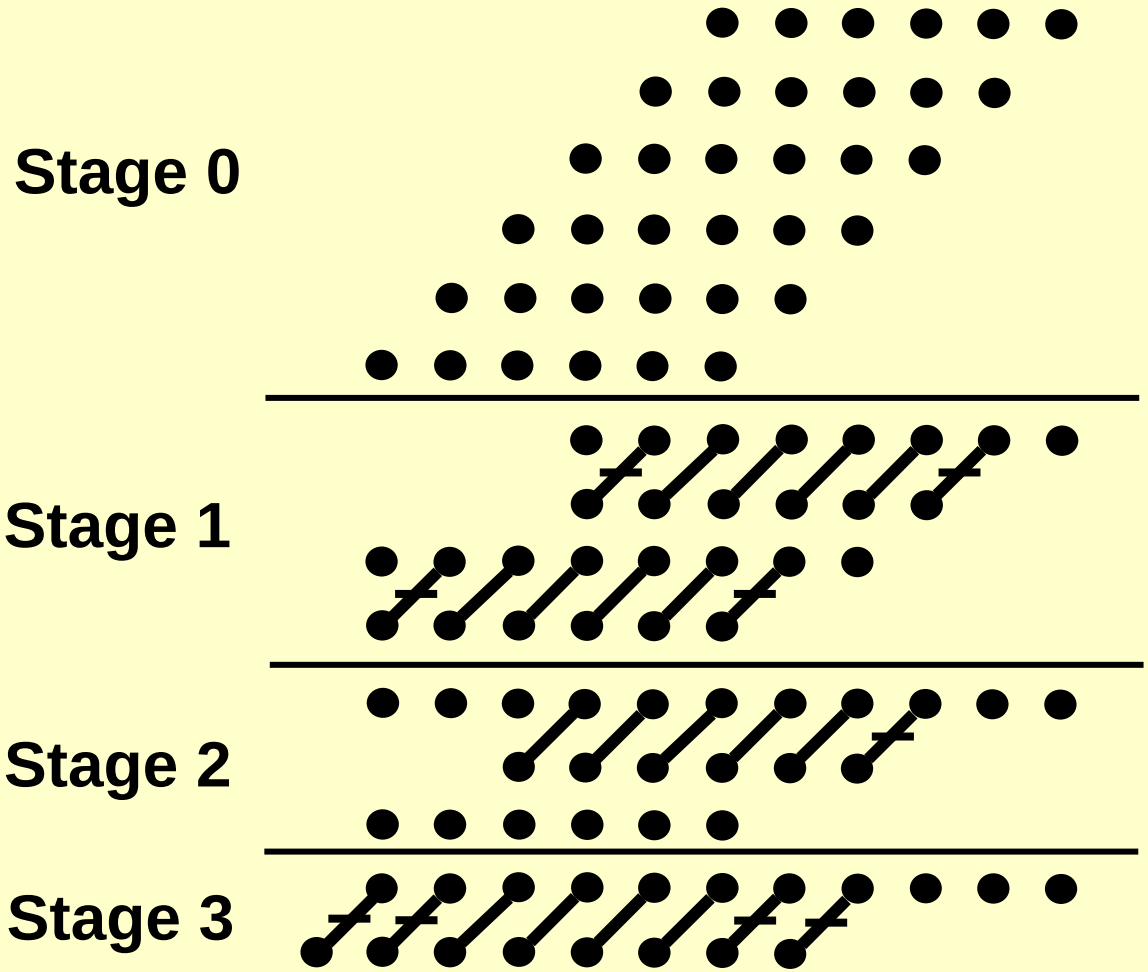
Block Diagram of N by N Multiplier



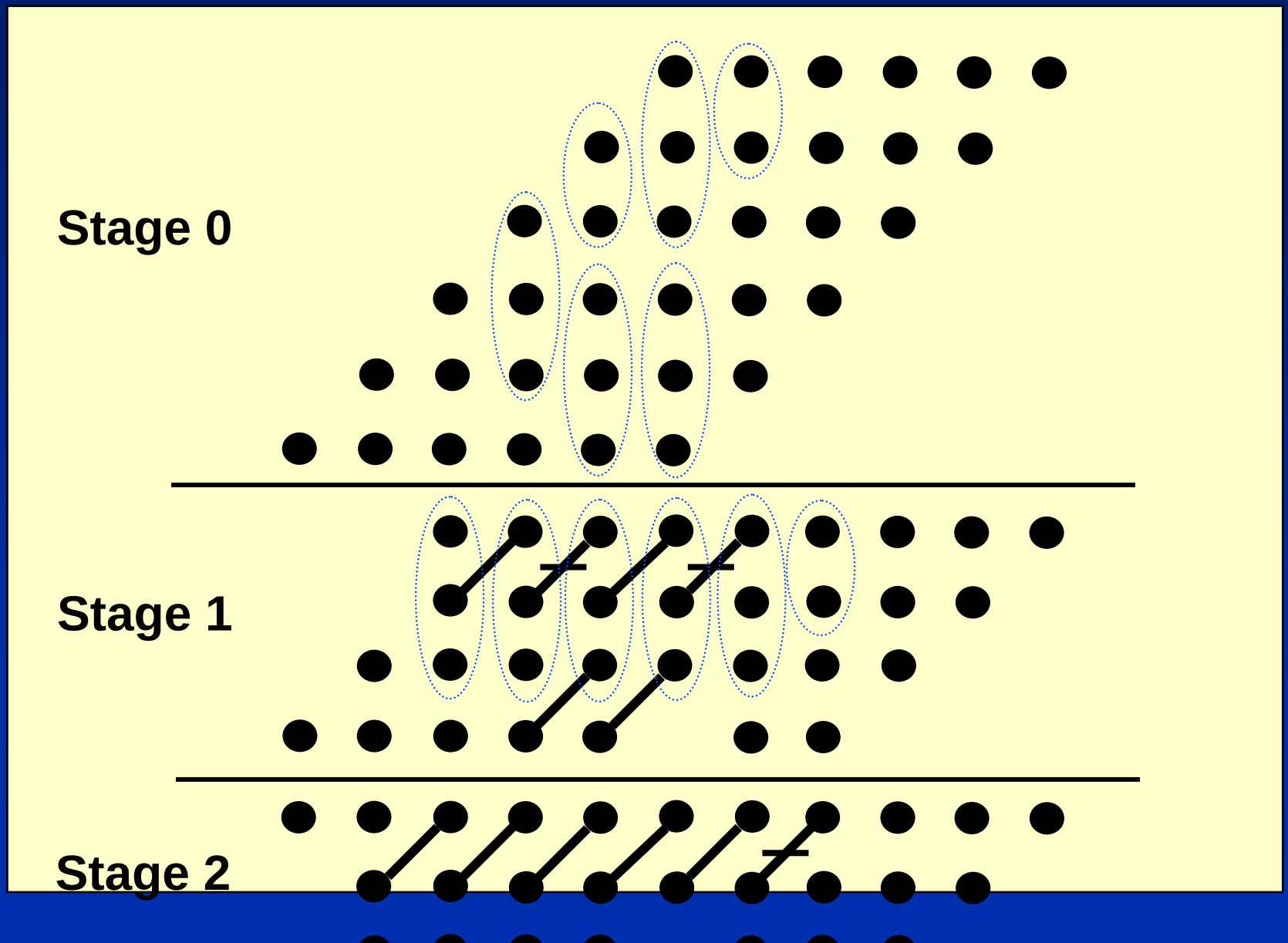
6 by 6 Wallace Multiplier



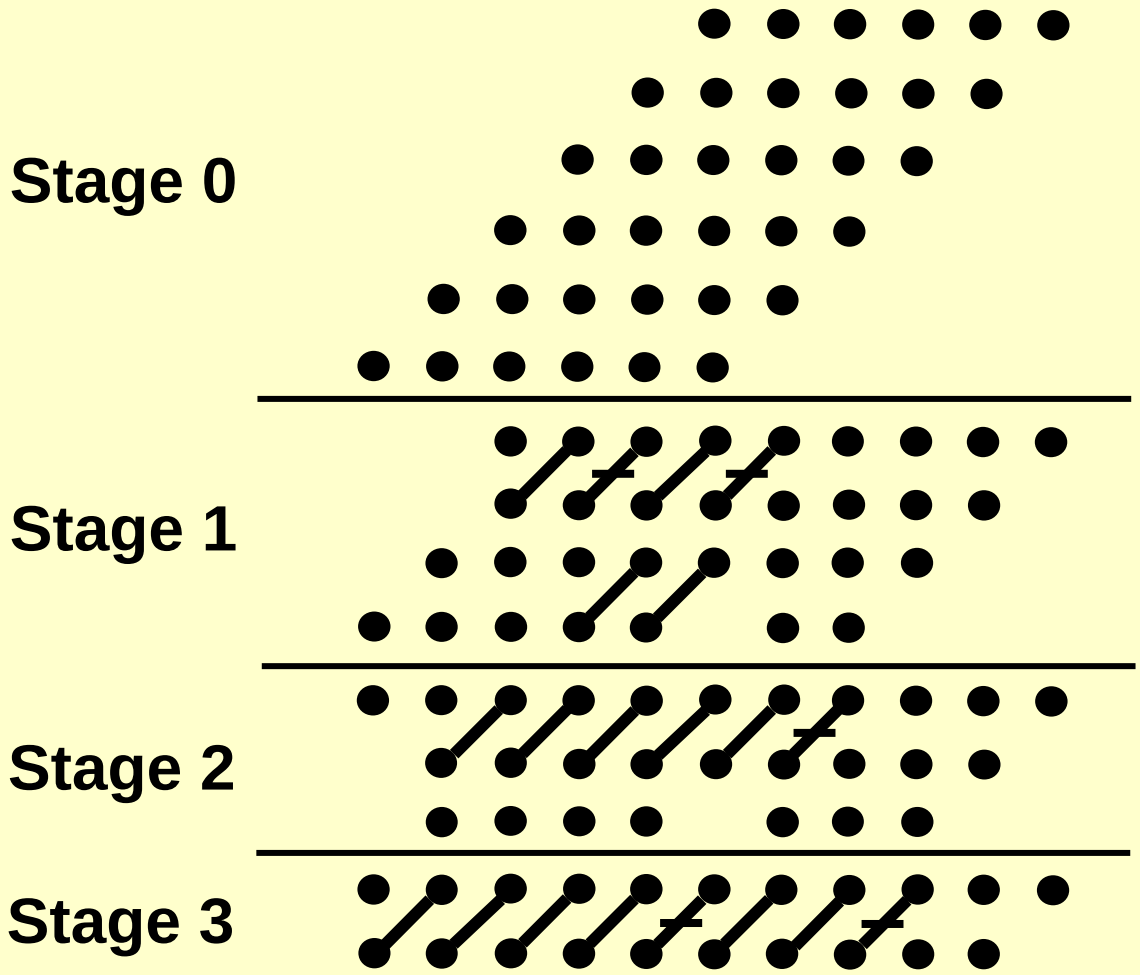
6 by 6 Wallace Multiplier



6 by 6 Dadda Multiplier



6 by 6 Dadda Multiplier



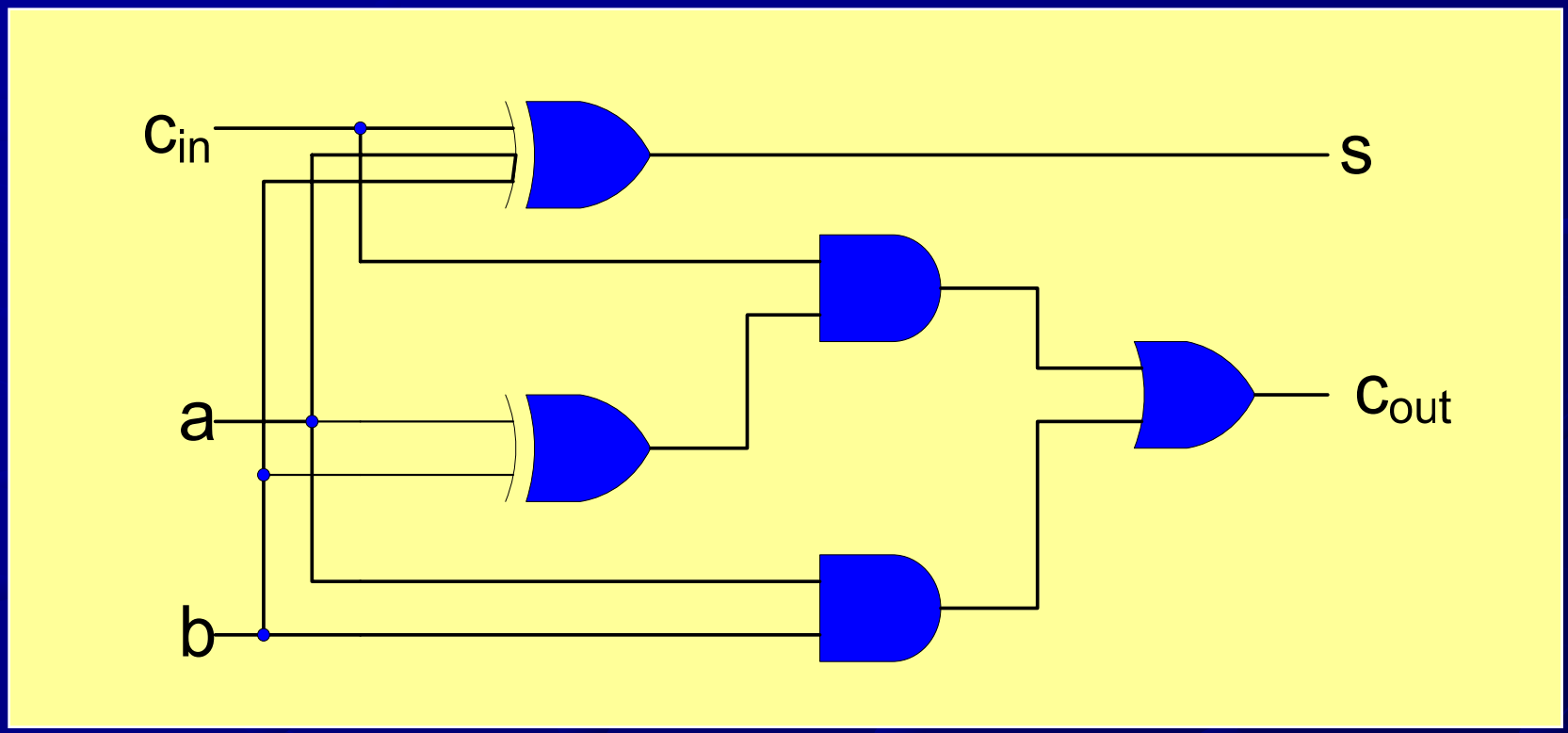
Multiplier Implementation

- Operand sizes:

8 by 8	32 by 32
16 by 16	64 by 64

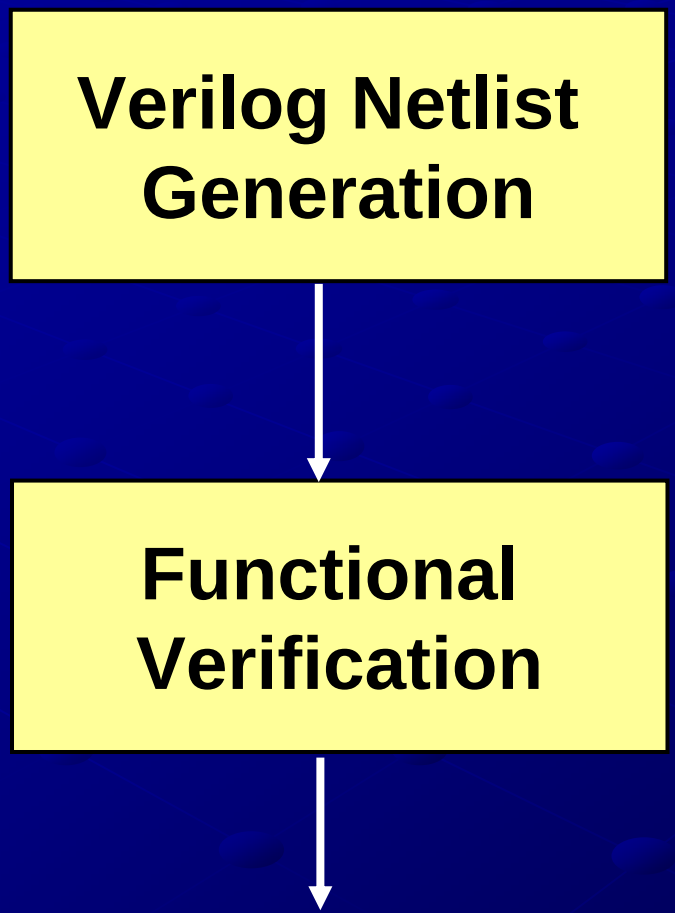
- Process Technologies (same foundry):
  - 250 nm, 5 metal layers, 2.5 V
  - 180 nm, 6 metal layers, 1.8 V
  - 130 nm, 8 metal layers, 1.2 V
  - 90 nm, 9 metal layers, 1.0 V
- Temperature: 25˚C
- Typical process models

Multiplier Implementation



Schematic of (3,2) counter

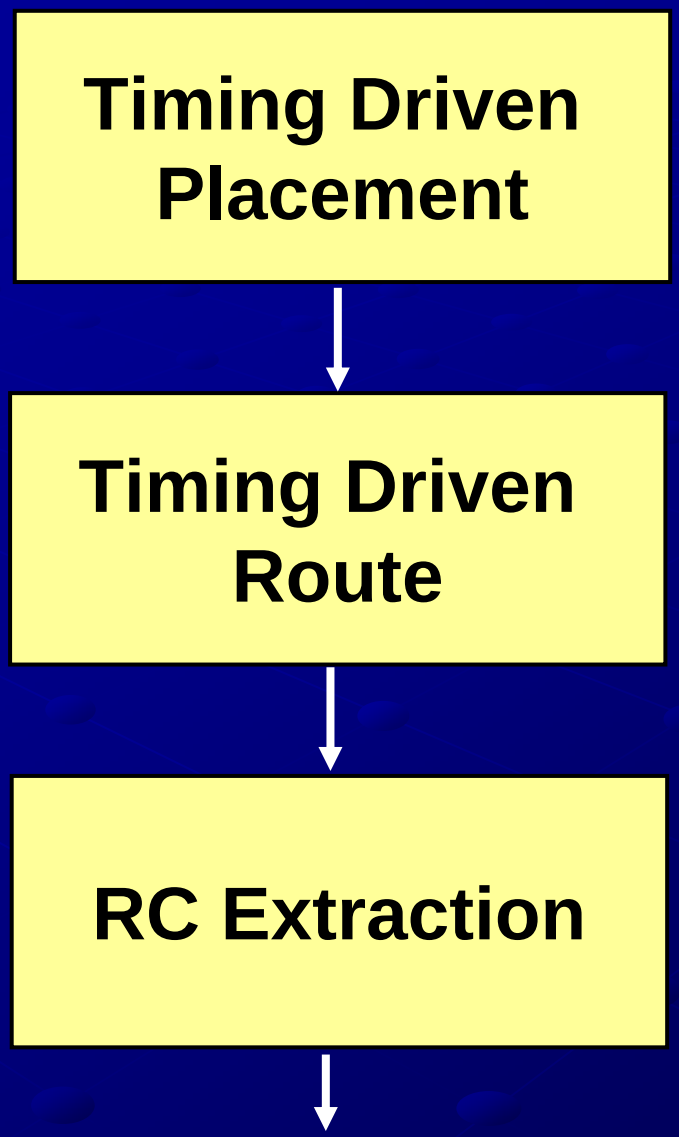
Design and Tool Flow (1)



genmult and spi2ver perl scripts

Conformal Equivalence Checking and Conformal Ultra

Design and Tool Flow (2)

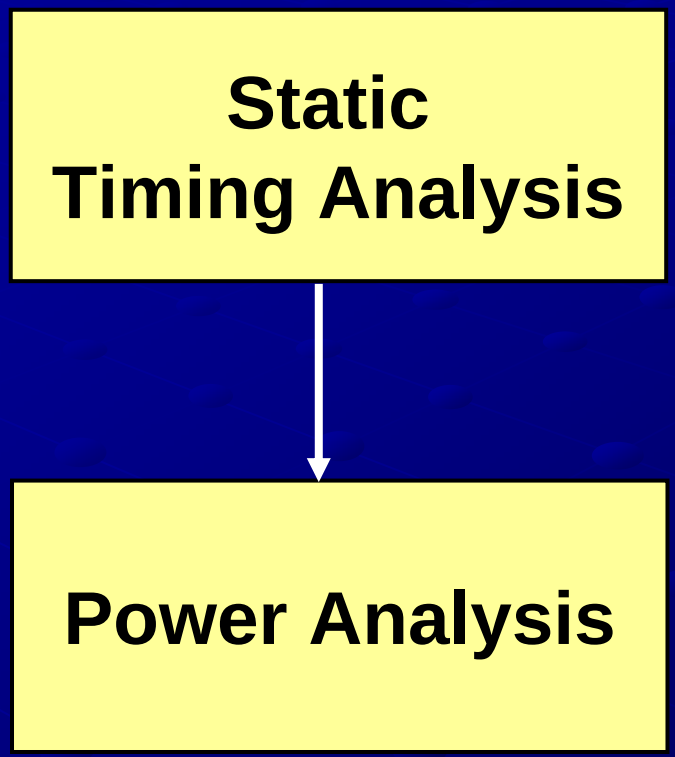


Encounter NanoPlace

Encounter NanoRoute

Encounter Native Extraction

Design and Tool Flow (3)



Encounter Common Timing Engine

Virtuoso UltraSim

Area Estimation

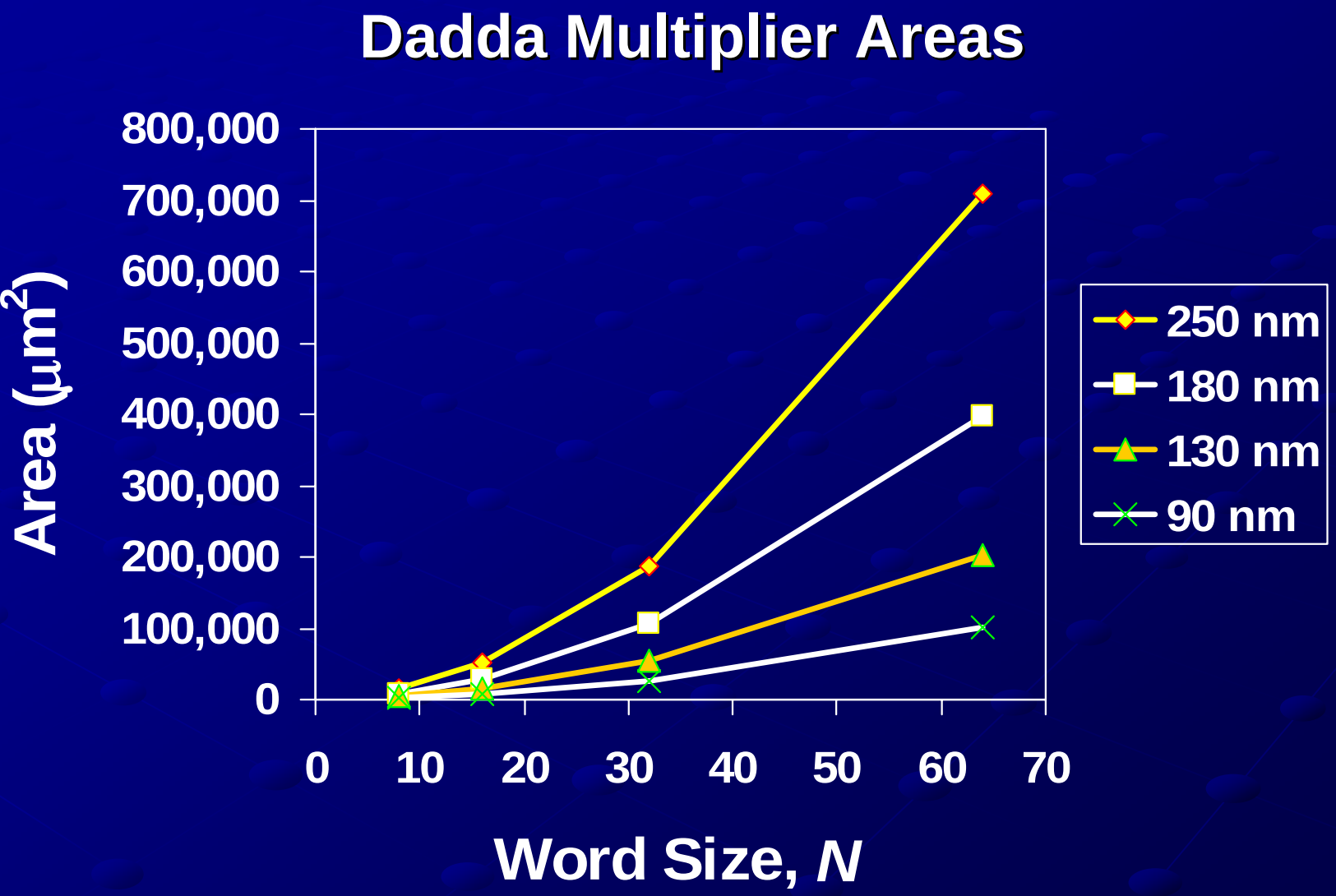
Method	(3,2) Counters	(2,2) Counters	CPA Length
Wallace	$N^2 - 4N + 2 + S$	$\geq N$	$2N - 1 - S$
Dadda	$N^2 - 4N + 3$	$N - 1$	$2N - 2$

where N = operand size and S = # of reduction stages

- **To a first order: Area proportional to $\lambda^2 N^2$**
- **Based on gate counts:**

  Wallace approx. 10% larger than Dadda

Area Measurements



Area Comparisons

- Area differences small (< 4%)
- For word sizes larger than 8 by 8, Wallace multipliers larger
- For 8 by 8 word sizes, Dadda multipliers larger due to CLA area

Area Analysis

- Area $\cong \alpha \lambda^2 N^2$ ← good rough estimate
- Area $\cong \alpha \lambda^2 N^2 + \beta \lambda^2 N$ ← Need N^2 and N terms
- For maximum error $\pm$ 7%:

$$\text{Area}_{Wallace} \cong 0.00283\lambda^2 N^2 + 0.015\lambda^2 N - 0.0472\,\lambda^2$$

$$\text{Area}_{Dadda} \cong 0.00275\lambda^2 N^2 + 0.0122\lambda^2 N - 0.0166\,\lambda^2$$

Delay Estimation

- $\text{Delay}_{N\text{x}N} = t_{\text{buffer}} + t_{\text{AND}} + S \cdot t_{(3,2)} + t_{\text{CPA}(2N-X)}$

  where

  - t_{buffer} = delay of the input buffers
  - t_{AND} = delay of a 2-input AND gate
  - S = number of reduction stages ($S \cong \log_{1.4} N$)
  - $t_{(3,2)}$ = delay of a (3,2) counter
  - $t_{\text{CPA}(Z)}$ = delay of a Z-bit carry propagate adder
  - $X = S+1$ for Wallace and $X = 2$ for Dadda

Delay Estimation

- With $S \sim \log N$ and $t_{CLA} \sim \log N$

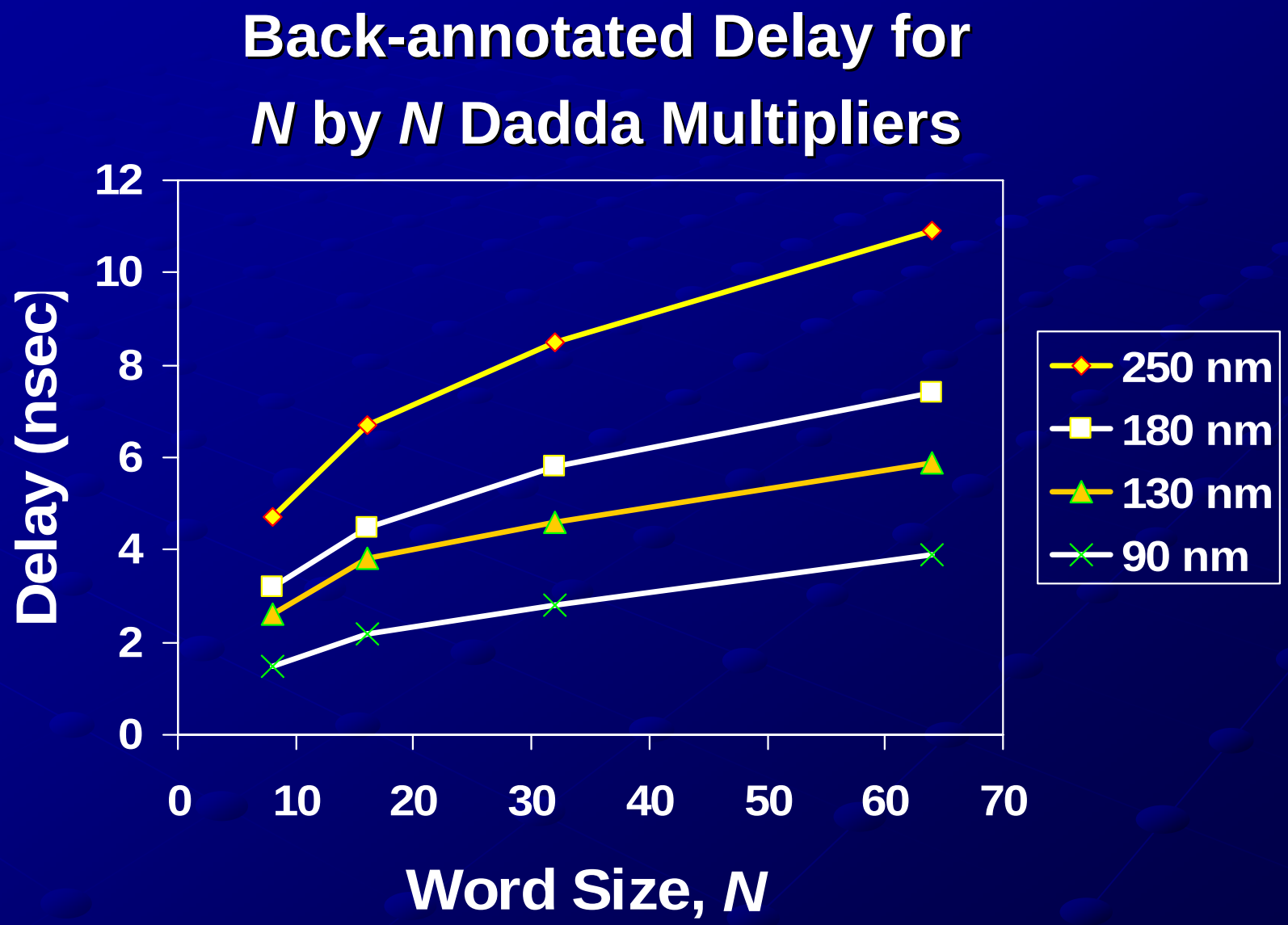
CC multiplier delay is proportional to log *N*.

- With λ = minimum process feature size
  - channel length and supply voltage ~ λ
  - gate delay ~ λ

CC multiplier delay is proportional to λ.

$$\text{Delay}_{N \times N} \approx k \lambda (\log N)$$

Delay Measurements



Delay Analysis

- Generally, $\text{Delay}_{\text{Wallace}} \cong \text{Delay}_{\text{Dadda}}$ (diff $\leq$ 4%)
- For 64 by 64 multipliers,
  Wallace slightly slower (diff $\leq$ 13%)

- For improved delay approximations:

$$\text{Delay}_{\text{Wallace}} \cong 0.0094\ \lambda \log_2(N) - 0.011\ \lambda$$

$$\text{Delay}_{\text{Dadda}} \cong 0.0082\ \lambda \log_2(N) - 0.0066\ \lambda$$

Conclusions

- For CC Multipliers:
  - Estimate Area using both N^2 and N terms
  - Doubling N increases Area by approx. 4x
  - For each generational transition in λ, Area reduces by 1/2.
  - Delays are approx. equal, for same N and λ
  - Estimate Delay using both log N and λ terms
- Consider using CC Multipliers when
  - time to $$ is critical
  - custom layout resources are limited
  - IP reuse or multiple foundry ports expected

Full Adder Evaluation and Selection for a Parallel Multiplier

Mike Spear

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
e-mail: spear@ece.utexas.edu

Gaurav Tuteja

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
e-mail: gtuteja@mail.utexas.edu

Earl E. Swartzlander, Jr.

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
e-mail: eswartzla@aol.com

Abstract- This paper presents an evaluation of CMOS full adders to optimize a Dadda multiplier with respect to delay and power consumption. An assortment of full adder cells from the literature are characterized for delay in HSPICE across voltages in both 65nm and 90nm technologies, and then static timing scripts are used to evaluate those cells in the context of 16-bit by 16-bit Dadda multipliers. Multipliers constructed from each cell across voltage and technology corners are simulated in HSPICE to evaluate average power, maximum power, and leakage. Finally, the best of these full adder cells are assembled in combination to produce a hybrid multiplier that achieves the best speed while also having very low power consumption.

I. Introduction

Wallace introduced the parallel multiplier with 3:2 counters [1]. Dadda elaborated on this idea and further enhanced the design [2]. Dadda's multiplier has the advantage of lower complexity in the reduction stages and more flexibility, which leads to lower delay. There are also several studies that evaluate the delay and power of individual CMOS full adder designs, e.g., [3]-[6].

However, there appears to be little work that evaluates the trade-offs of those full adders in the context of a parallel multiplier tree. Neither Wallace nor Dadda specified an ideal full adder design in their papers [1], [2]. Also, most studies of full adders have been done in test-benches outside of a larger network, where capacitive loads, glitches, cascading of circuits, etc., can lead to magnification of otherwise small flaws that lead to delay increases or a large amount of power consumption. Here, some of these full adders and their trade-offs are analyzed.

The 9-gate full adder is a standard choice, but there are others in the literature with lower transistor counts that may provide better results. However, the carry-save network in a multiplier has very different critical paths from the traditional carry chain in a large ripple carry adder. Evaluating these designs individually, then in the context of real multipliers, and finally in combination will demonstrate which full adders work well and why.

II. Approach

This analysis comprises three steps. First, an assortment of full adders is characterized. The second step involves building Dadda multipliers using the adders and evaluating their power consumption. Finally, the overall fastest full adder is combined with the one that has the lowest power consumption to build an optimized hybrid tree.

A. Full Adder Characterization

For this paper, a total of 6 adders were considered. They are as follows:

1. *fa14trans* – this adder has 14 transistors. It is low power and uses XOR and XNOR functions to generate the outputs. [5]
2. *fa10trans2* – this adder has 10 transistors. It has a relatively faster method of generating the XOR and XNOR functions than fa14trans. [5]
3. *fa10trans* – this adder has 10 transistors and is low power. It is recommended for use in embedded architectures. [4]
4. *fa12trans* – this adder has 12 transistors and uses multiplexers to generate the outputs.
5. *fadder9* – this is the standard 9 gate full adder, consisting of NAND2, NOR2 gates and inverters.
6. *fadder12* – this is the standard 12 gate full adder, consisting of NAND2, NAND3, NAND4 gates and inverters.

Diagrams for each of these are shown in Figures 1-6. The half adders (standard and modified) used for this paper are shown on Figure 7. Both of the half adders are standard implementations using NOR2, NAND2 and inverters. The transistor-level adders (*fa10trans, fa10trans2, fa12trans, fa14trans*) needed to have buffers added to the sum and cout outputs to allow for acceptable drive strength and a more smooth and glitch-free curve as shown by Figure 8.

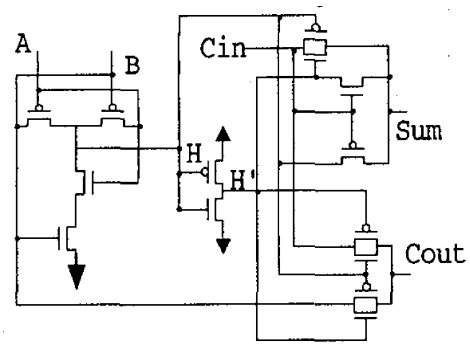


Fig. 1: Diagram of fa14trans

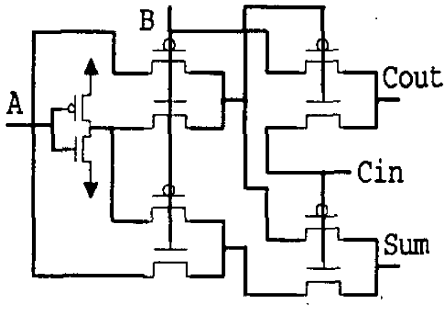


Fig. 2: Diagram of fa10trans2

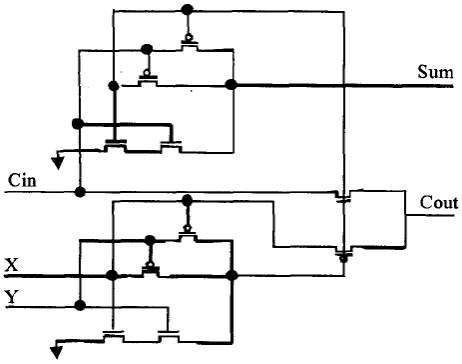


Fig. 3: Diagram of fa10trans

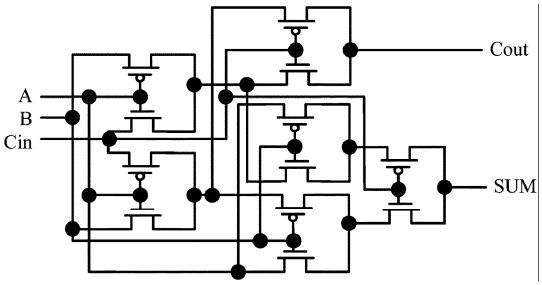


Fig. 4: Diagram of fa12trans

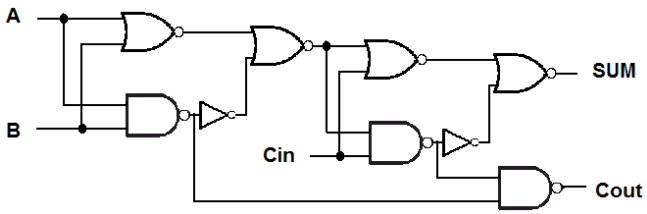


Fig. 5: Diagram of fadder9

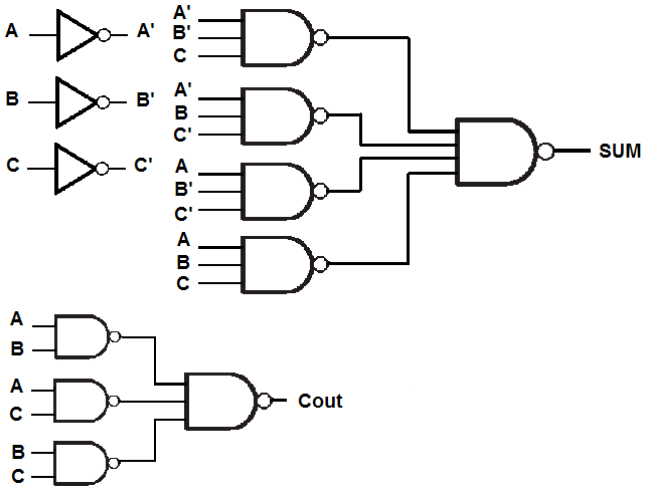


Fig. 6: Diagram of fadder12

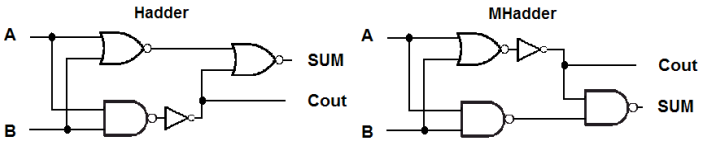


Fig. 7: Diagrams of the half adders

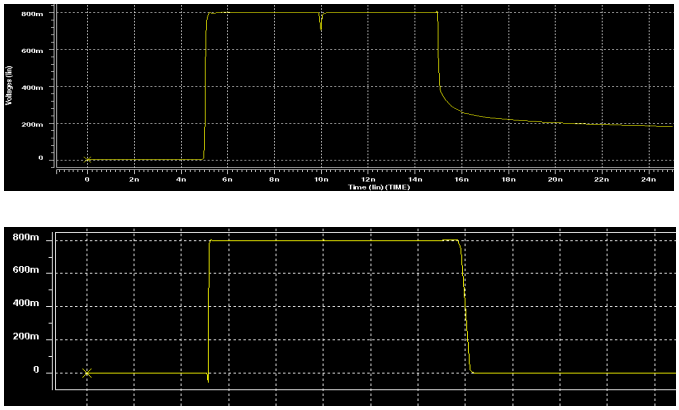


Fig. 8: Curves of a non-buffered (above) and buffered (below) outputs

The first step involves characterizing the delay of the six full adders across two technologies (65nm and 90nm) and three voltages (0.8 V, 1.2 V, and 1.6 V). In order to accomplish this, there was a single SPICE test bench for each circuit, and this test bench was run across all the voltages and the two technologies. Parameter details are given below:

a) Load: The sum and carry outputs of each adder are loaded in one run with a minimum sized 2/1 inverter and in another run, a 10/5 inverter.

b) Input Combinations: A total of 24 input combinations are used to characterize each adder for delay and power. This is because there are a total of 3 inputs, keeping two inputs static 1 or 0, and switching one input to either rising (0->1) or falling (1->0).

c) Input Capacitance: There is a separate analysis to calculate the input capacitance of each input for each adder. To accomplish this, minimum-sized inverters are used to drive each of the three inputs of the full adder and delay data is calculated. Since the delay of a minimum-sized inverter driving itself was already available, a simple linear model of capacitance and delay allowed for the calculation of each input capacitance.

d) Maximum power: The maximum power from each run was stored.

B. Tree Generation and Evaluation

To evaluate the full adders from the section above properly, it was necessary to do so in the context of a real multiplier tree such as that shown on Figure 9. This allowed for weaknesses due to drive strength, capacitive loading, and spurious glitches to be easily exposed. These factors can only be measured when several of the circuits are combined in a carry-save framework. Therefore, the full adders were used to build 16-bit by 16-bit Dadda multiplier trees, and dynamic power consumption (at 1 GHz) and leakage of those multiplier trees were characterized in HSPICE.

i) Tree Synthesis

To assist in the creation of the Dadda trees, a script is used to generate a tree given a size N-bit by N-bit and a full adder cell to use. The trees have the nice property of being fairly easy to synthesize, since the number of rows and the height of each row in the dot diagram matrix can be predetermined as shown on Figure 10. The arrival time of each "dot" in the tree is computed using the delay and input capacitive loading data from the full adder characterization.

The only ambiguousness that had to be considered was

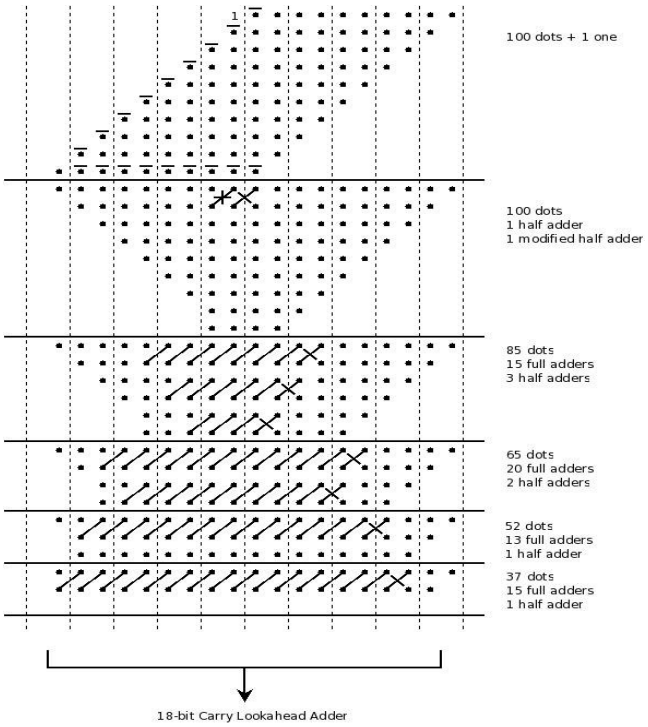


Fig. 9: Example 10-bit by 10-bit Dadda multiplier tree dot diagram

how to determine which dots in a given row and column of the dot matrix are applied to the various full adder inputs. To accomplish this, at each level the script sorts the dots in a column in order of ascending delay. Of course, the delay

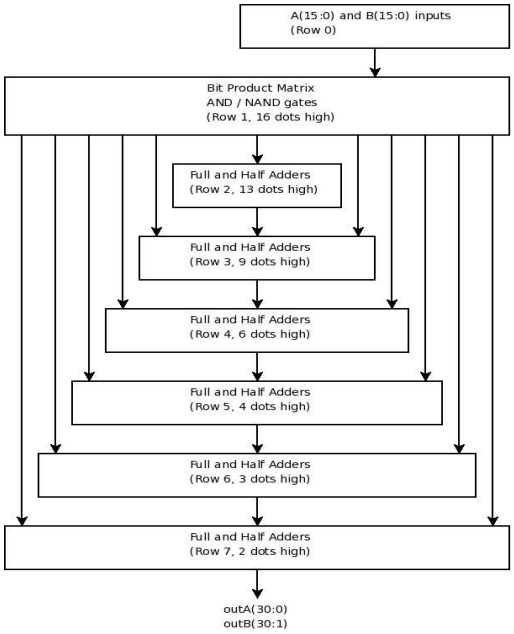


Fig. 10: 16-bit by16-bit Dadda multiplier block diagram

is a function of the capacitance load, which requires knowing which dots are assigned to which inputs in the next row, but for a good approximation, the loads are calculated assuming that the downstream pin is the "B" pin of the full adder. After sorting the dots, each group of three dots is assigned to a full adder and the latest-arriving dot among the three is assigned to the carry input. Consider the following example program dump:

```
r4c12
delay       out      in
143.73      1c       1a
163.09      1s       1b
188.28      2c       1c
265.53      3c       2a
265.53      2s       2b
284.89      3s       2c
```

In this example, the dots of row 4, column 12 are shown for a 16-bit by 16-bit multiplier, where row 0 contains the original A and B input bits, and row 1 contains the bit product matrix. There are six "dots" in this example column, each of which consists of three fields: "delay", "out", and "in", where "delay" is the worst-case arrival time for that signal, and "out" and "in" correspond to full adder instances and pin labels. For example, the "in" fields 1a, 1b, and 1c mean that those three dots are consumed by full adder "1" inputs A, B, and C, and the "out" fields 1c and 1s mean that those dots are produced by the column 11, adder "1" output COUT or the column 12, adder "1" output S in the previous row.

When the program reduces the matrix to two rows, it terminates. It constructs a net list data structure that can be passed over to a static timer module, described in the next section. This net list can also be passed to an HSPICE file for use in power measurement test-benches. Because full adder delays are sensitive to both transistor length and voltage, a different multiplier was synthesized for each corner (65 and 90 nanometer technologies, and 0.8, 1.2, and 1.6 voltages).

ii) Static Timing

The trees are then analyzed with a static timing script. Like in the tree synthesis step, characterization data from the full adders is employed to calculate the worst-case delay to every node in the tree. Static timing has the advantage of determining the worst-case delay very quickly, whereas it was unclear which input stimuli would have to be tested in HSPICE to determine that delay, and of course it would be very slow. The disadvantage of using the static timer is that it can be pessimistic, because it may account for combinations of transitions that in fact cannot happen, but for the purposes of comparison here it met the needs of this analysis.

Unlike during tree synthesis, capacitive loads at each node could be accurately computed because all of the matrix rows are already available. Input pin capacitances are known for each full adder, and the delay characteristic for each output is linearly fit by the timer to the observed load. The load approximations from tree synthesis only caused the expected result to vary from the static timer result by 1-2% in most cases. Wire capacitance is not considered, nor is any wire load estimation (this would certainly be a subject for future work).

The inputs to the static timer are a set of input Arrival Times (ATs), a set of output Required Arrival Times (RATs), and a set of output load capacitances. The input ATs are all assumed to be zero. The output RATs are not important for tree synthesis, but are important in the tree optimization step to determine non-critical paths (discussed in more detail in the "Optimization" section below). The output loads are assumed to be three minimum-sized inverters, which approximate the input to a 9-gate full adder.

iii) Power Measurement

Power measurement is performed using HSPICE on the net list outputs of tree synthesis, as described above. Three types of power consumption are analyzed: leakage, average dynamic power, and maximum dynamic power. Two spice test-benches are created -- one to test for leakage and one to test for dynamic power. As before, data is collected for 65 and 90 nanometer technologies, and 0.8, 1.2, and 1.6 voltages, from multiplier trees that are tailored to those parameters.

Leakage is calculated by making a series of runs with randomized static inputs, measuring the average power for those runs, and then averaging the average powers across the runs. Multipliers may spend a lot of time in standby mode, so leakage is a very important metric. Unfortunately these runs took almost a full hour to complete for each 16-bit by 16-bit multiplier, with only ten sets of input stimulus, so this analysis was unable to produce any meaningful data for 32-bit by 32-bit or 64-bit by 64-bit multipliers, even though their synthesis and timing ran in only a few seconds. It is desirable to have more than ten runs, but the existing results are precise enough to allow relative comparison.

Dynamic power is calculated by randomly switching inputs on cycle boundaries for a number of cycles and measuring both the average and maximum power dissipation. Average power dissipation is a good metric for energy consumption, which is important for battery life, while maximum or peak power dissipation is a packaging constraint. The cycle clock period was chosen to be one nanosecond, based on both the fastest static timing results at 0.8V and 90nm, and the observed output transitions in an HSPICE waveform. Again, this analysis was limited to ten sets of inputs on only 16-bit by 16-bit multipliers because the HSPICE simulations were quite slow. Dynamic power would have been especially nice to measure for 32-bit by 32-bit and 64-bit by 64-bit multipliers in order to expose the effect of cascaded glitches, but these runs failed to complete (this is another

interesting subject for future work).

C. Optimization

After determining the fastest full adder and the one with the lowest power, the next step is to design a hybrid tree that combines the two to achieve the best delay at the lowest power possible. In most designs, many of the gates are not on a critical timing path, and these Dadda multiplier trees are no exception. Clearly, the central columns in the matrix tend to have the tightest timing, while the columns on the left and right arrive relatively early, so there are definite areas of which advantage could be taken.

For multiplier trees, in general, the plot of output arrival times against column numbers conforms roughly to a piece-wise linear function with three sections [7]. As shown in the arrival time plot of Figure 11, in the first section, where column numbers are small, the arrival times appear to rise linearly with respect to the column number. In the second section, among the center columns, the arrival times plateau. Finally in the third section, where column numbers are high, the arrival times appear to fall linearly with respect to column number. Therefore, the overall appearance is trapezoidal.

The early and staggered arrival times of the lower columns can be taken advantage of by implementing that segment of the final adder stage as a ripple-carry adder [7]. It is not appropriate to remove that advantage when selecting critical paths. However, the upper-order columns need not arrive earlier than the latest arriving central columns. Therefore, the overall desired output arrival time characteristic would rise in the lower order columns and plateau across the central and upper columns as shown on the required arrival time plot of Figure 11.

Using the static timer described above, the RAT input file was tailored to test against this desired output characteristic. The timer script then reported RAT minus AT ("slack") for each node in the tree, where positive values indicated areas where fast full adders could be replaced by low power adders. Specifically, a full adder could be swapped if it could be replaced with the low power version, re-timed, and still not create a negative slack. The order of selection was chosen such that swapping full adders closest to the bottom row and center column would have preference over those that were further away as shown on Figure 12. The motivation for this ordering was that full adders on the bottom row and center column should be more likely to see glitches on their inputs, so replacing those first would save more dynamic power.

Some final power optimization touches were put into the hybrid tree in the form of pin swapping and transistor stack tapering. For gates with transistor stacks, specifically the standard CMOS gates in the fadder9, arranging for the later-arriving signals to drive the gates of stack transistors

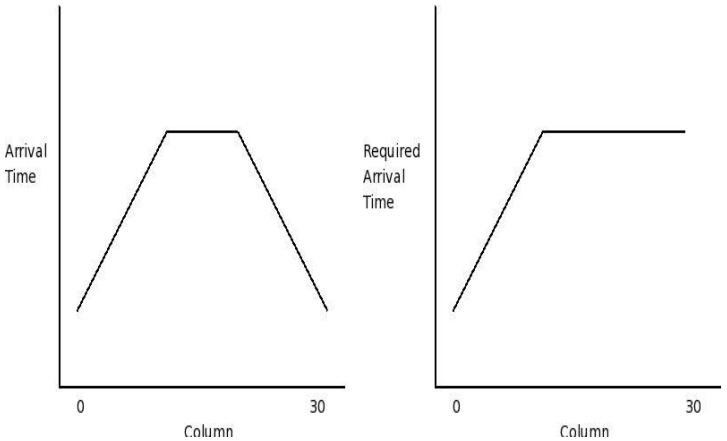


Fig. 11: Real arrival time and desired arrival time delay characteristics

that are closest to the outputs should reduce dynamic power consumption by reducing glitches [8]. In this case, since the "B" input of all of the full and half adders drives the bottom of a stack, and the "A" input drives the top, "B" was swapped with "A" throughout the tree. Also, sizing the transistors in those stacks so that the transistors close to the power supply are wider should reduce power consumption [8], so this was done as well.

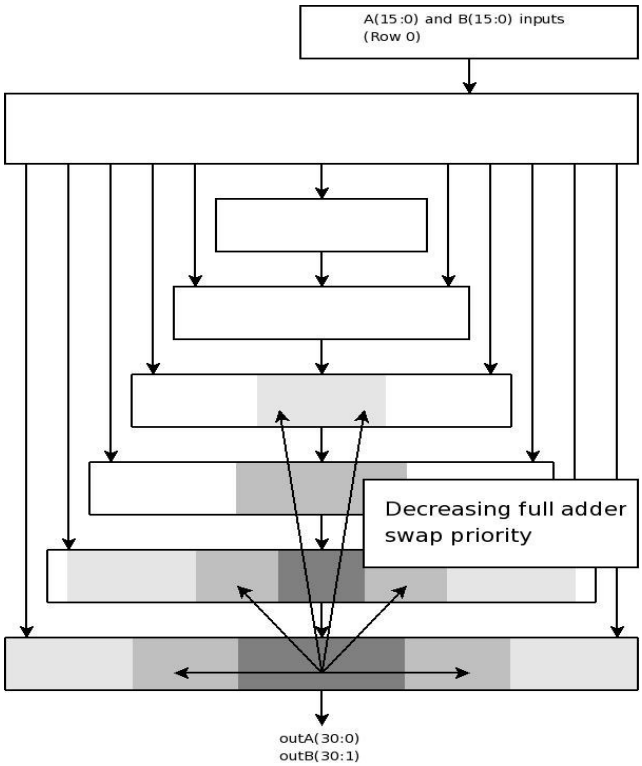


Fig. 12: 16-bit by 16-bit Dadda multiplier optimization: block swapping precedence

III. Results

A. Full Adder Characterization

Figures 13 and 14 show the delays of adders to Cout and sum for each adder in 90nm technology. Overall, the fastest adders were *fa14trans, fadder9* and *fadder12*. Neither of the 10 transistor adders worked for the 0.8V configuration. The transistor models were picked up from [9].

Full adders *fa10trans, fa10trans2,* and *fa12trans* all suffered from one or more "V_{th}-drops," where a single pfet or nfet drove an output node. This phenomenon occurs when a pfet pass transistor passes a logic ZERO or an nfet pass transistor passes a logic ONE by itself. The resulting output signal is not at ground or Vdd, but is in fact at the voltage threshold of the transistor, or Vdd-V_{th}. This inhibits the ability of the output nodes to switch downstream transistors, and effectively manifests itself as added delay. In this case, at nominal and low voltages, the effect appears to cripple the performance of fa10trans, fa10trans2, and fa12trans adders. Usually the solution to this is to implement pass transistors with a pfet and nfet pair, but of course this requires both an extra pass transistor and the complement of the select signal.

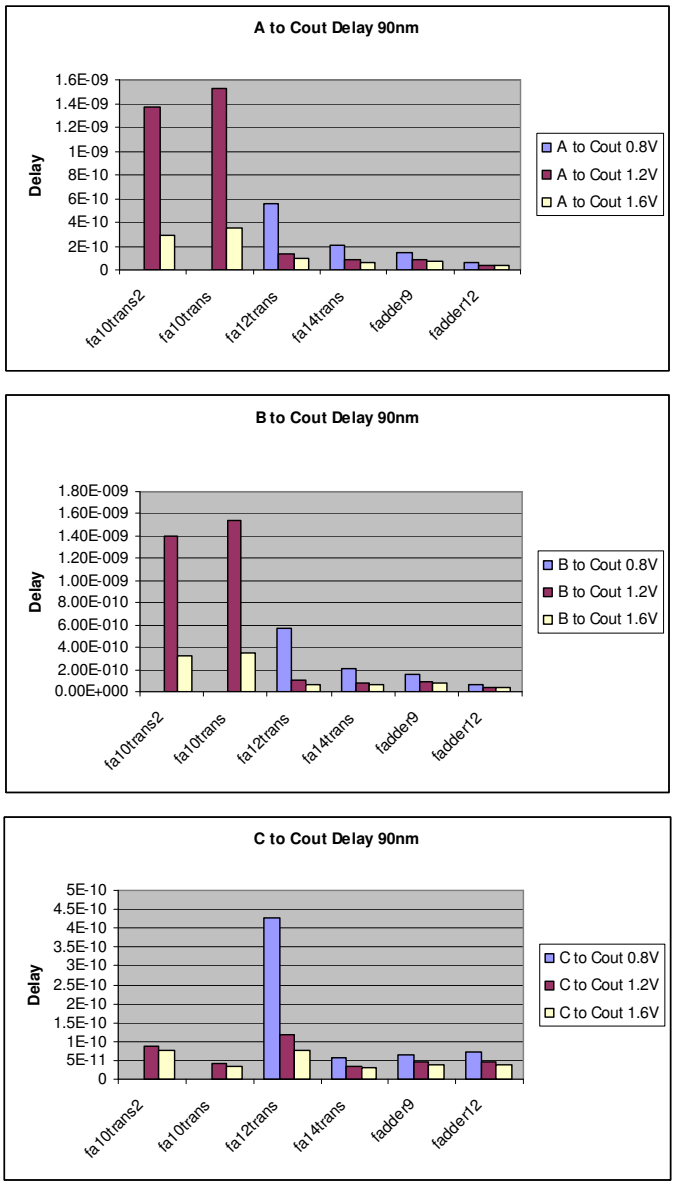


Fig. 13: Full adder Cout delays at 90nm

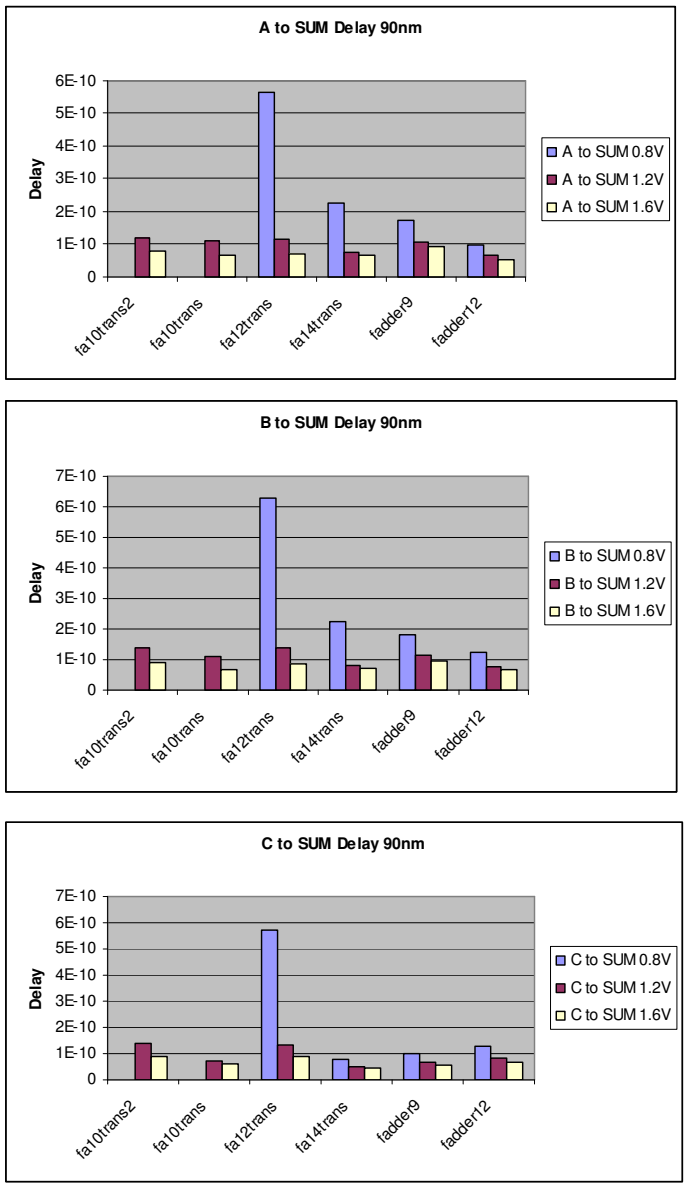


Fig. 14: Full adder SUM delays at 90nm

B. Tree Generation and Evaluation

i) Delay

As shown on Figures 15 and 16, the trees constructed with *fa14trans* are the fastest at 1.2V and 1.6 V in both 90nm and 65nm processes, while trees using *fadder12* are the fastest at 0.8V, with *fadder9* being a close second at that same voltage. The *fa10trans*, *fa10trans2*, and *fa12trans* trees were extraordinarily slow at low voltages. Apparently, the *fadder12* and *fadder9* implementations with standard CMOS gates were more resilient at low voltages, although the *fa14trans* had respectable performance at 0.8V. *fadder12* barely outperformed the *fadder9* in practice, even though it was much faster when characterized in a standalone test-bench, because of its high input capacitance.

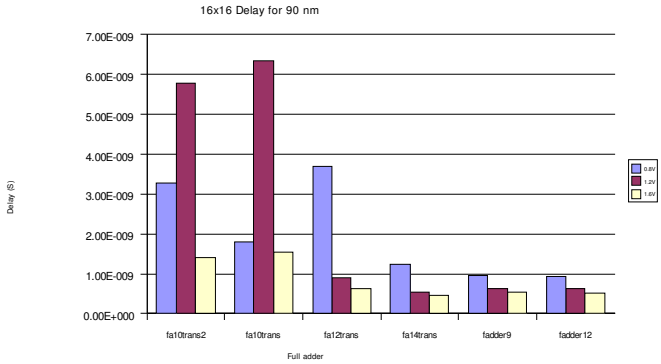


Fig. 15: Tree delays for 90nm

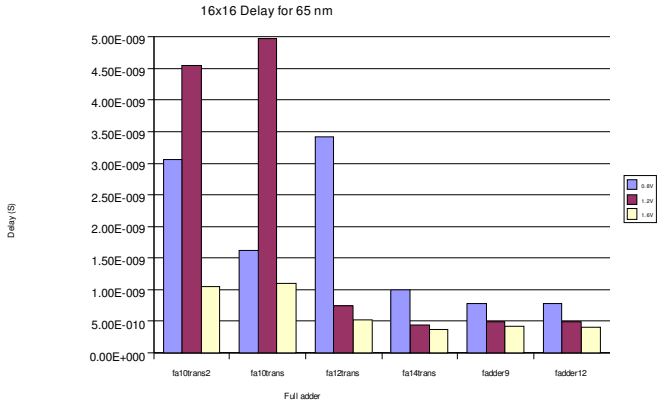


Fig. 16: Tree delays for 65nm

ii) Power

Figures 17 through 22 show the average power consumption, the maximum power consumption and the leakage power for reduction trees constructed with the various full adders with the 65nm and 90nm technologies. The results show that trees implemented with *fadder9* have the lowest average power, maximum power, and leakage at every voltage and process. The fadder12 implementation also has very low leakage, which indicates that the standard CMOS gates are very robust against leakage. However, *fadder12* has a very high dynamic power usage due to its large input capacitance, and since leakage is such a small component of the total power consumption at 0.8V and 1.2V, it is not a desirable full adder to use for low power. The *fa14trans* implementation, on the other hand, has decent leakage performance at 0.8V and 1.2V, and was second best for dynamic power usage.

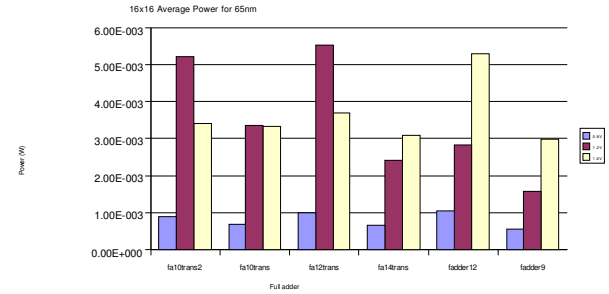


Fig. 17: Average Power for 65nm

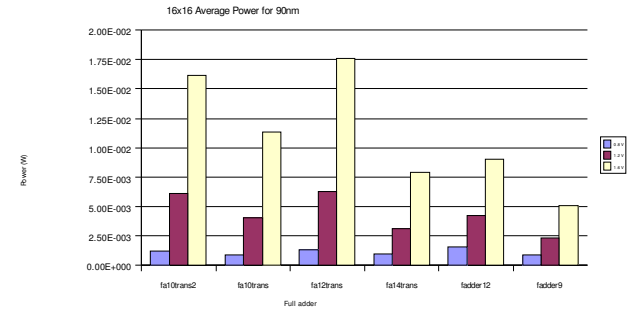


Fig. 18: Average Power for 90nm

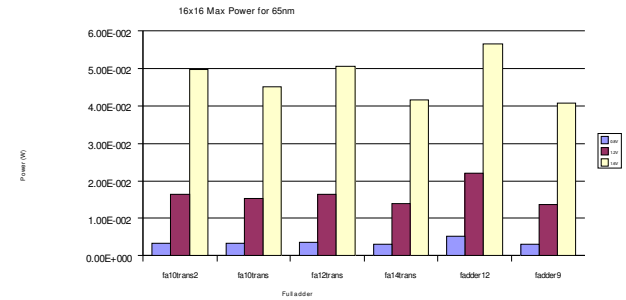


Fig. 19: Max Power for 65nm

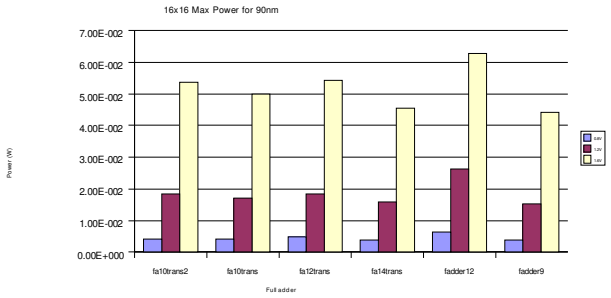


Fig. 20: Max Power for 90nm

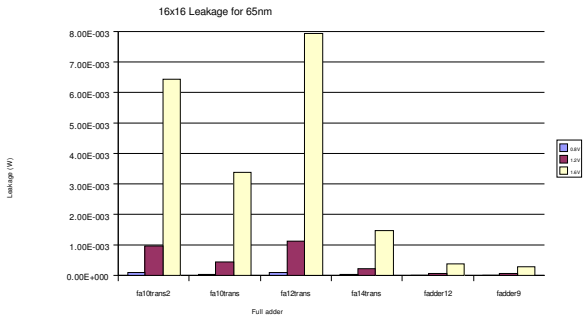


Fig. 21: Leakage for 65nm

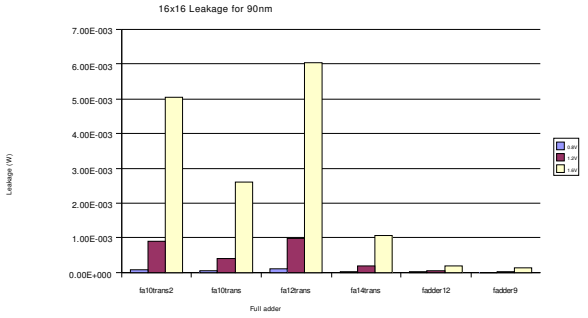


Fig. 22: Leakage for 90nm

iii) Optimization

From the results above, it was determined that the *fa14trans* full adder is the fastest at nominal voltages, and *fadder9* consumes the lowest power. Therefore the former is used to construct the framework of a hybrid tree and the latter is used in non-critical paths for low power. The difference between the actual output arrival times and the expected output arrival times shown on Figure 23 indicates that there is considerable opportunity for swapping, mostly because the required arrival time curve is piecewise linear while the real output arrival times have a much smoother slope transition, which leaves a rather large slack gap in the lower-middle columns. Of course, there was plenty of slack in the upper order columns as well.

The results of the hybrid tree optimization step are attractive. The result as shown on Figures 24 and 25 is a tree that has dynamic and leakage power results that are quite close to the low power *fadder9* tree (25% lower than the *fa14trans* tree), while achieving the same delay as the *fa14trans* tree. Table 1 shows which groups of columns required the most fast adders. As expected, the columns that have the biggest timing slack received the most power benefit. Note that the critical columns, 15-19 and 5-9, receive the least benefit but still only required less than 40% of their adders to be fast full adders to accomplish the same timing requirement. Also, as shown on Figure 26, the new hybrid tree's output arrival time curve approaches the required arrival time curve in the less critical columns.

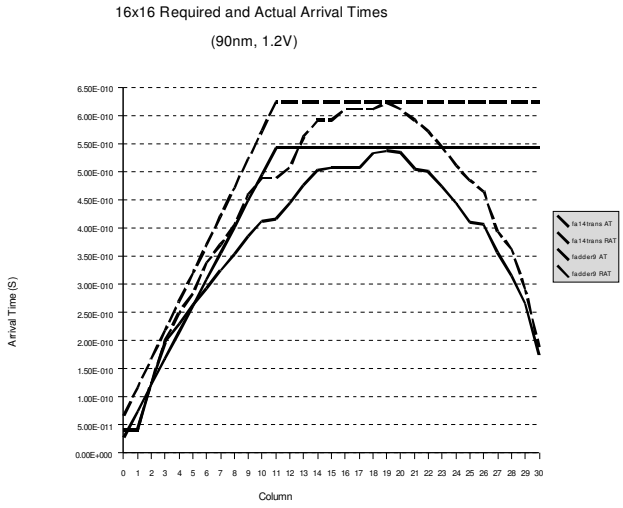


Fig. 23: Fast and Low Power Arrival Time Curves

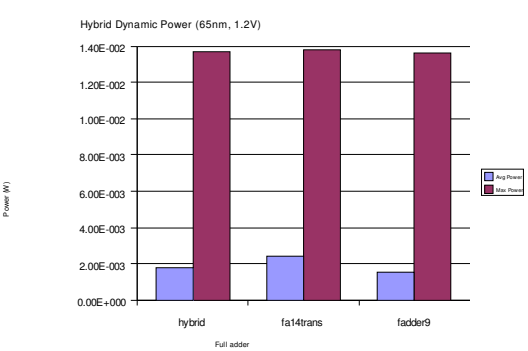


Fig. 24: Hybrid Tree Dynamic Power Comparison

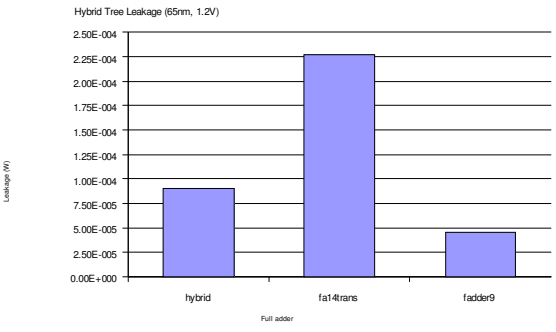


Fig. 25: Hybrid Tree Leakage Comparison

Table 1: Distribution of full adders in hybrid tree by multiplier column (65nm, 1.2V)

Column group	*fa14trans*	*fadder9*	*% fadder9*
0-4	3	0	0.0
5-9	9	16	64.0
10-14	11	39	78.0
15-19	23	39	62.9
20-24	7	33	82.5
25-30	0	15	100
Total	53	142	72.8

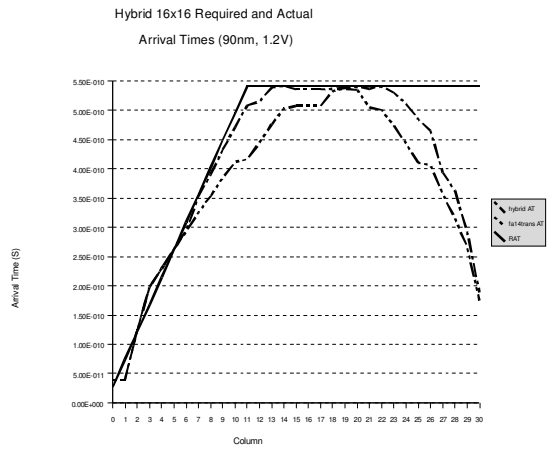


Fig. 26: Hybrid Tree Output Delays

The pin swapping and transistor stack tapering experiments proved less successful. Swapping signals going into the "A" and "B" pins throughout the Dadda tree did have a minor average power effect (6.0% for 65nm at 1.6V), but a negligible effect on maximum power, leakage, and delay. Transistor stack tapering initially increased dynamic power consumption when the tapering was applied such that stack resistance was preserved but a small gate capacitance penalty was paid. Applying the tapering such that gate capacitance was preserved had a negligible effect on the power consumption, but of course would probably increase delay due to the driver resistance penalty. In short, the pin swapping did have a small beneficial effect and was simple enough to implement, but the transistor stack tapering had no good effect and may be detrimental.

IV. Conclusions

Determining the best full adder to use in a Dadda multiplier is largely dependent on the application. Of the six full adders that were analyzed, three of them, *fadder9*, *fadder12*, and *fa14trans* all had unique advantages. For fast operation and relatively low power consumption at nominal voltages, *fa14trans* is a good choice. If dynamic power is not a concern, but robustness across a large voltage range is desired, then *fadder12* is a good choice for fast operation and relatively low leakage. For good speed and low power consumption at any voltage, *fadder9* performed very well. Paradoxically, the three full adders with the most transistors consumed the least power and had the least delay.

Using a heterogeneous set of full adders in a Dadda multiplier can have benefits as well. Combining the *fa14trans* and *fadder9* full adders in one multiplier can take advantage of non-critical paths to retain the speed of the former while approaching the power consumption of the latter. The results show that constructing such a hybrid multiplier can save about 25% of the average power consumed in a design consisting of only *fa14trans* full adders, and can save quite a bit in leakage power as well.

References

[1] C. S. Wallace, "A suggestion for fast multipliers," *IEEE Transactions on Computers*, vol. EC-13, pp. 14-17, Feb. 1964.

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

[3] H. Bui, Y. Wang and Y. Jiang, "Design and Analysis of Low-Power 10-Transistor Full Adders Using Novel XOR-XNOR Gates," *IEEE Transactions on Circuits and Systems -- II: Analog and Digital Signal Processing*, vol 49, no. 1, January 2002.

[4] A. A. Fayed and M. A. Bayoumi, "A Low Power 10-Transistor Full Adder Cell for Embedded Architectures," *IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 226-229, May 2001.

[5] L. Junming, S. Yan, L. Zhenghui and W. Ling, "A Novel 10-Transistor Low Power High-Speed Full Adder Cell," *International Conference on Solid-State and Integrated-Circuit Technology*, vol. 2, pp. 1155 – 1158, Oct 2001.

[6] Y. Jiang, A. Al-Sheraidah, Y. Wang, E. Sha and J. Chung, "A Novel Multiplexer-Based Low-Power Full Adder," *IEEE Transactions on Circuits and Systems – II: Express Briefs*, vol. 51, no. 7, July 2004.

[7] V. Oklobdzija and D. Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," *IEEE Transactions on VLSI Systems*, vol 3, no. 2, June 1995.

[8] E. de Angel and E. E. Swartzlander, Jr., "Survey of Low Power Techniques for VLSI Design," *Eighth Annual IEEE International Conference on Innovative Systems in Silicon*, pp. 159-169, Oct 1996.

[9] http://www.eas.asu.edu/~ptm , Related publications include:

- A. Balijepalli, S. Sinha and Y. Cao, "Compact modeling of carbon nanotube transistor for early stage process-design exploration," *ISLPED*, 2007.
- W. Zhao and Y. Cao, "New generation of Predictive Technology Model for sub-45nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816-2823, November 2006.
- Y. Cao, T. Sato, D. Sylvester, M. Orshansky and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," pp. 201-204, *CICC*, 2000.

Full Adder Evaluation and Selection for a Parallel Multiplier

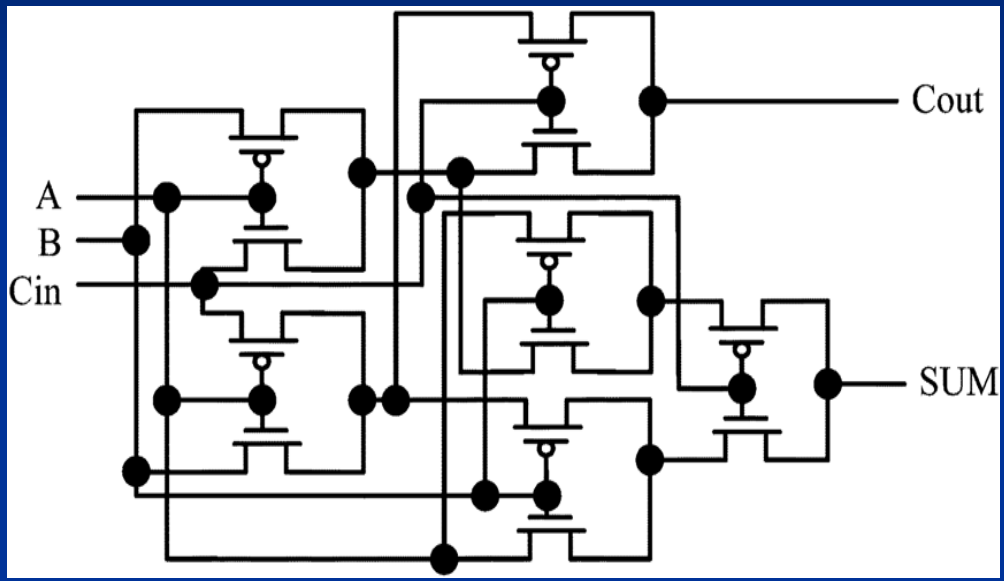
Michael Spear
Gaurav Tuteja
Earl E. Swartzlander, Jr.

Introduction

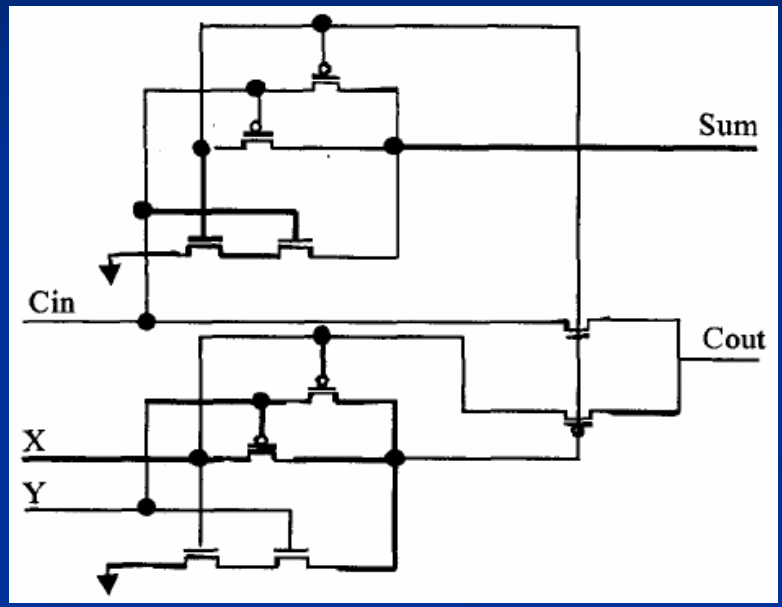
- Goal: to use different full adders along identified paths in a Dadda multiplier and achieve optimization in speed and power.

- Approach: involved 2 major steps
  1. Adder Selection and Characterization
  2. Tree Optimization

Full Adder Selection



fa12trans

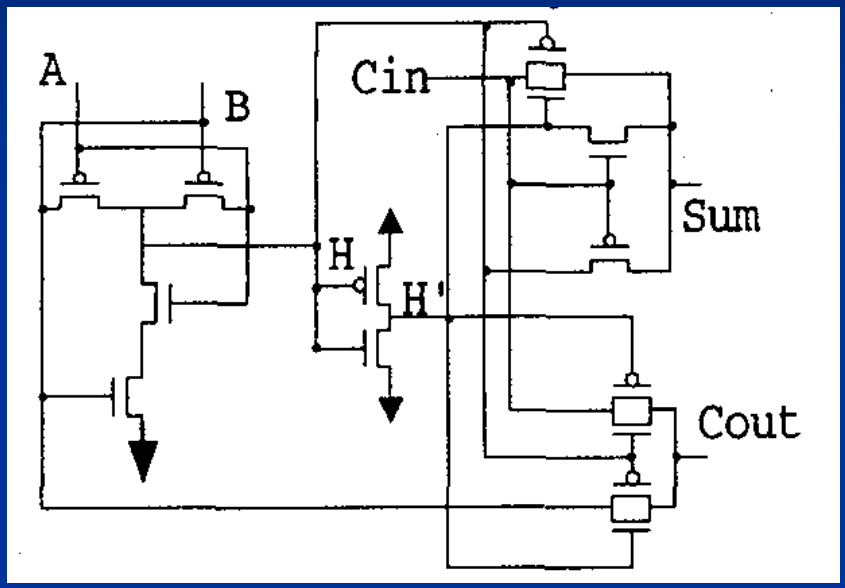


fa10trans

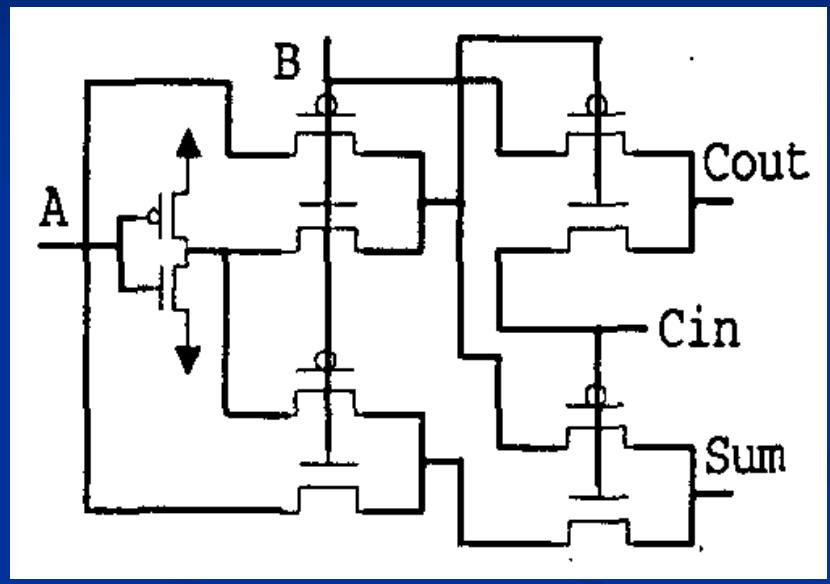
- Size: 12 transistors
- Property: uses mux's to generate outputs

- Size: 10 transistors
- Property: low power, used in embedded architectures

Full Adder Selection (contd.)



fa14trans

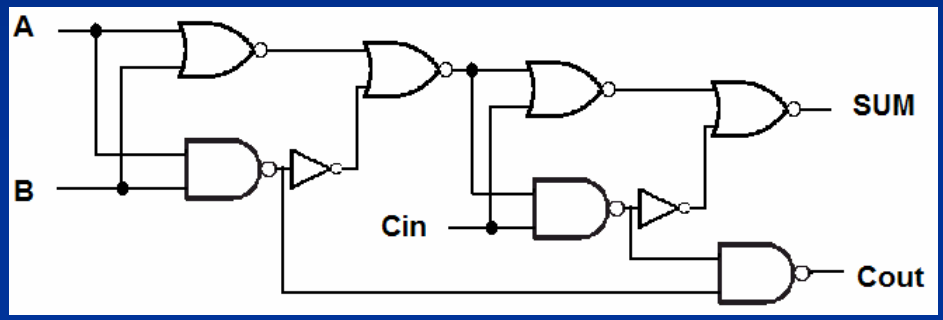


fa10trans2

- Size: 14 transistors
- Property: low power, uses xor and xnor functions for generating outputs

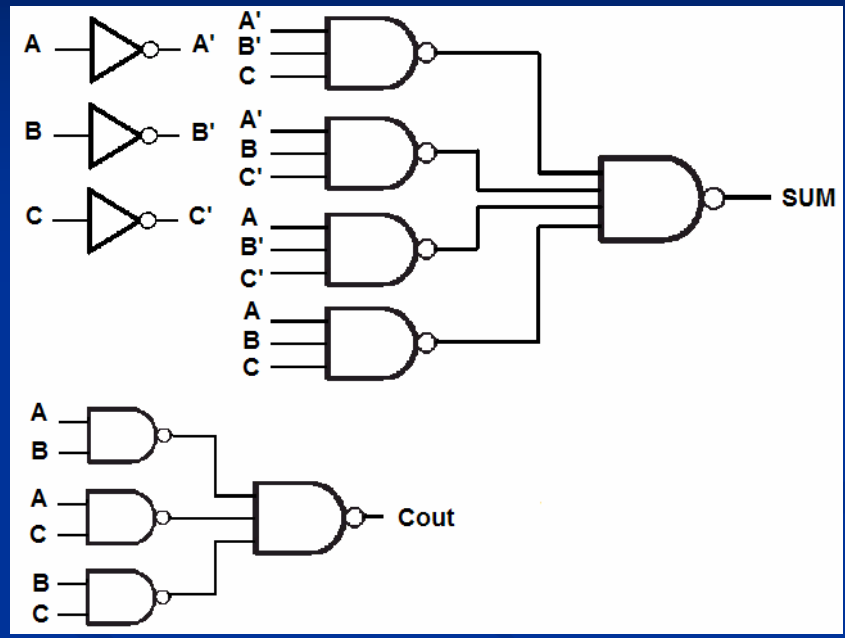
- Size: 10 transistors
- Property: relatively faster method of generating xor and xnor functions

Full Adder Selection (contd.)



fadder9

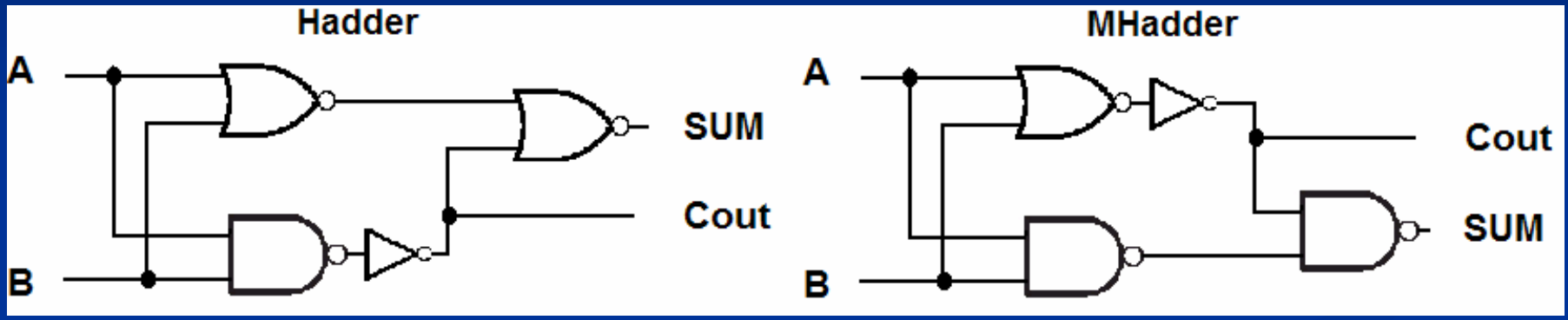
- Standard 9 gate full adder



fadder12

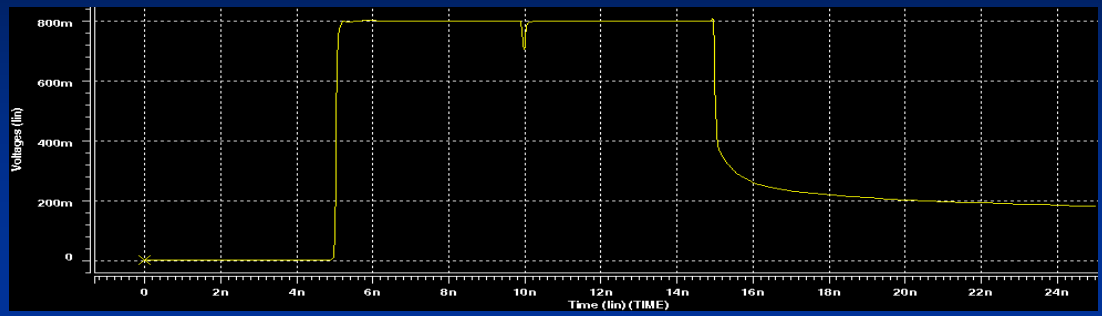
- Standard 12 gate full adder

Half Adders

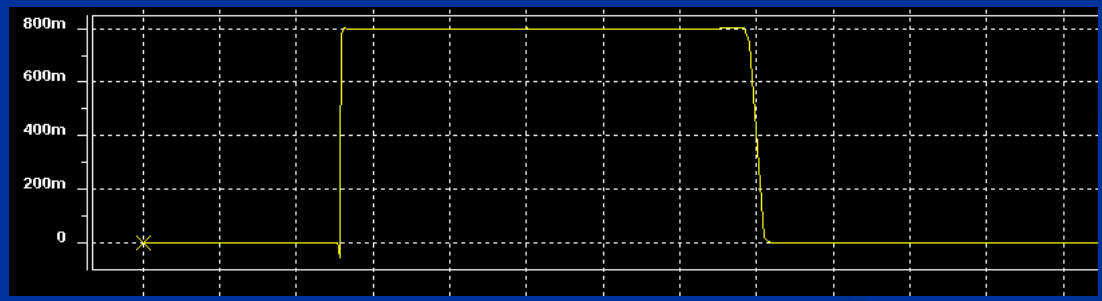


- Standard 4 gate half adder
- Standard modified half adder

Buffered Outputs



Glitch in sum output before buffering



Smoother sum output after adding buffers

- Transistor level adders needed to have buffered outputs -
  - To allow for acceptable drive strengths
  - To generate a smoother and glitch-free curve

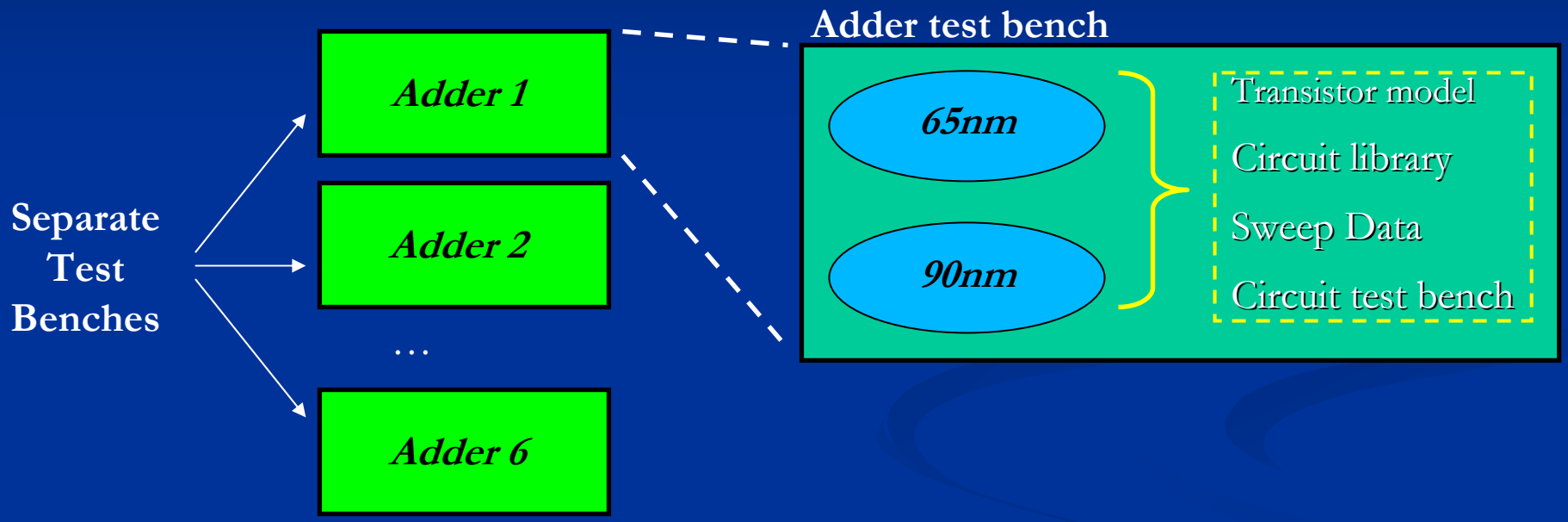
Characterization Parameters

- Tools:
  - Library of adders and testbenches built using SPICE
  - Data gathered using PERL scripts
- Technology:
  - characterized across 2 technologies - *65nm, 90nm*
  - Transistor models picked up from internet (http://www.eas.asu.edu/~ptm)
- Voltage: characterized across 3 voltages - *0.8V, 1.2V, 1.6V*

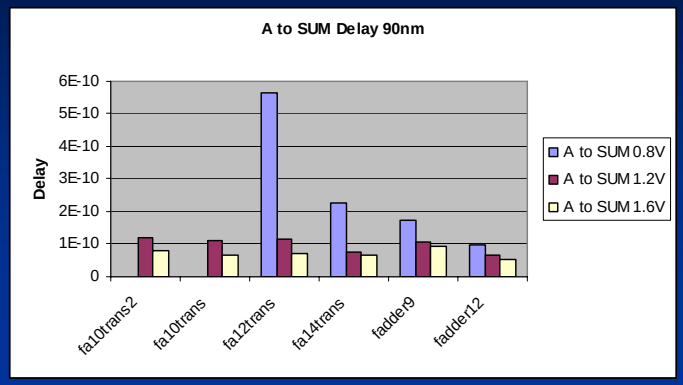
Characterization Parameters

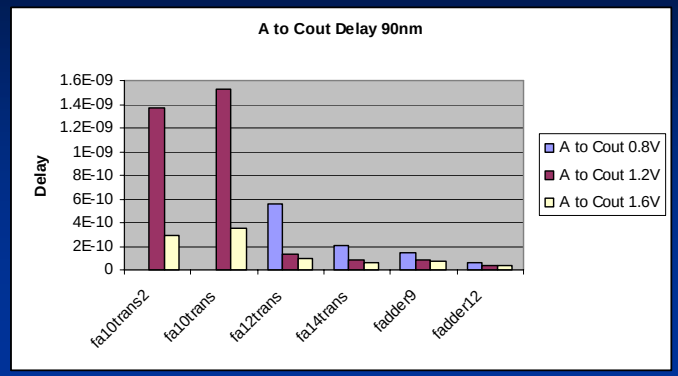
- Load: adder outputs loaded with min. sized *2/1* inverter and *10/5* inverter in separate runs
- Input capacitance: approximated using simple inverter linear model of capacitance versus delay
- Input combinations: for all 3 inputs, total of 24 combinations to run
- Power and Delay numbers were recorded from each run

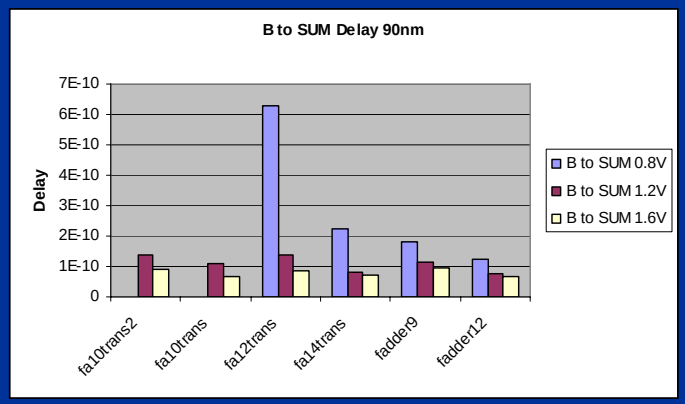
Characterization diagram

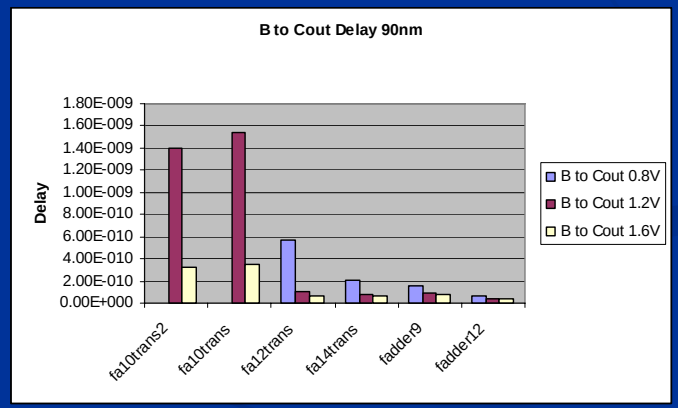


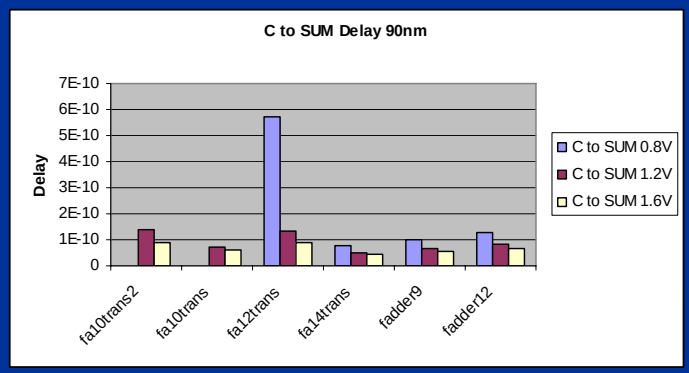
SUM delay at 90nm Cout delay at 90nm

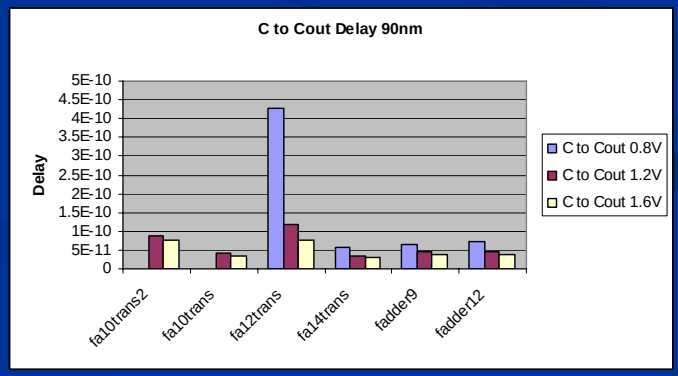








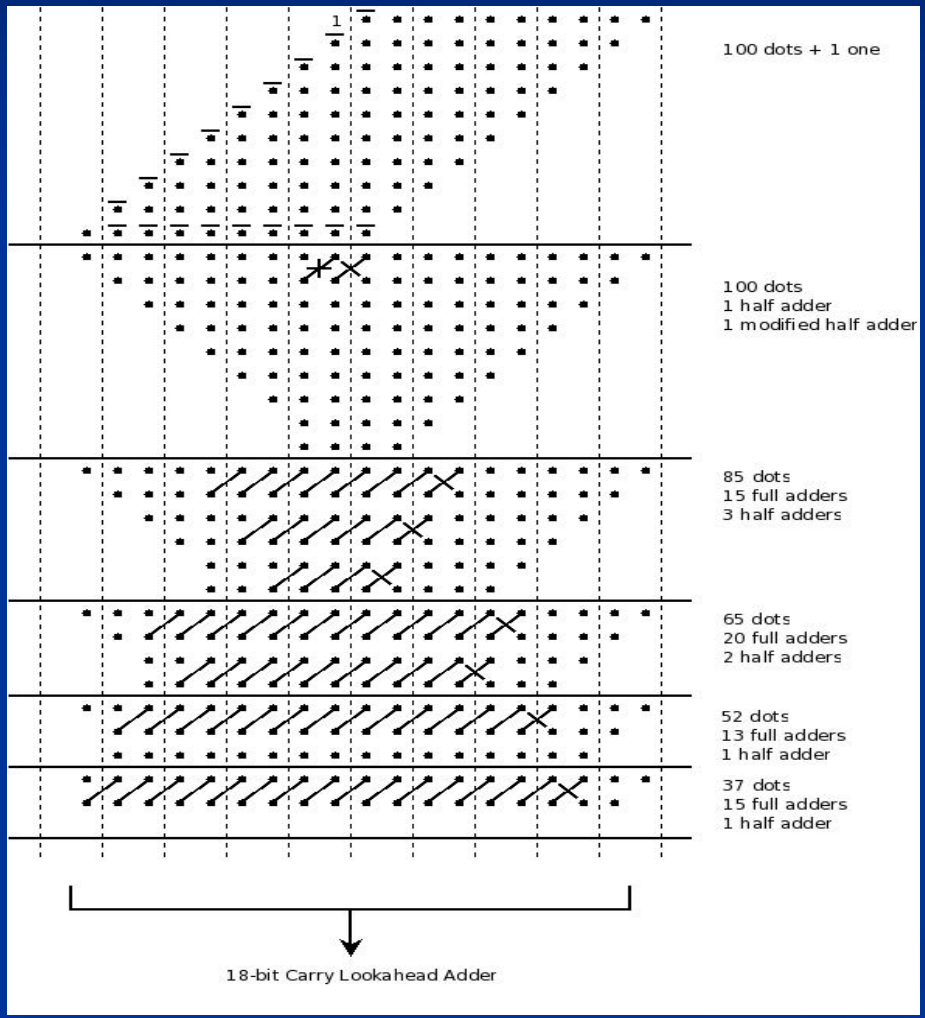


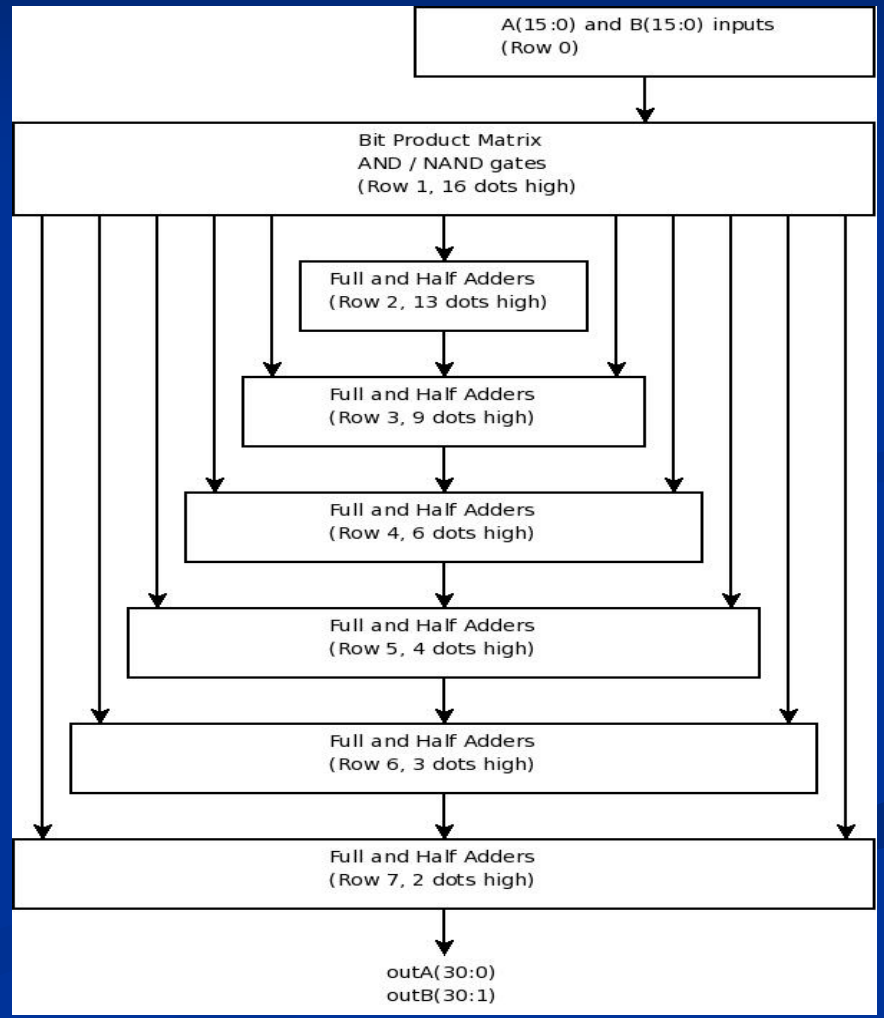


Observations

- *fa10trans, fa10trans2* did not work at 0.8V
- Transistor level adders appeared to suffer from "Vth-drop" effect
  - pfet passing a 0 or nfet passing a 1
  - affects delay at nominal and low voltages

Tree Optimization

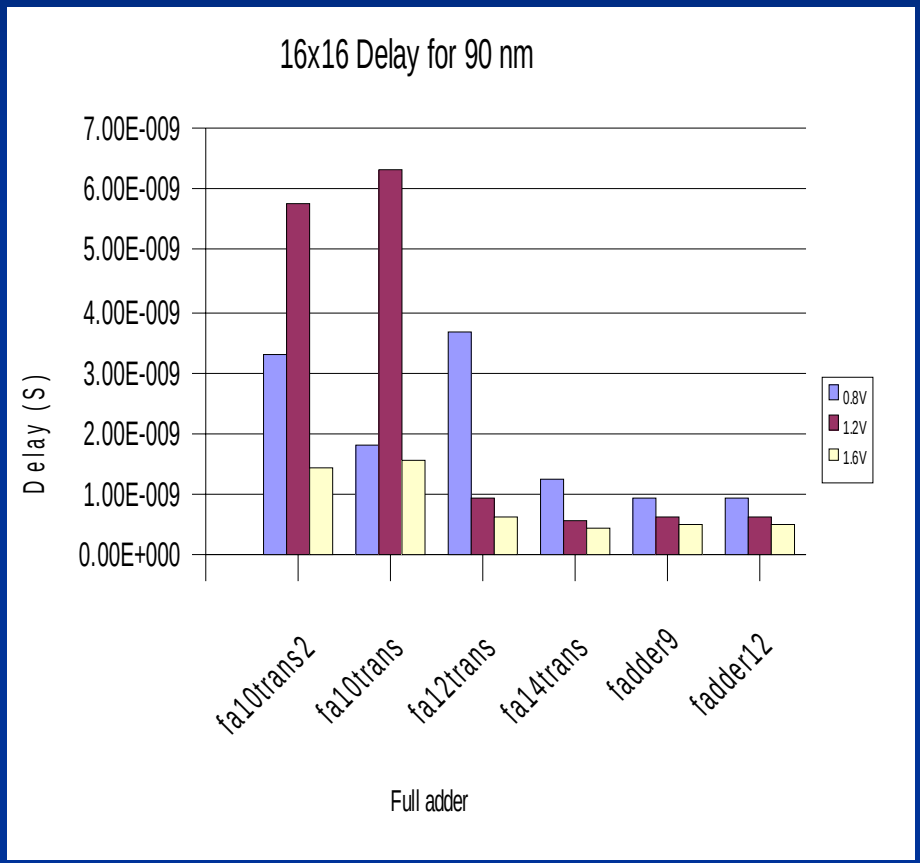


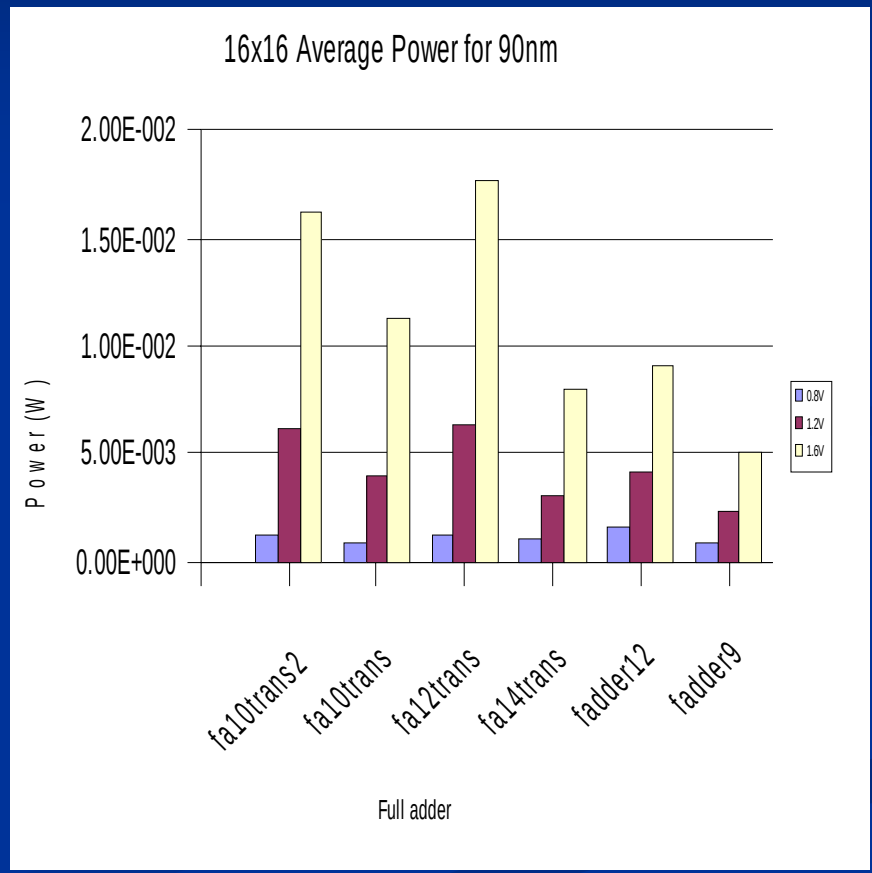


Tree Optimization Strategy

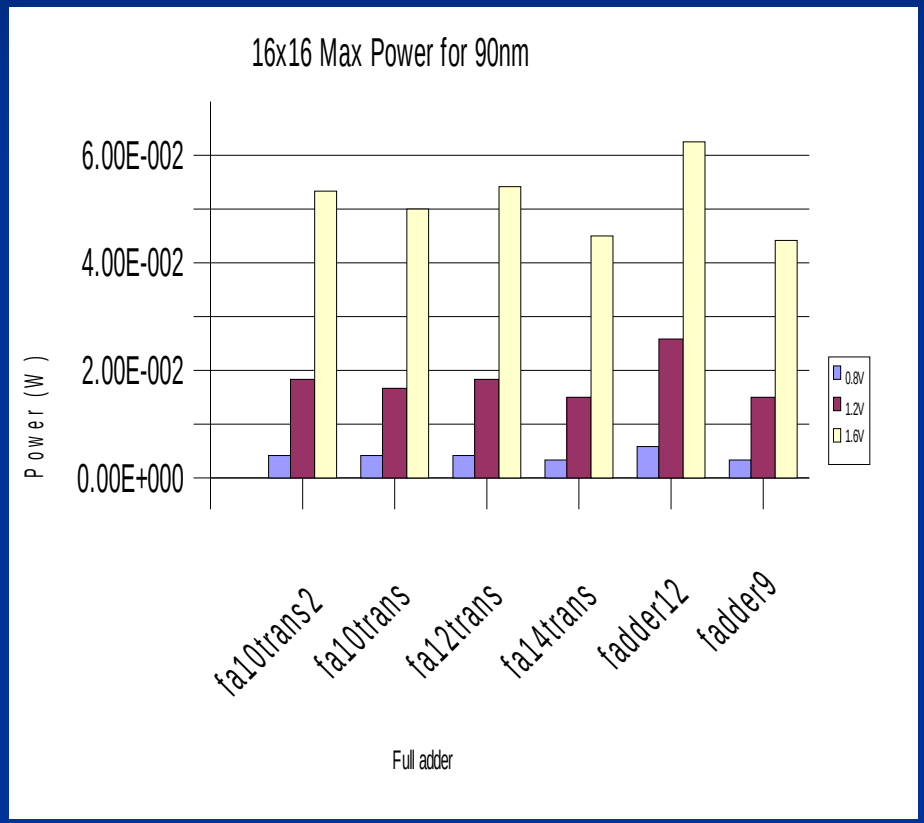
- Select the full adders with the least delay and lowest power consumption
- Construct a Dadda multiplier tree using the fast full adder
- Swap the low-power full adder into non-critical paths.

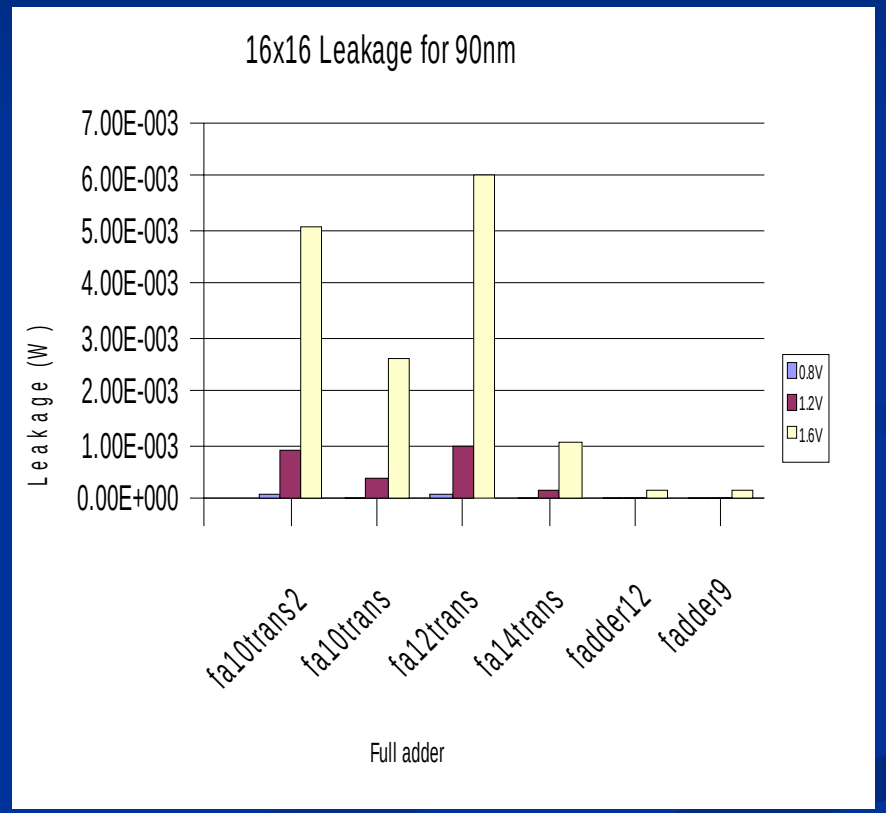
Tree Results by Full Adder





Tree Results by Full Adder



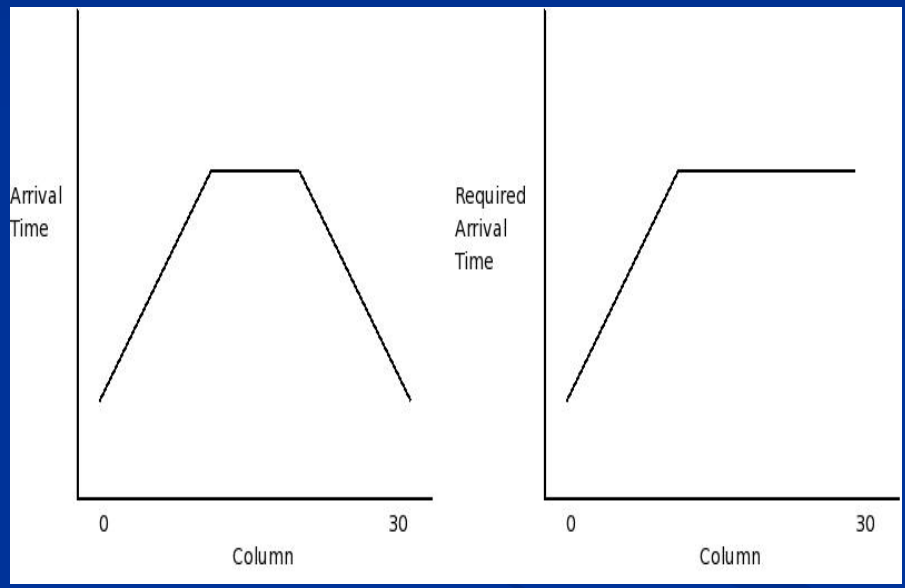


Fast and Low Power Adders

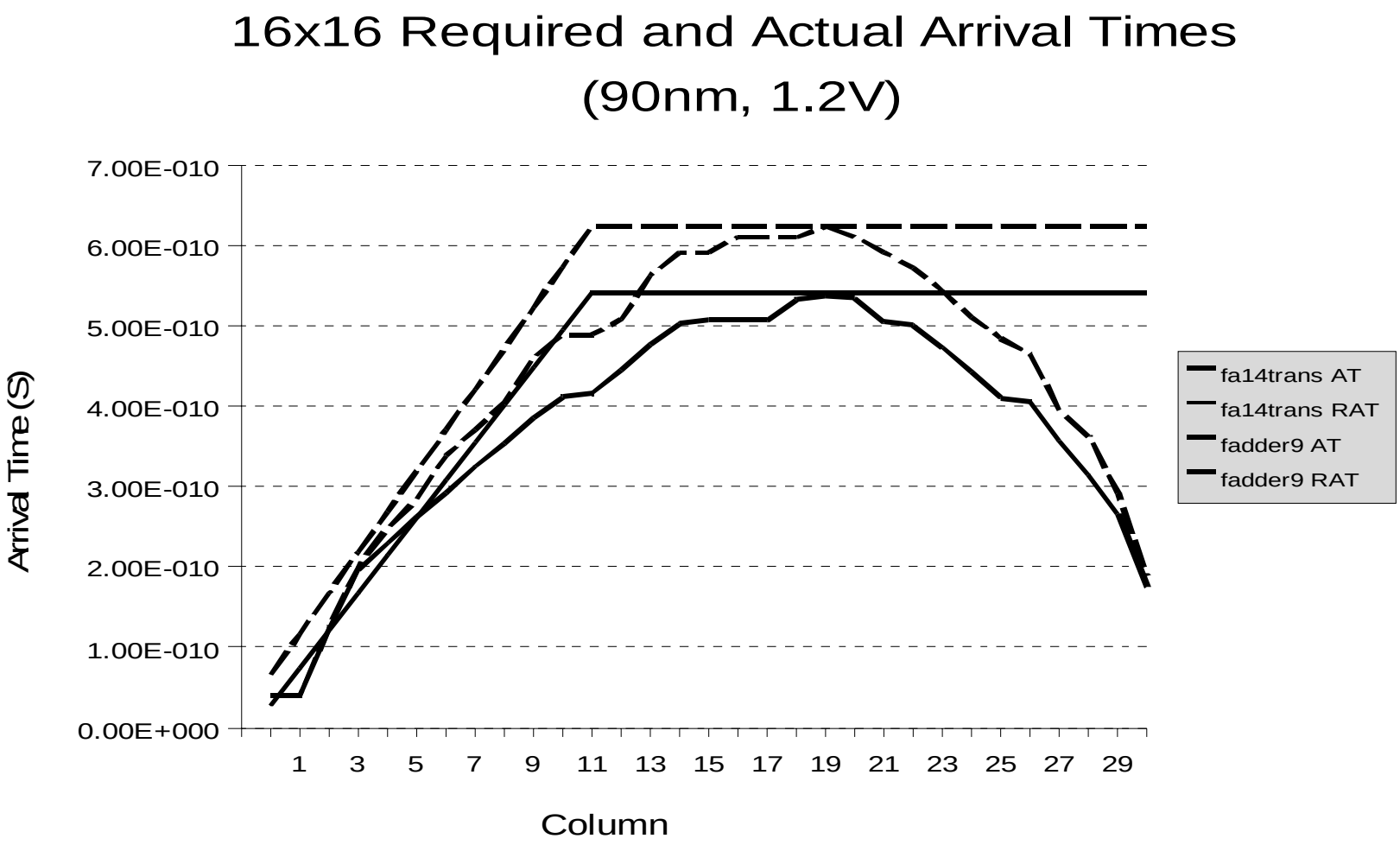
- *fatrans14* is the fastest full adder at 1.2Vand 1.6V in both 65nm and 90nm
- *fadder9* is the lowest power across all voltages at both 65nm and 90nm
- *fadder12* is relatively fast and has low leakage power but high dynamic power
- *fatrans14* and *fadder9* were chosen to construct the hybrid tree

Multiplier Critical Paths

- Plot of arrival times to column numbers forms a trapezoidal shape
- LSB's can be consumed by a ripple-carry adder
- MSB's can tolerate low-power adders

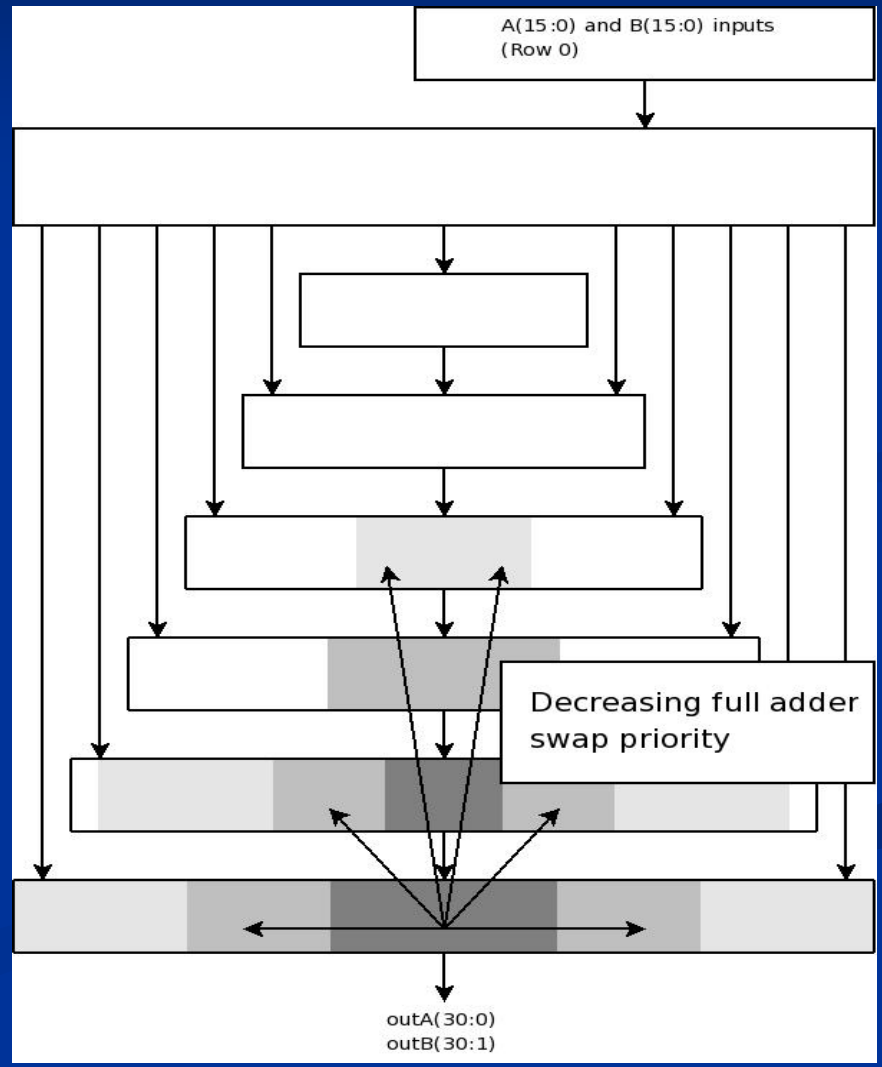


Fast and Low Power Tree Delay



Full Adder Swapping Algorithm

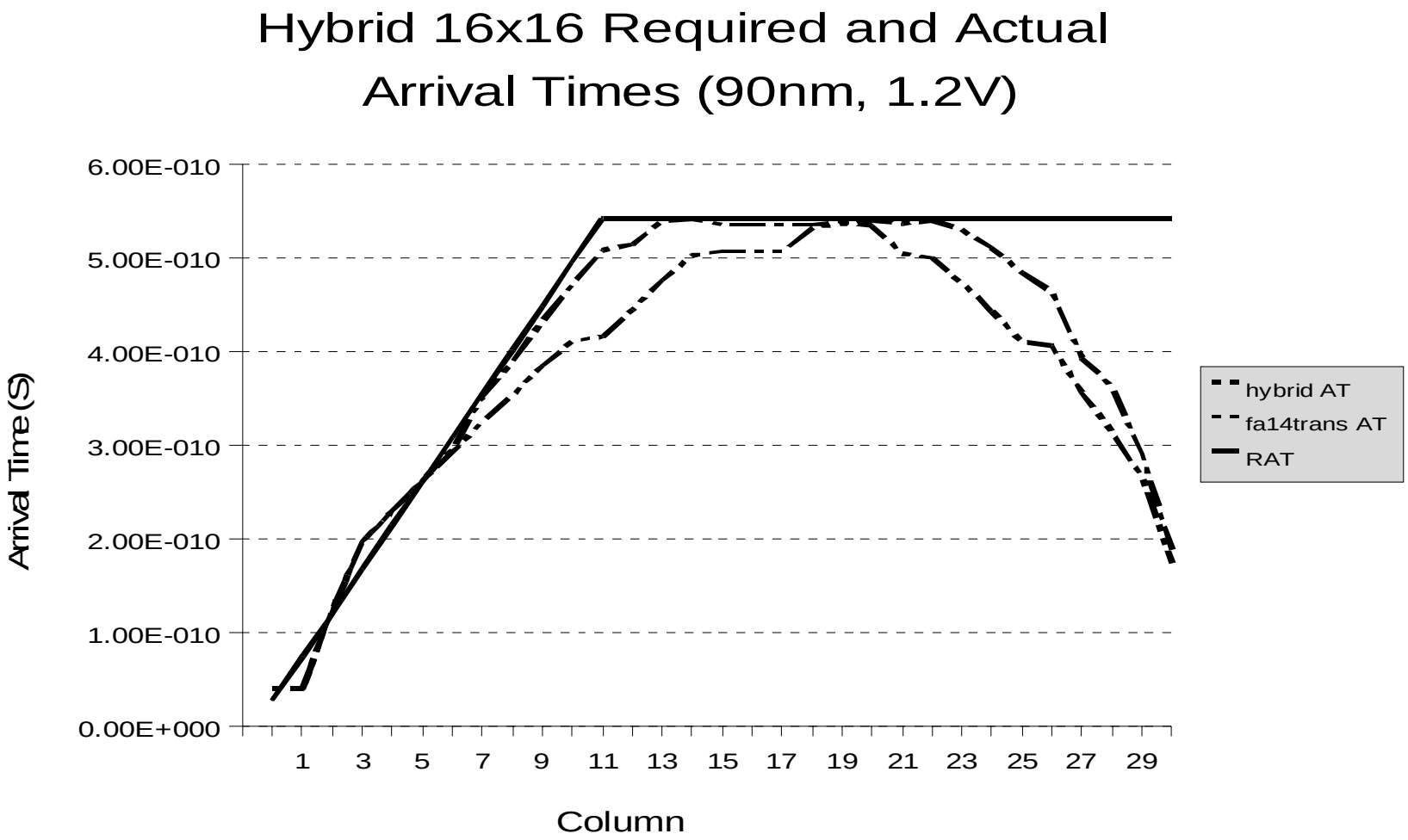
- Swap full adders where it can be done without creating non-negative timing slack
- Prioritize by distance from the lower row and middle column, where glitches are likely to be more frequent



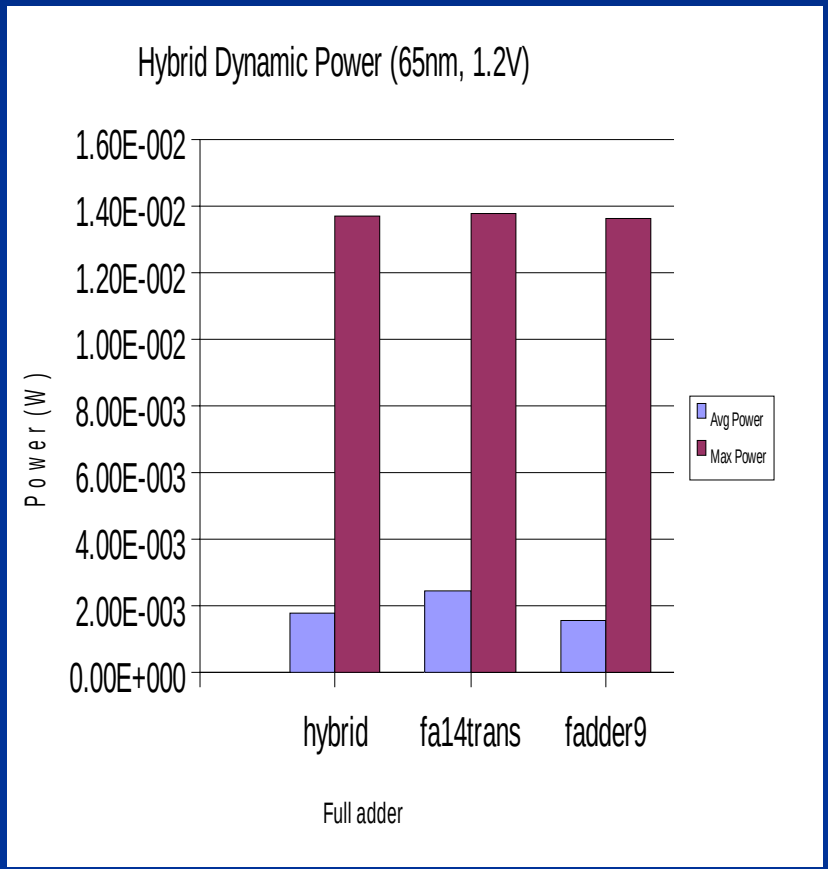
Swapping Results

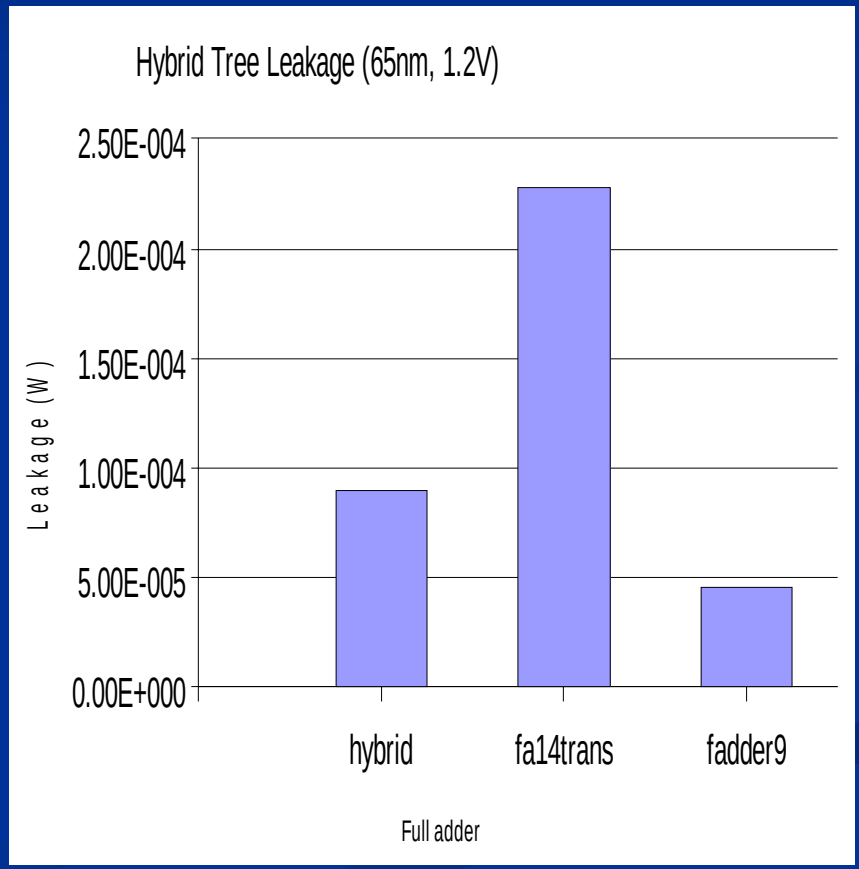
Column group	*fa14trans*	*fadder9*	*% fadder9*
0-4	3	0	0
5-9	9	16	64
10-14	11	39	78
15-19	23	39	62.9
20-24	7	33	82.5
25-30	0	15	100
Total	53	142	72.8

Fast and Hybrid Tree Delay



Hybrid Power Savings





Other Optimizations

- Pin swapping – swapped A and B pins throughout the network
  - Minor average power decrease (6.0% for 65nm at 1.6V)
  - No delay, leakage, or max power effect
- Stack tapering -
  - Increased power when gate capacitance was preserved.

Conclusions

- Transistor count is not really related to performance with respect to power and speed
  - 9 & 12 gate full adders performed better than others at low voltages
- Combining different adders in one tree will save power and at the same time have high performance

Dual Vdd Design Optimization of Fast Multipliers

Eun Jung Jang
ECE, University of Texas
at Austin
Austin, TX 78712
eunjung.jang@gmail.com

Earl E. Swartzlander, Jr.
ECE, University of Texas
at Austin
Austin, TX 78712
eswartzla@aol.com

Jebediah Keefe
ECE, University of Texas
at Austin
Austin, TX 78712
jebk@sbcglobal.net

Abstract - Reducing power consumption is important, but, it is also important to maximize the speed of the circuit especially if the circuit is heavily used, which tends to increase the power consumption. Both Dadda and Wallace multipliers have obvious critical paths. Since bits close to the MSB or the LSB come earlier than the bits in the center, paths near the MSB and the LSB are non-critical paths. In a dual-Vdd circuit, the high-Vdd gates will have lower delay, while the low-Vdd gates will consume less power. In this study, low-Vdd gates are assigned to gates on non-critical paths to save power and high-Vdd gates are assigned to gates on the critical paths to attain the same worst-case delay as that of a single high-Vdd circuit.

Keywords
Low-Power, Dual-Vdd, Multipliers, Wallace, Dadda

I. Introduction

The use of multiple supply voltages is one technique to decrease the power consumption of circuits. Consider a dual-Vdd circuit, in which gates are connected to one of two supply voltages. In comparing two gates that differ only in supply voltage, the high-Vdd gate will have lower delay, while the low-Vdd gate will consume less power. An ASIC design [1] can use multiple voltage levels to achieve an optimal balance between delay and power consumption. Despite both Dadda and Wallace multipliers have obvious critical paths, it is common to assign a single Vdd to both multipliers.

This study measured the power benefit of applying the dual-Vdd design technique at the gate level to Dadda and Wallace multipliers. This study was done at the gate level, and the level conversion and the problem of routing multiple supply voltages were ignored.

The remainder of this paper is structured as follows. Section 2 describes the approach. Simulation results are presented in Section 3, and Section 4 has the conclusions.

II. Approach

First of all, the power consumptions of single-Vdd and dual-Vdd multipliers were compared at the gate level. 8-bit by 8-bit and 16-bit by 16-bit Wallace and Dadda multipliers were implemented in Verilog HDL to see which benefits most from the dual-Vdd approach. Also, a simple discrete event simulator was implemented with Verilog HDL to estimate the power consumption by counting logic transitions. To attain the power consumption and delay of each gate of which the multipliers composed, transistor level simulation was done with the Virtuoso schematic editor and Hspice based on the TSMC 0.20μm technology. Discrete event simulation was also used in the design of the dual-Vdd implementations, since it was necessary to identify which gates were on a critical path for the purpose of voltage assignment. With the worst-delay constraints of a single high-Vdd multiplier, it was possible to assign low-Vdd to the gates on non-critical paths. After tuning the implementations, it was possible to quantify the benefit of this approach by comparing the power and delay of the dual-Vdd design with a design that uses a single high voltage supply.

In this study, high-Vdd was set to 1.4V, and low-Vdd was set to 1.0V. This is because the optimal low-Vdd in a dual-supply design is generally 40% below high-Vdd, but supply voltage cannot be scaled much below 1V due to Vth saturation in modern technology [1].

III. Result

The delay and power consumption of each gate type was estimated by transistor level simulation. Since both multipliers consist of inverters, 2-input NAND gates and 2 input NOR gates, simulations were done for an inverter, a 2-input NAND gate and a 2-input NOR gate. Table 1 presents the delay and power consumption of an inverter.

From tables 2 and 3, if low-Vdd is assigned to gates, the delay will increase by around 60%, but there is almost a 60% decrease in power consumption.

Rising and falling propagation delays attained from Hspice simulation were assigned to each gate of the Verilog code so that more accurate delays could be estimated.

TABLE I
Inverter Characteristic

	Delay		Power Consumption	
input	0 -> 1	1->0	0->1	1->0
Vdd=1.0V	0.077 ns	0.093 ns	9.852 μW	9.923 μW
Vdd=1.4V	0.128 ns	0.147 ns	4.508 μW	4.379 μW

(PMOS : 800/200, NMOS: 300/200, Room temperature)

TABLE II
Delay increase with low-Vdd relative to high-Vdd

	INV	NAND	NOR
tpdr	58.1%	57.3%	55.5%
tpdf	65.7%	63.2%	46.8%

TABLE III
Decrease in power consumption with low-Vdd relative to High-Vdd

INV	NAND	NOR
-55.1%	-57.0%	-56.0%

Implementing multipliers is straightforward. For the last stage, a carry lookahead adder was used. Initially, high-Vdd was assigned to all gates of the Wallace and Dadda multipliers. The multiplication was done from 1x1 to 255x255 for the 8-bit by 8-bit multipliers, and 2000 sets of random input were generated for the 16-bit by 16-bit multipliers. Multipliers were reset to zero after each multiplication. The discrete event simulator counts the number of transitions of each gate. The discrete event simulator counts the number of transitions for the intermediate results as well as for the correct results, which gives a more accurate number of transitions. Through simulation, it was possible to check the functional correctness of both multipliers. Also, the worst-case delays of high-Vdd Wallace and Dadda multipliers were measured. The worst-case delay of an 8-bit by 8-bit Wallace multiplier is 5194ps, and that of an 8-bit by 8-bit Dadda multiplier is 5364ps. The worst-case delay of a 16-bit by 16-bit Wallace multiplier is 6230ps, and that of a 16-bit by 16-bit Dadda multiplier is 6485ps. The Wallace multiplier is around 3% to 4% faster than the Dadda multiplier. Tables 4 to 7 present the number of transitions of each gate for the entire input sets and the average power consumption of each gate for a single multiplication. The total average power consumption of an 8-bit by 8-bit Wallace multiplier for a single multiplication is 12.08 mW, and that of an 8-bit by 8-bit Dadda multiplier is 12.96 mW. The total average power consumption of a 16-bit by 16-bit Wallace multiplier for a single multiplication is 54.38 mW, and that of a 16-bit by 16-bit Dadda multiplier is 52.64 mW.

TABLE IV
Power consumption of an 8-bit by 8-bit high-Vdd Wallace multiplier

	# of transistions	Pwr consumption
High-Vdd INV	6635654	1008.9 µW
High-Vdd NAND	6639271	2026.1 µW
High-Vdd NOR	25049331	9043.9 µW

TABLE V
Power consumption of an 8-bit by 8-bit high-Vdd Dadda multiplier

	# of transistions	Pwr consumption
High-Vdd INV	8049727	1223.9 µW
High-Vdd NAND	7471373	2279.8 µW
High-Vdd NOR	26188017	9455.1 µW

The discrete event simulator also specifies the worst-case delays of each path. Fig. 1 presents the row reduction process of an 8-bit by 8-bit Wallace multiplier. The last two rows, row 4_0 and row 4_1, are inputs to the carry propagation adder. In this case, the bits in center of the row come later than bits near to MSB and LSB. Fig. 2 shows the result of discrete event simulation. From Fig. 2, the center bits have longer worst-case delays when single high-Vdd is assigned to all gates of a Wallace multiplier.

TABLE VI
Power consumption of a 16-bit by 16-bit high-Vdd Wallace multiplier

	# of transistions	Pwr consumption
High-Vdd INV	1052810	5204.6 µW
High-Vdd NAND	1248457	12389.7 µW
High-Vdd NOR	3134415	36793.3 µW

TABLE VII
Power consumption of a 16-bit by 16-bit high-Vdd Dadda multiplier

	# of transistions	Pwr consumption
High-Vdd INV	1143959	5655.2 µW
High-Vdd NAND	1330158	13197.2 µW
High-Vdd NOR	2878574	33790.1 µW

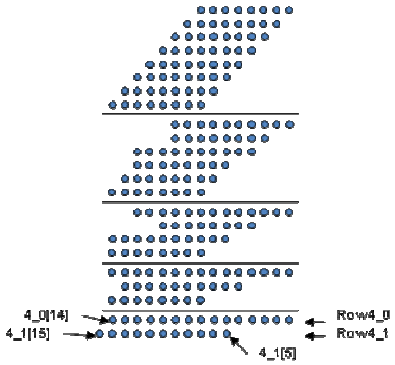


Fig. 1. Row reduction process of an 8-bit by 8-bit Wallace multiplier

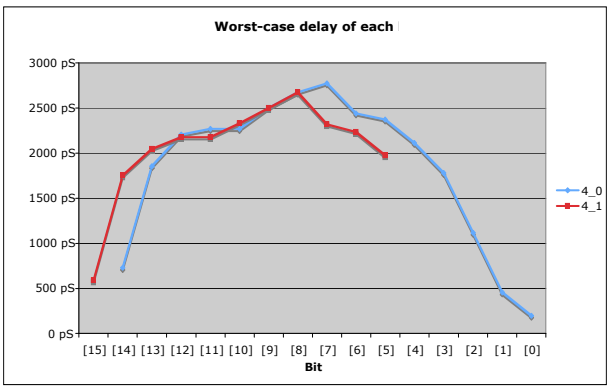


Fig. 2. Worst-case delays of CPA inputs of an 8-bit by 8-bit high-Vdd Wallace multiplier

Fig. 3 represents the row reduction process of a Dadda multiplier. Fig. 4 shows the result of worst-case delays of each bit.

The gate-level simulations confirmed that both Dadda and Wallace multipliers have shorter worst-case delays around

the MSB and the LSB. Also, the transistor level simulation results showed obvious trade-offs between the power-consumption and the delay of gates. Based on these results, if low-Vdd is assigned to the gates on non-critical paths, the power consumption will be increased while the worst-case delay of the entire multiplier remains the same. Low-Vdd assignment was done based on the result of worst-case delay of each bit. Also, the worst-case delay of the entire multiplier was estimated each iteration as well to meet the worst-case delay constraints.

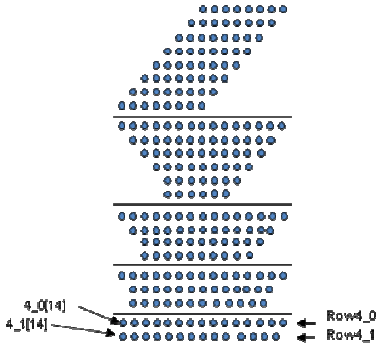


Fig. 3. Row reduction process of an 8-bit by 8-bit Dadda multiplier

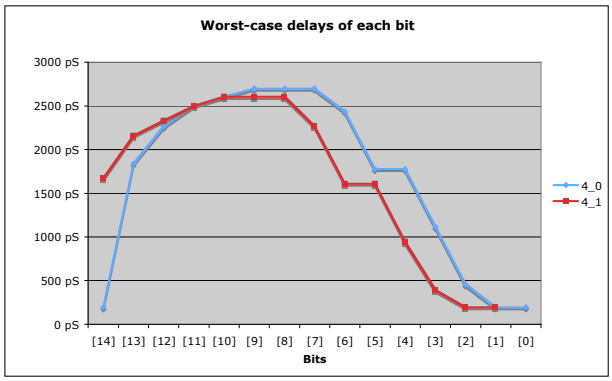


Fig. 4. Worst-case delays of CPA inputs of an 8-bit by 8-bit high-Vdd Dadda multiplier

Fig. 5 shows the dual-Vdd assignment of an optimized Wallace multiplier. For the best delay-performance, high-Vdd was assigned to the every gate of the carry lookahead adder. Low-Vdd was assigned to as many gates as possible, but high-Vdd was assigned to the gates on the critical paths to meet the worst-case delay constraints. In Fig. 5, black dots represent the gates the supply voltage of which is low-Vdd. Grey dots represent wires that don't need any supply voltage.

After assigning dual-Vdd, the power consumption and worst-case delays of each gate were estimated. Dual-Vdd assignment can be implemented easily by replacing gates. Thus, 6 different types of basic gates were used to build dual-Vdd multipliers – high-Vdd inverters, 2-input NAND and 2-input NOR gates, low-Vdd inverters, 2-input NAND, and 2-input NOR gates. High-Vdd gates and low-Vdd gates have the same structure. There are only two differences between high-Vdd gates and low-Vdd gates; they have different rising and falling propagation delay and they increase separate counters when there is a transition.

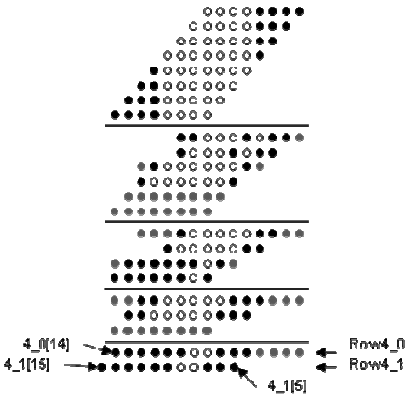


Fig. 5. Dual-Vdd assignment of an 8-bit by 8-bit Wallace multiplier

Table 8 presents the number of transitions of each type of gate of Wallace multiplier. It also shows power consumption as well. Table 8 shows the decrease in number of transitions of high-Vdd gates. However, the total number of gate transitions remains about the same because there is also a certain amount of number of transitions of low-Vdd gates. The power consumption represents the average power consumed for a single multiplication. Total average power consumption of an 8-bit by 8-bit dual-Vdd Wallace multiplier for a single multiplication is 9.85 mW. An 8-bit by 8-bit dual-Vdd Wallace multiplier consumes less power than an 8-bit by 8-bit high-Vdd Wallace multiplier by 2.23 mW (an 18% reduction) while keeping the worst-case delay the same at 5194 ps.

TABLE VIII
Power consumption of an 8-bit by 8-bit dual-Vdd Wallace multiplier

		# of transitions	Pwr. Cons.
High-Vdd	INV	5416121	823.5 µW
	NAND	4976734	1518.7 µW
	NOR	16866078	6089.4 µW
Low-Vdd	INV	1290111	88.2 µW
	NAND	1716425	224.5 µW
	NOR	7991079	1105.9 µW

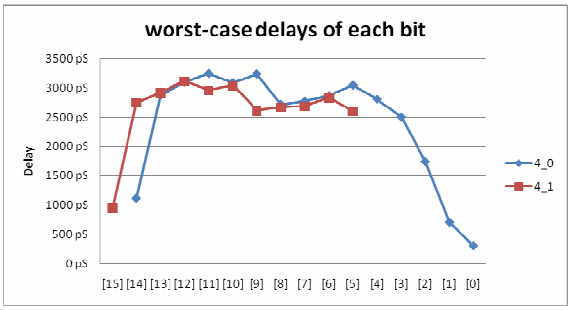


Fig. 6. Worst-case delays of CPA inputs of an 8-bit by 8-bit dual-Vdd Wallace multiplier

Fig 6 shows the delays of an 8-bit by 8-bit dual-Vdd Wallace multiplier. The difference between Fig 2 and Fig 6

are obvious.

Fig 7 shows Dual-Vdd assignment of a Dadda multiplier. Low-Vdd was assigned to as many gates as possible keeping the worst-case delay the same as that of a single 8-bit by 8-bit high-Vdd Dadda multiplier.

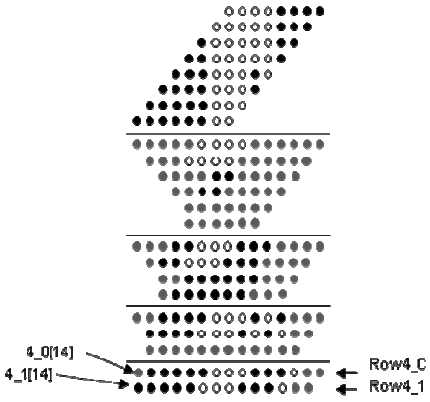


Fig 7. Dual-Vdd assignment of a Dadda multiplier

Table 9 presents the number of transitions and power consumption of each type of gate of an 8-bit by 8-bit Dadda multiplier. Total average power consumption of an 8-bit by 8-bit dual-Vdd Dadda multiplier for a single multiplication is 10.27 mW. Since the power consumption of an 8-bit by 8-bit high-Vdd Dadda multiplier was 12.96 mW, an 8-bit by 8-bit dual-Vdd Dadda multiplier consumes 2.69 mW less than an 8-bit by 8-bit high-Vdd Dadda multiplier keeping the worst-case delay the same at 5364 ps. Thus, a 21% power consumption savings is obtained for the 8-bit by 8-bit dual-Vdd Dadda multiplier. Fig 8 presents the worst-case delays of each bit of an 8-bit by 8-bit dual-Vdd Dadda multiplier.

TABLE IX
Power consumption of an 8-bit by 8-bit dual-Vdd Dadda multiplier

		# of transitions	Pwr. Cons.
High-Vdd	INV	6210065	944.2 µW
	NAND	4993141	1523.7 µW
	NOR	16971218	6127.4 µW
Low-Vdd	INV	1877264	128.3 µW
	NAND	2555742	334.3 µW
	NOR	8770921	1213.8 µW

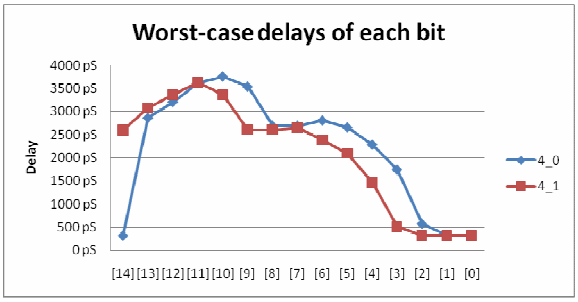


Fig 8. Worst-case delays of CPA inputs of an 8-bit by 8-bit dual-Vdd Dadda multiplier

Tables 10 and 11 present the number of transitions and power consumption of each type of gate of a 16-bit by 16-bit Wallace and Dadda multiplier, respectively. Total average power consumption of a 16-bit by 16-bit dual-Vdd Wallace multiplier for a single multiplication is 43.01 mW. Therefore, a dual-Vdd Wallace multiplier consumes 11.37 mW (21%) less than a high-Vdd Wallace multiplier while keeping the worst-case delay the same at 6230 ps. Total power consumption of a 16-bit by 16-bit dual-Vdd Dadda multiplier for a single multiplication is 39.59 mW. 13.06 mW (25%) of power is saved while keeping the worst-case delay the same at 6485 ps.

TABLE X
Power consumption of a 16-bit by 16-bit dual-Vdd Wallace multiplier

		# of transitions	Pwr. Cons.
High-Vdd	INV	748454	3700.0 µW
	NAND	867859	8610.5 µW
	NOR	2146348	25195.0 µW
Low-Vdd	INV	262684	583.7 µW
	NAND	343955	1462.8 µW
	NOR	768719	3458.9 µW

TABLE XI
Power consumption of a 16-bit by 16-bit dual-Vdd Dadda multiplier

		# of transitions	Pwr. Cons.
High-Vdd	INV	767553	3794.4 µW
	NAND	778949	7728.3 µW
	NOR	1785290	20956.6 µW
Low-Vdd	INV	341211	758.2 µW
	NAND	498241	2119.0 µW
	NOR	939786	4228.6 µW

IV. Conclusions

Dual-Vdd multipliers consume less power than single high-Vdd multipliers, while the worst-case delay is the same as those of high-Vdd multipliers. In cases of an 8-bit by 8-bit and a 16-bit by 16-bit dual-Vdd Wallace multiplier, the power consumption was reduced by 18 % and 21%, repectively. An 8-bit by 8-bit and a 16-bit by 16-bit dual-Vdd Dadda multiplier consumed 21% and 25% less power, respectively. In summary, multipliers can save around 18% to 25% of the total power consumption when dual-Vdd assignment is possible. A dual-Vdd Dadda multiplier benefits a little more from dual-Vdd assignment than a dual-Vdd Wallace multiplier. However, the difference is very small. Since this result was attained from relatively small multipliers, this study can be extended by using multipliers with wider inputs.

References

[1] R. Puri, L. Stok, J. Cohn, D. Kung, D. Pan, D. Sylvester, A. Srivastava and S. Kulkarni. “Pushing ASIC Performance in a Power Envelope,” DAC 2003, pp. 788-793.

Dual Vdd Design Optimization of Fast Multipliers

The University of Texas at Austin

Eun Jung Jang
Earl Swartzlander
Jebediah Keefe

Problem

- Reducing power consumption is important in modern circuit design
- Reducing power consumption tends to increase the delay of a circuit
- Performance is also an important factor
- ***How to save the power consumption of Dadda and Wallace multipliers without a performance penalty***?

Outline

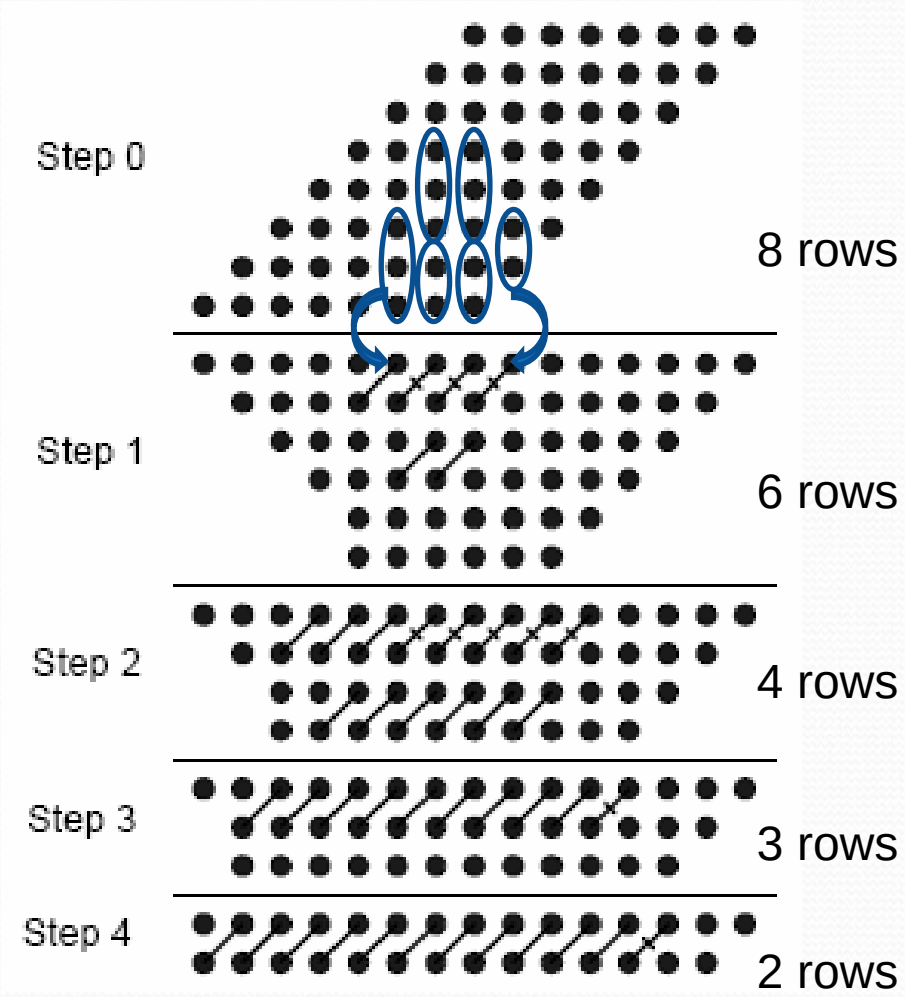
- Problem
- Row reduction process of fast multipliers
- Approach
- Experimental methodology
- Dual –Vdd assignment
- Power consumption
- Conclusions

Row reduction process

- 8x8 Dadda multiplier
- Length of rows

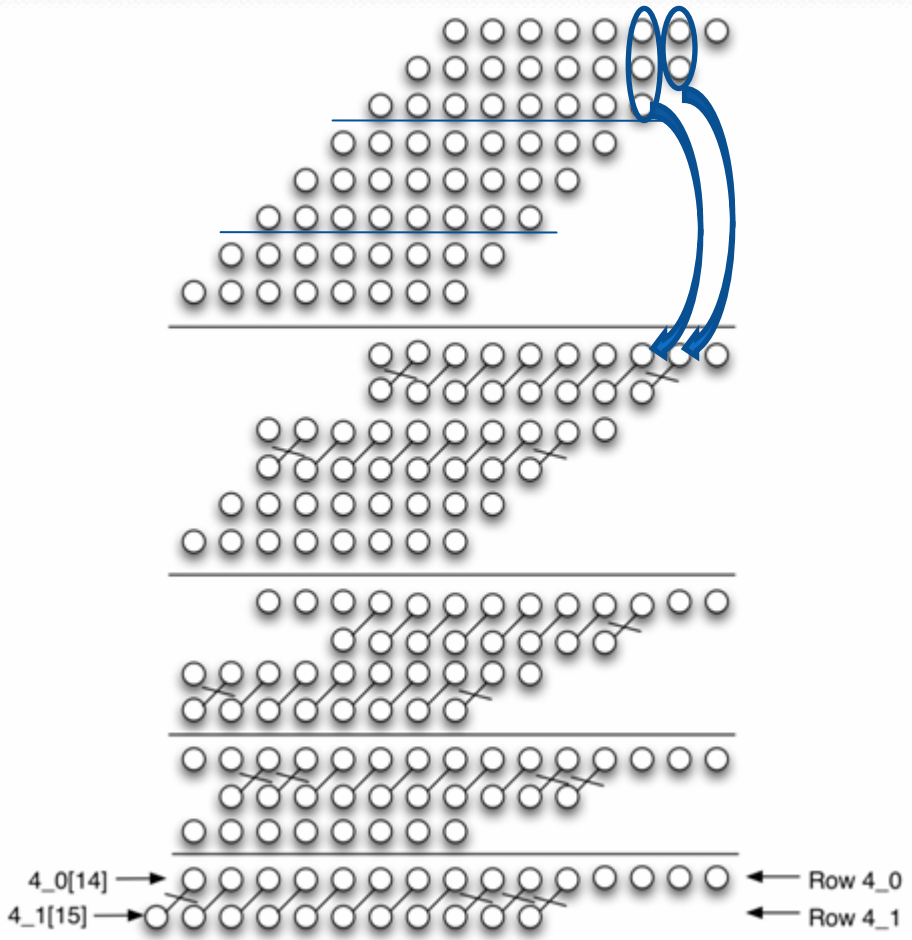
$$x_{n+1} = \left\lceil \frac{2}{3} x_n \right\rceil$$

  - x_n : number of rows in step n



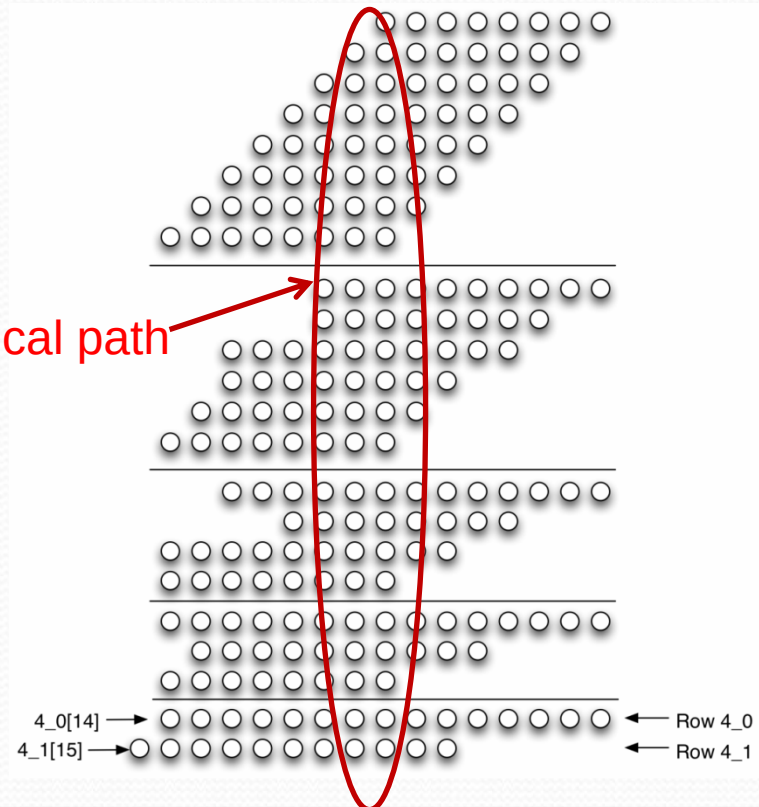
Row reduction process – cont.

- 8x8 Wallace multiplier



Approach

- Both multipliers have obvious critical paths
- Assign lower Vdd to gates on non-critical paths
- Achieve optimal balance between delay and power consumption



Experimental methodology

- Hspice + discrete event simulation
  - Hspice simulation for the power consumption and the delay of each type of gate, TSMC 0.20μm technology
  - Discrete event simulator counts the logic transitions including glitch transitions
- Dual Vdd assignment was done at gate level
- Power consumption of single-Vdd and dual-Vdd Wallace and Dadda multipliers were compared at gate level

Experimental methodology –cont.

- High Vdd : 1.4V , Low Vdd : 1.0V

	INV	NAND	NOR
t_{pdr}	58.1 %	57.3 %	55.5%
t_{pdf}	65.7 %	63.2 %	46.8 %

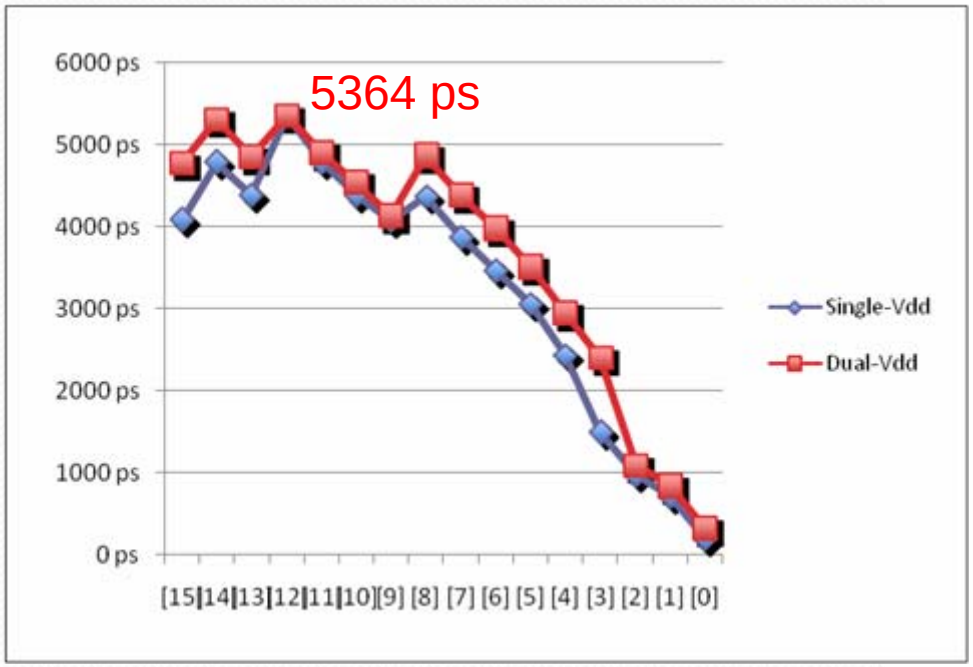
Delay increase with low-Vdd relative to high-Vdd

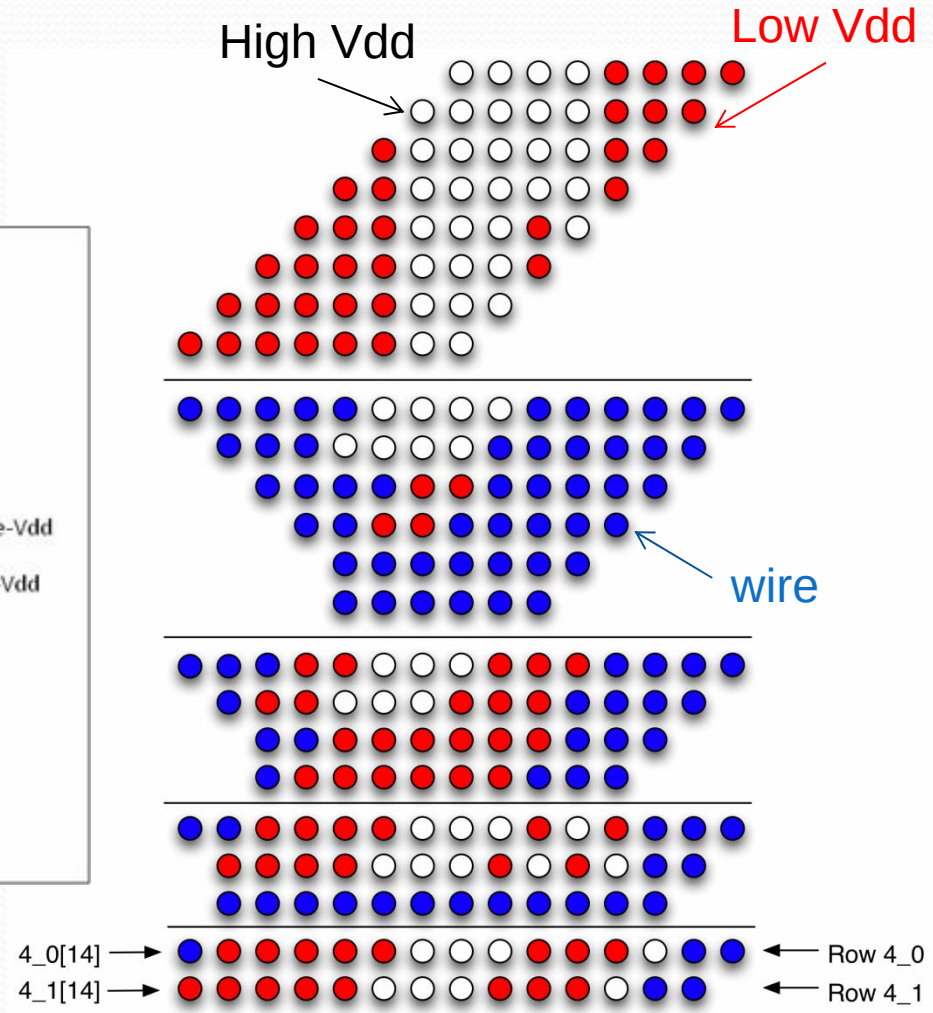
INV	NAND	NOR
- 55.1 %	- 57.0 %	- 56.0 %

Decrease in power consumption with low-Vdd relative to high-Vdd

Dual-Vdd assignment

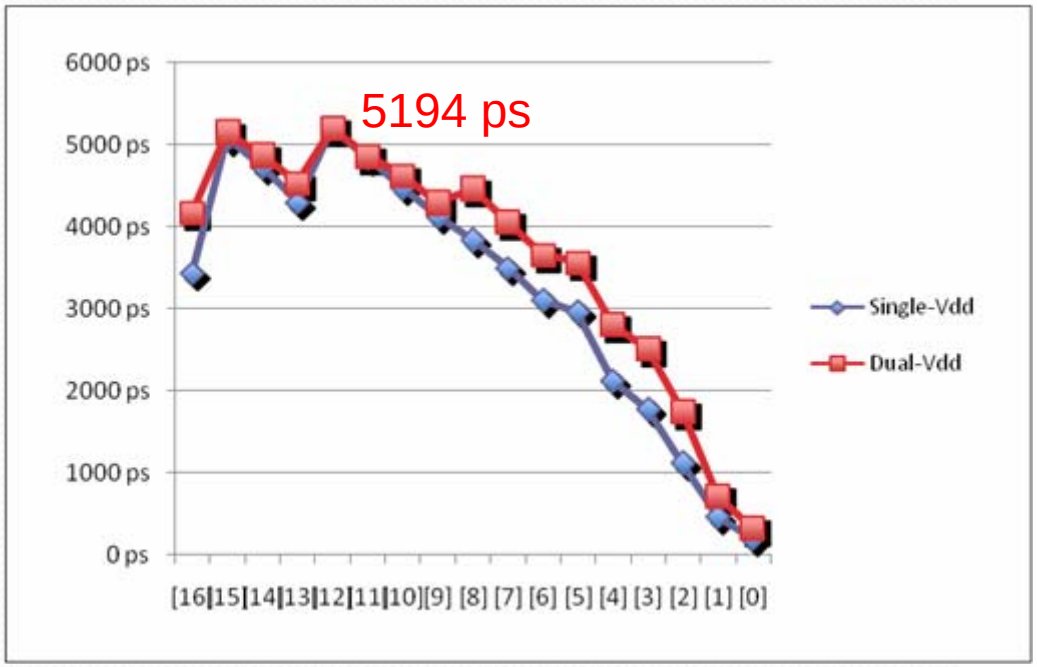
- 8x8 Dadda multiplier

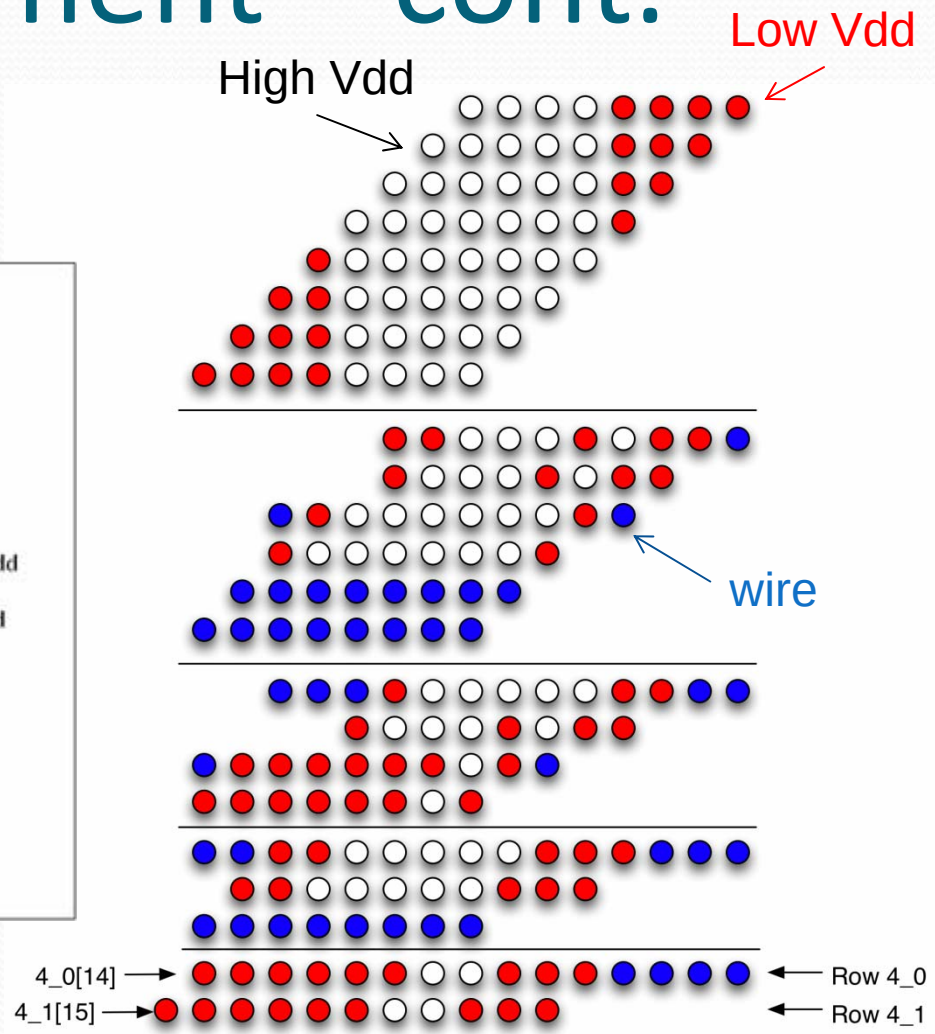




Dual-Vdd assignment – cont.

- 8x8 Wallace multiplier





Power consumption

Number of logic transitions of 8bit by 8bit multipliers

Single-Vdd	Dadda	Wallace
High-Vdd INV	8049727	6635654
High-Vdd NAND	7471373	6639271
High-Vdd NOR	26188017	25049331

Dual-Vdd	Dadda	Wallace
High-Vdd INV	6210065	5416121
High-Vdd NAND	4993141	4976734
High-Vdd NOR	16971218	16866078
Low-Vdd INV	1877264	1290111
Low-Vdd NAND	2555742	1716425
Low-Vdd NOR	8770921	7991079

Power consumption-cont.

	8x8		16x16	
	Dadda	Wallace	Dadda	Wallace
Single-Vdd	12.96 mW	12.08 mW	52.64 mW	54.38 mW
Dual-Vdd	10.27 mW	9.85 mW	39.59 mW	43.01 mW
Difference	-2.69 mW **(-21%)**	-2.23 mW **(-18%)**	-13.06 mW **(-25%)**	-11.37 mW **(-21%)**

Conclusions

- Multipliers consume around ***18%~25% less power*** when dual-Vdd assignment is possible
- The worst-case ***delay remains the same***

A Hybrid Approach to Last Stage Addition for Wallace and Dadda Multipliers

James Haydn Nelson Eiman Ebrahimi Mahnaz Sadoughi Earl E. Swartzlander, Jr.

Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78705
512-731-7251
nelsonjh@alumni.utexas.net e_ebrahimi@mail.utexas.edu mahnaz@mail.utexas.edu Eswartla@aol.com

Abstract - A hybrid final adder stage to Wallace and Dadda multipliers is investigated. By studying the non-uniform arrival times in each bit position for both Wallace and Dadda reduction methods, the critical path can be optimized by exploiting the timing characteristics of Carry Lookahead, Carry Skip and Carry Select adders. Using such a hybrid adder improves the critical path of a 32 and 16 bit Wallace multiplier by 10.8% and 7.8% respectively and by 8.4% on a 16 bit Dadda multiplier.

I. Introduction

A fast method for binary multiplication was suggested by Wallace [1] and refined by Dadda [2], [3]. These approaches compute the product in three steps: 1) a bit product matrix is formed; 2) the bit product matrix is reduced to two rows; 3) the two rows are added to compute the product result. Optimizing both the bit product reduction and the final addition stage is critical to producing a minimal delay.

The difference between the Wallace and Dadda methods is how each reduction stage is created. Wallace proposed the use of (3,2) and (2,2) counters forming groups of three rows in the bit product matrix where each iteration uses half adders and full adders to reduce the bit product by as much as possible. The Dadda algorithm takes a different approach stating that given a matrix depth one can only reduce as many rows as available by the type of counters used. Using this theory, Dadda's algorithm strives to save hardware by not reducing more than needed to achieve the next possible number of levels. This reduction approach uses less hardware than Wallace's approach but results in a wider final adder. Table I shows the difference in adder requirements for some common multiplier widths.

Robinson and Swartzlander [4] discuss a method to improve the delay of the bit matrix reduction; they state that the final adder is simply a "fast adder." In addition to treating the final adder stage as a "fast adder" many treat each node within a reduction matrix as having a single delay from input to every output, this is not the case for many full and a half adder implementations and is not assumed in the delay numbers reported herein.

TABLE I. Final Stage Adder Widths

Multiplier Type	Adder Width
32 Bit Dadda	61 Bits
32 Bit Wallace	56 Bits
16 Bit Wallace	25 Bits
16 Bit Dadda	30 Bits

The goal of improving delay in Wallace and Dadda multipliers in [4] is continued in this investigation by exploring the optimizations possible in the final adder stage. This paper presents an investigation of a hybrid approach to the final addition stage of both Wallace and Dadda multiplication algorithms when an accurate gate level delay model is analyzed in the matrix reduction. By studying the non-uniform arrival times in each bit position for both reduction methods, the critical path can be optimized by exploiting the timing characteristics of Carry Lookahead, Carry Skip and Carry Select adders.

In the next section the technology model and adder designs are discussed. Section III will show timing results of gate level models of the Wallace and Dadda bit matrix reductions. Section IV will utilize these timing distributions to optimize the final adder stage in a hybrid structure utilizing several desirable characteristics of Carry Lookahead, Carry Skip and Carry Select adders. Section V will summarize and include some concluding remarks.

II. Technology Model

A gate level structural Verilog model was developed. The typical 65nm technology device results of [5] are used as the gate delay model. In this study both rise and fall times are considered equal, hence a single delay number is reported. Table II shows the gates utilized along with their respective delays. The delay numbers were chosen from those simulated with a fixed output load of two minimum sized inverters. The

TABLE II. Gate Delays

Gate	Delay (ps)
Inverter	33.44
2-input NAND	43.92
3-input NAND	46.75
4-input NAND	60.21
2-input NOR	78.19
3-input NOR	47.85
4-input NOR	67.55

fan-out of two is chosen due to the adder designs utilized. The gate level implementations of the basic half adder and full adder that are used is shown on Figure 1. Other base modules were also implemented such as a 4 bit carry lookahead blocks, carry select, and carry skip adders. These designs have an average gate fan-out of two.

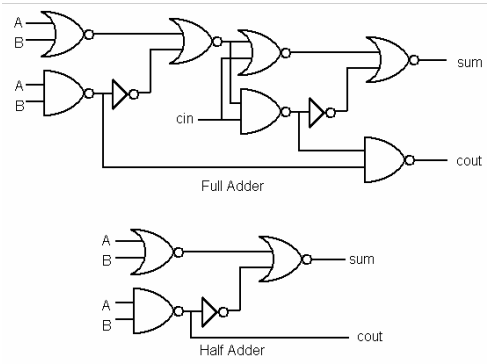


Figure 1. Adder Gate Level Implementation

III. Reduction Logic Timing Analysis

Both 32 and 16 bit Wallace reduction matrices were developed. Due to the wiring irregularities, the 32 bit Dadda reduction matrix proved challenging to implement so only the 16 bit version was developed and analyzed. A parallel design effort showed the overall distribution shape for both Wallace and Dadda design remained consistent for 16, 32 & 64 bit widths, hence the method below could be easily extended to these widths.

It is intuitive to expect non-uniform delays for each output of a bit matrix reduction, especially in the presence of (2,2) & (3,2) counters that have the delays shown in Table 3. Figures 2-4 show distributions of the worst case delays for each output bit of the respective reduction methods. Bit 1 is the least significant bit of each adder. These are the delays before being applied to the input of the final adder. These delays were achieved for the adder widths shown in Table I, the gate delays in Table II and the adder designs in Figure 1, all analyzed with Cadence's Primetime static timing analysis tool.

As mentioned earlier the Wallace and Dadda approaches differ in the method used in each stage of row reduction. The Dadda method only reduces to a specified row count per stage, thus requiring row reductions largely in the middle where the row count is the highest. This results in the skewed shape shown in Figure 3. The shape of the Wallace delay distribution turned out to be mostly uniform in the low order and middle bits, this is characteristic of the attempt to always reduce a row if it can hence in the center areas there are always reductions to be made. The steep slope on the high order bits of the Wallace reduction is due to requirement of using groups of three; many high order bits skip stages as they fall into a group of one or two, resulting in this shorter delay for these bits. The latest arriving bits of both the Wallace and Dadda reduction trees proved to be approximately the same; however the

TABLE III. Adder Delays

	Sum Out	Carry Out
Half Adder	0.13ns	0.08ns
Full Adder	0.26ns	0.22ns

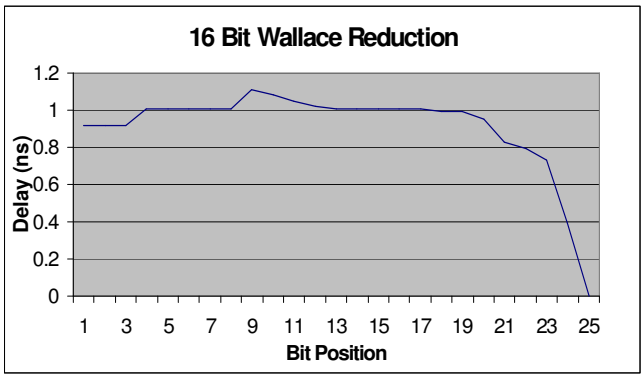


Figure 2. 16 Bit Wallace Reduction

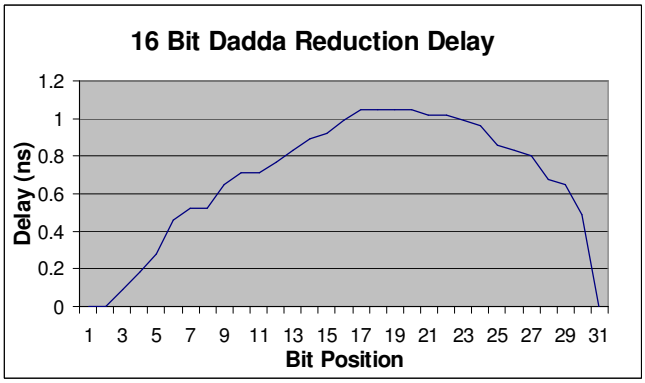


Figure 3. 16 Bit Dadda Reduction

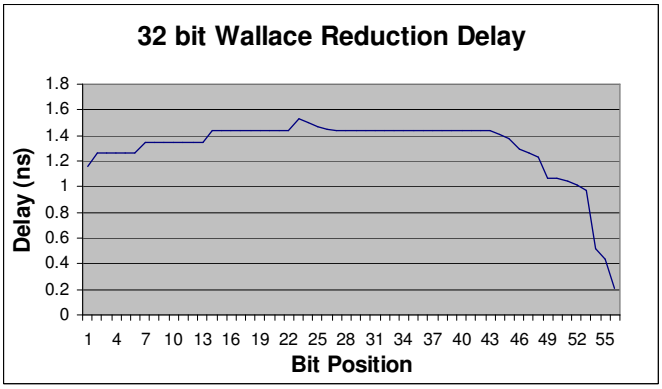


Figure 4. 32 Bit Wallace Reduction

difference in the adder widths and delay distribution shape resulted in different final adder design requirements.

IV. Final Stage Adder Design

In this section the relationship between the delay distributions in Figures 2-4 and a hybrid final adder stage are discussed. As a basis for comparison a carry lookahead adder

using 4-bit lookahead logic blocks is used as the baseline final adder implementation for all three designs.

A. Wallace Multiplier Final Stage Addition

The 32 bit Wallace reduction requires a final adder of 56 bits wide. Since the baseline carry lookahead adder utilized 4 bit lookahead blocks, a 64 bit carry lookahead adder is used as the base line as depicted in Figure 5. The critical path of this multiplier was 3.04ns.

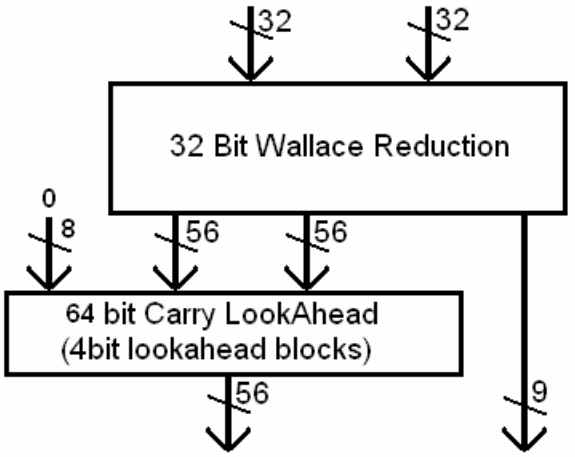


Figure 5. 32 Bit Wallace Baseline Design

Considering the arrival times of the different bit positions, a first (naive) step at optimizing the final adder is to split the arrival times graph of Figure 4 into two distinct sections. The first section spans from bit 0 to bit 47 and has the characteristic that the input bits have approximately the same arrival times (within 0.37 ns of each other, about a 14% difference). The second section is the sharp drop of arrival times towards the higher order bits into the final adder.

A carry look ahead adder is used for the low order 48 bits, and a carry select adder for the high order bits. Figure 6 shows a block diagram of this design. Since the high order bits arrived earlier than the lower order bits, double the hardware is used to pre-compute the results for both carry in possibilities. The effect is that as soon as this carry bit arrives, the result is simply selected quickly. This method results in a small speedup with a critical path delay of 3.00ns.

Developing the optimization further and analyzing the critical path, the small spike in the middle of Figure 4 at bit position 22 is in the critical path. Interestingly the final carry select portion is not in the slowest path; rather the slowest path is in the 48 bit carry lookahead section. When the latest bit arrives, it must traverse up and down three levels of carry lookahead logic to produce the final sum out. Since this is the case, it is logical to reduce the levels of carry lookahead logic that this late arriving bit must traverse. To gain this effect Figure 4 is divided further into three distinct sections. The first section is from bit 0 to bit 21. The second section consisted of the 4 latest arriving bit positions, 22 thru 25, this is the small spike shown in Figure 4. Finally the highest order 30 bit positions comprise the third section.

As before the first section uses a carry lookahead adder with 3 levels of lookahead logic. A ripple carry adder was considered for these bit positions, however, the slope of this section does not justify this; the delay of a ripple carry adder is greater than the difference in the longest and shortest delays across these bits, essentially the slope of the graph in Figure 4 along these bits is not steep enough. A ripple carry adder would be on the critical path, which is undesirable.

In the second section of the adder (bits 22 thru 25) an adder is needed to generate the carry to the next section quickly while minimizing the number of lookahead blocks each sum bit traverses. Hence a small 4 bit carry lookahead adder is used. This small carry lookahead broke down the critical path seen in the initial optimization.

For the highest order bits, the early arrival times are exploited, split carry select adders are used with carry-lookahead adders instead of the usual ripple carry adders. The final adder section is split from 30 bits into a 16 bit and a 14 bit modified carry-select adder; this is done to minimize the number of lookahead levels (two instead of three).

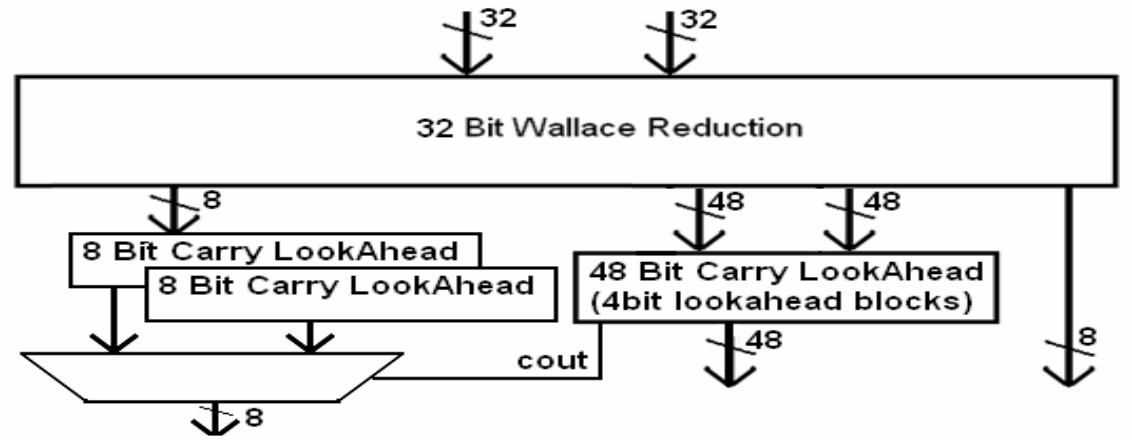


Figure 6. First Attempt at 32 Bit Wallace Final Adder Design

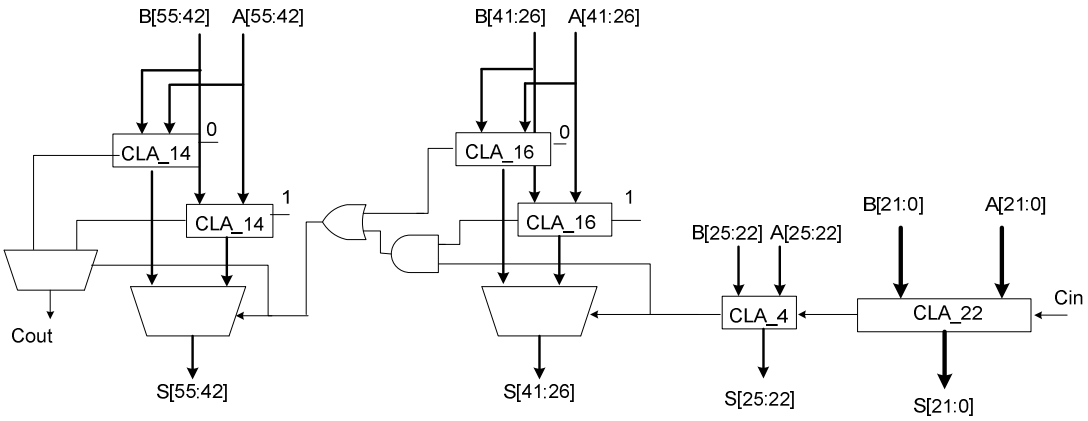


Figure 7. 32 Bit Wallace Hybrid Adder Design

The choice of a carry lookahead adder as the basis of the carry select adder is due to the fact that the low order bits of the third section arrive close in time to the high order bits of the second section. For this reason there is not much time to do the speculative sum calculation using ripple carry modules as is usually the case in carry select adders. The final hybrid design can be seen in Figure 7. Using this hybrid adder a 10.8% improvement in the critical path is measured with a critical path delay of 2.71ns.

The 16 bit Wallace multiplier is similar to its 32 bit counterpart. For the baseline of this multiplier a 32 bit carry look-ahead adder is used. Using this adder a delay of 2.44ns is obtained. Using a method similar to the approach described for the 32 bit case, an optimization is implemented as a hybrid adder using three sections for the 16 bit Wallace multiplier's final stage addition. The first section consists of an 8 bit carry lookahead adder. The second section which takes care of the latest arriving bit positions is a 4 bit carry lookahead adder. The third and fourth stages are carry select/carry lookahead adders of 4 and 9 bits, respectively. The arguments behind the choice of these sizes are similar to that of the 32 bit case. Using this structure a 7.8% improvement is measured in the critical path bringing it down to 2.25ns; a block diagram of this design in shown in Figure 8.

B. Dadda Multiplier Final Adder Stage

Based on the design of the 16 bit Dadda bit matrix reduction, the final adder width needed to be 30 bits. As a baseline a 32 bit carry lookahead adder is used as shown in Figure 9. The critical path delay for this design is 2.38ns.

The arrival times of the different bit positions of a Dadda multiplier are quite different from that of a Wallace multiplier. As seen from Figure 3 the distribution of arrival times is more evenly spaced on the Y axis compared to the arrival times of a Wallace multiplier in Figure 2. As before the final adder is divided into three sections based on the timing characteristics of Figure 3.

The first section consists of the low order 16 bits. These bits arrive at a rate suitable for a ripple carry adder as each adder would receive its input at approximately the same time as the rippled carry. This is a key difference from the Wallace design, that the slope of Figure 3 in the low order bits is much steeper than that of Figure 2. A ripple carry adder is a good choice as it is simple to design, requires less power and does not affect the performance.

The second section consists of bit positions 17 through 20 which are the latest arriving bit positions in the Dadda reduction. The same argument that was made for the Wallace

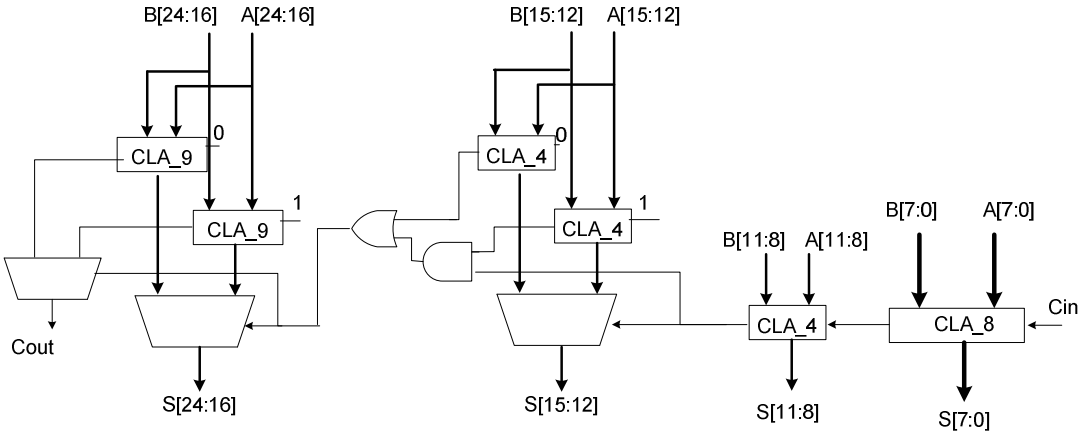


Figure 8. 16 Bit Wallace Hybrid Adder Design

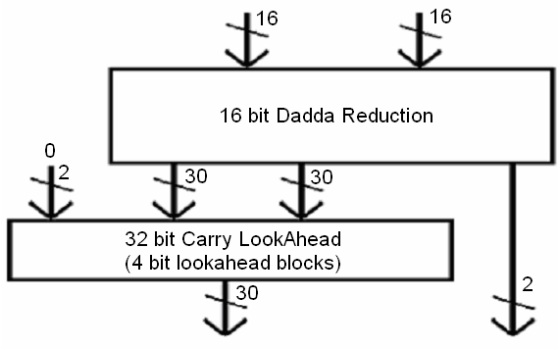


Figure 9. 16 Bit Dadda Baseline Design

reduction holds true for these late arriving bits, hence a 4 bit carry lookahead structure is used.

The third section of the adder involves carry select/carry lookahead hybrid blocks similar to the 32 bit Wallace design discussed earlier. However in this section three blocks of carry select are used which resulted in better performance than a single 11 bit carry select module. This is due to the steep drop off of arrival times of these high order 11 bits as seen in Figure 3. This is in comparison to the 16 bit Wallace multiplier which only had 5 bits that fell into this downward slope region. Using this design an 8.4% improvement in the critical path is measured, at 2.18ns. The final design of this optimization is seen in Figure 10. The results are summarized on Table IV.

V. Conclusions

The Wallace and Dadda approaches to binary multiplication are impressive designs for their ingenuity and effectiveness. There are two sections in which optimizations can be made in order to improve performance; the bit matrix reduction and the final adder stage. In this paper the final adder stage is investigated using standard Wallace and Dadda reduction algorithms. By analyzing the critical paths of these reduction strategies a suite of hybrid final adders were designed that consistently out performed standard carry lookahead adder designs. For the Wallace reduction a mixture of varying width carry lookahead and hybrid carry lookahead/carry select adders are used. With the design seen in Figure 7 a 10.8 % performance improvement is measured

TABLE IV. Performance Improvement Results

	Baseline	Hybrid	% Improvement
16 Bit Wallace	2.44ns	2.25ns	7.8%
16 Bit Dadda	2.38ns	2.18ns	8.4%
32 Bit Wallace	3.04ns	2.71ns	10.8%

for the 32 bit Wallace multiplier and a 7.8% performance improvement for the 16 bit Wallace multiplier as shown in Figure 8. The delay distribution for the Dadda reduction method led to a different design allowing the use of a ripple carry adder on the low order bits. Similar techniques used on the Wallace designs for the high order bits proved effective for the Dadda design. For the 16 bit Dadda multiplier, an 8.4% performance improvement is measured.

The small amount of extra hardware used is well justified since an 8% to 10% performance increase is realized. The extra hardware utilized is reasonable compared to the hardware employed in the bit matrix reduction when looking at the multiplier as a whole.

References

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

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

[3] L. Dadda, "On Parallel Digital Multipliers," *Alta Frequenza*, vol. 45, 1976, pp. 574-580.

[4] M. E. Robinson and E. E. Swartzlander, Jr., "A Reduction Scheme to Optimize the Wallace Multiplier," *International Conference on Computer Design,* Austin, TX, October 5-7, 1998, pp. 122-127.

[5] E. Ebrahimi and M. Sadoughi, "Standard Cell Library Characterization in 65nm Technology," Project for VLSI 2 Course, Spring 2007. (Results can be found at: http://www.ece.utexas.edu/~ebrahimi/Delay values FO1-FO8-Worst case27-3.xls)

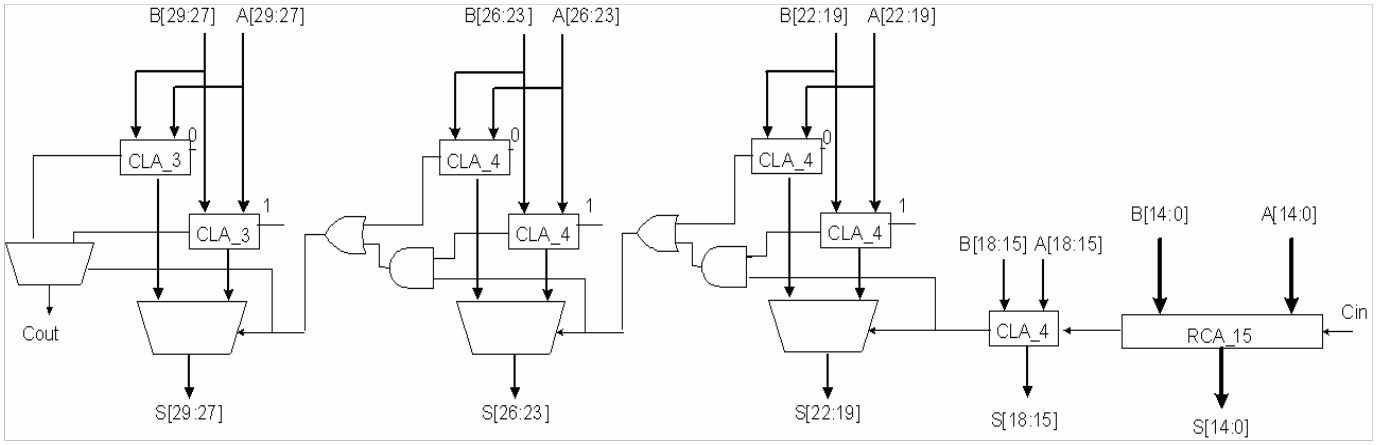


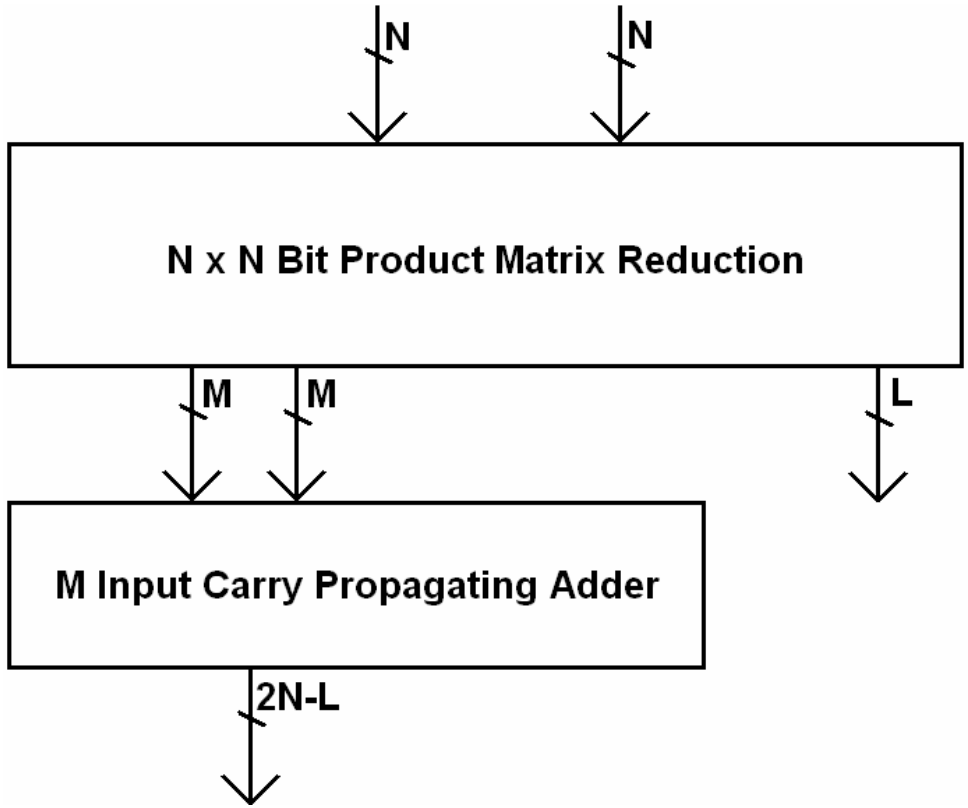
Figure 10. 16 Bit Dadda Hybrid Adder Design

A Hybrid Approach To Last Stage Addition For Wallace and Dadda Multipliers

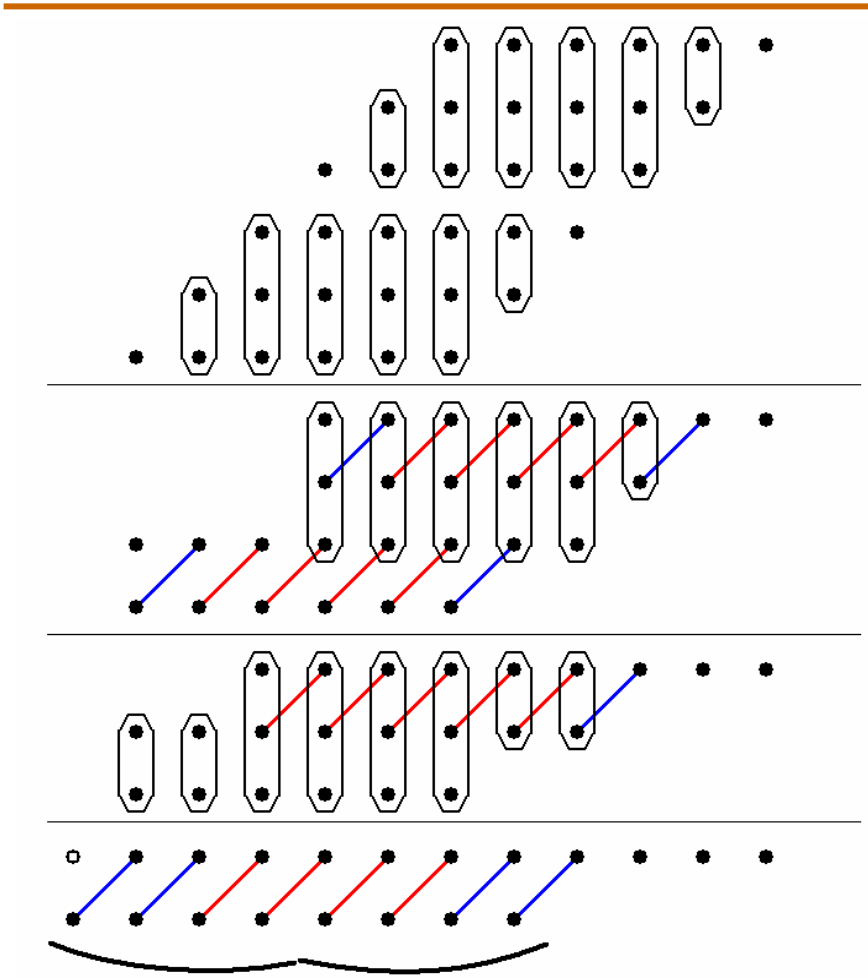
James Haydn Nelson, Eiman Ebrahimi,
Mahnaz Sadoughi, Earl E. Swartzlander Jr.

Wallace and Dadda Multipliers

- Final Adder Stage
  - "Fast Adder"
- Adder Options
  - Ripple Carry
  - Carry Lookahead
  - Carry Skip
  - Carry Select
  - Hybrid
- Adder Model
  - Gate Level
  - Single Delay ?

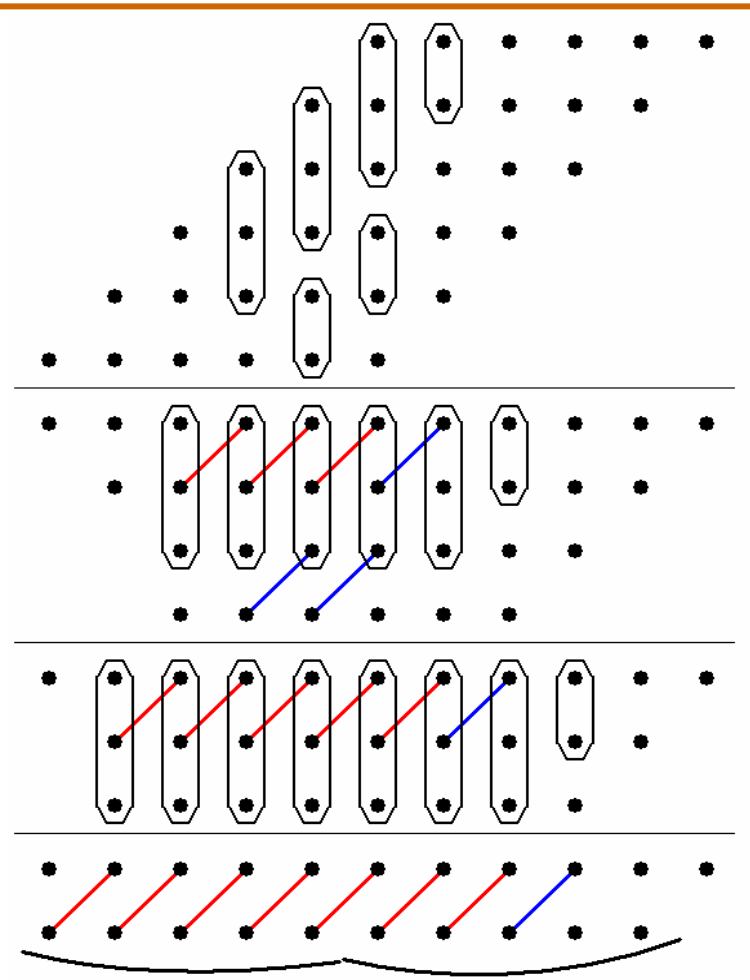


Wallace

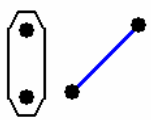


8-input Carry Propagating Adder

Dadda



10 Input Carry Propagating Adder



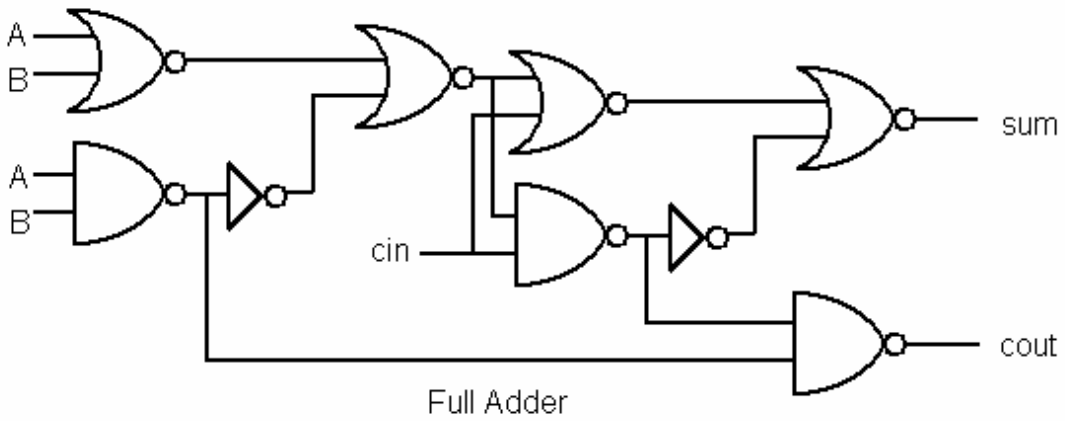
Half Adder

Full Adder

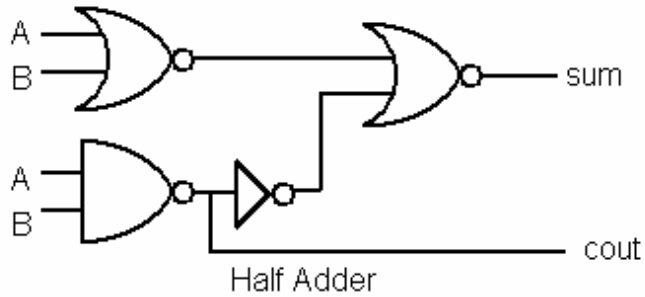
Carry Propagating Adder Width

Multiplier Type	Adder Width
32 Bit Wallace	56 Bits
32 Bit Dadda	61 Bits
16 Bit Wallace	25 Bits
16 Bit Dadda	30 Bits

Adder Delays in 65nm CMOS



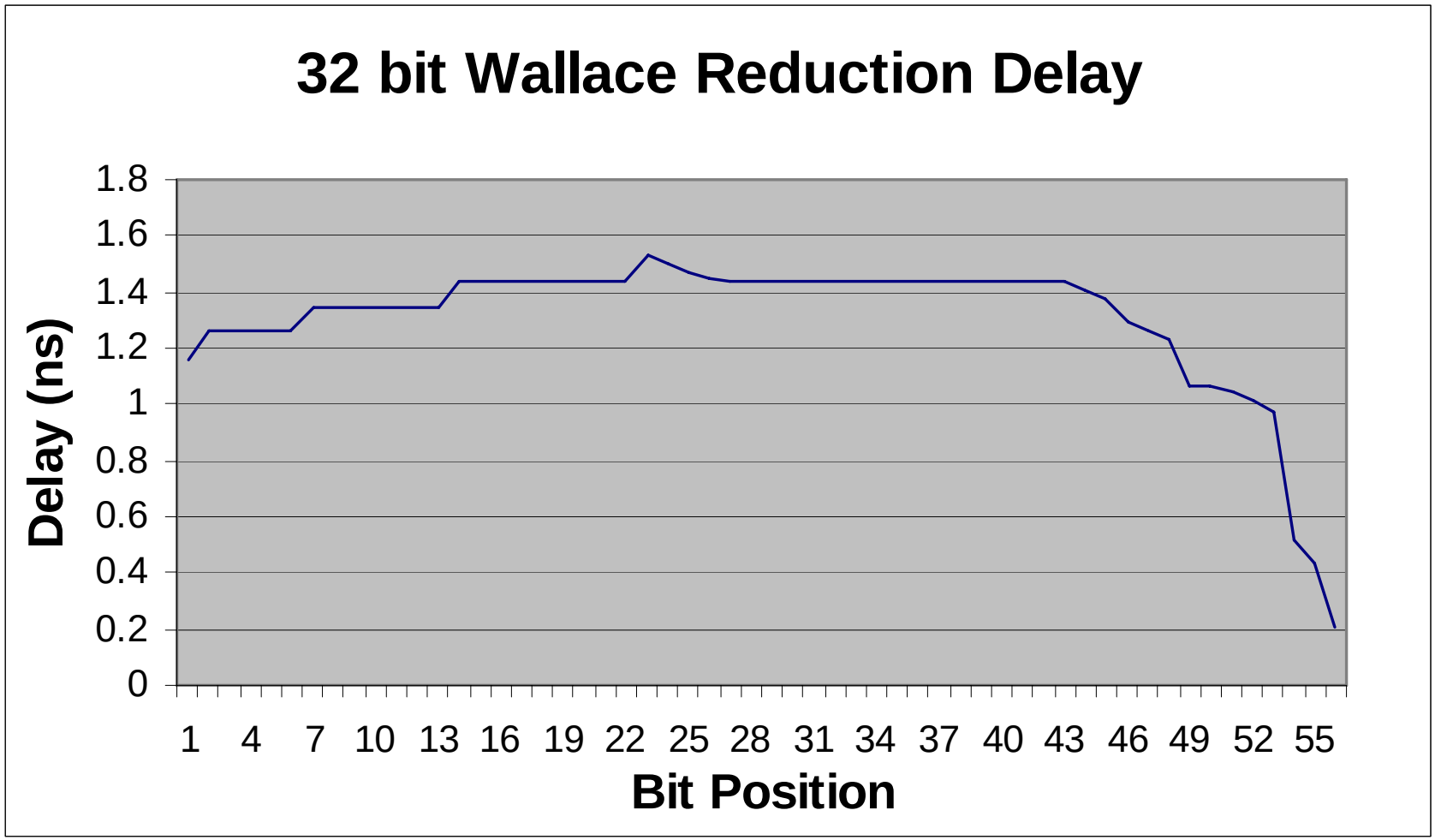
Full Adder



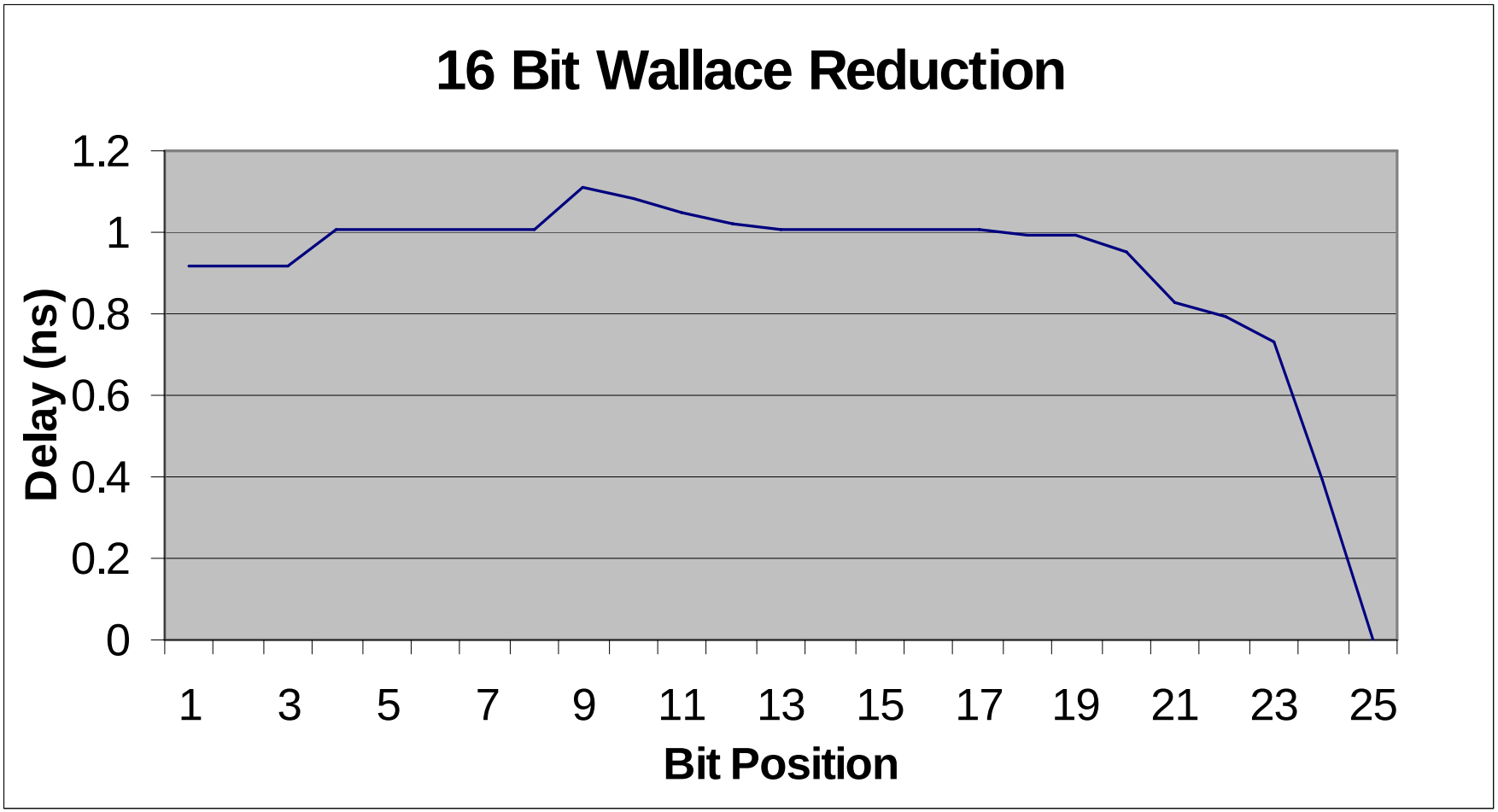
Half Adder

FO2	Sum Out	Carry Out
Half Adder	0.13ns	0.08ns
Full Adder	0.26ns	0.22ns

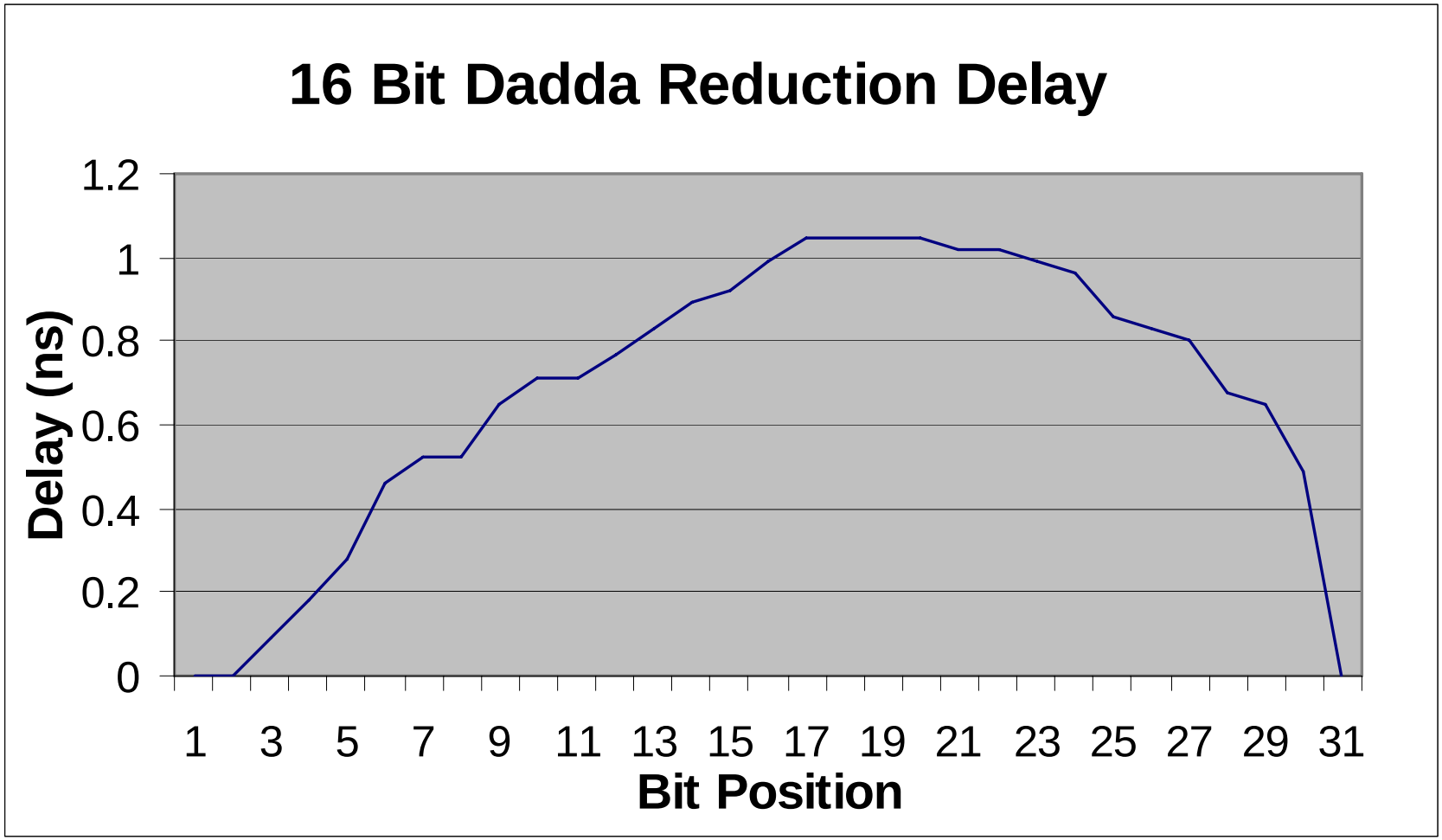
32 Bit Wallace Reduction



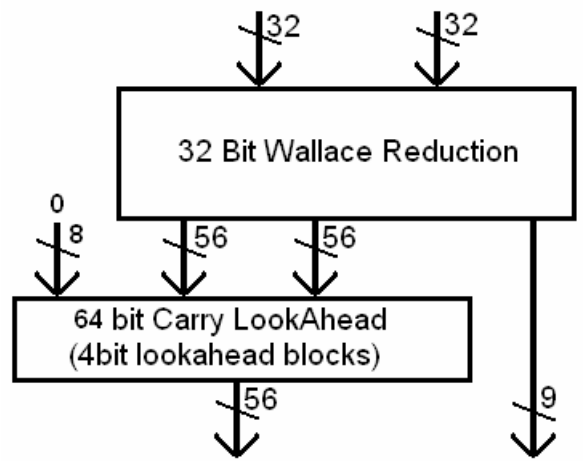
16 Bit Wallace Reduction

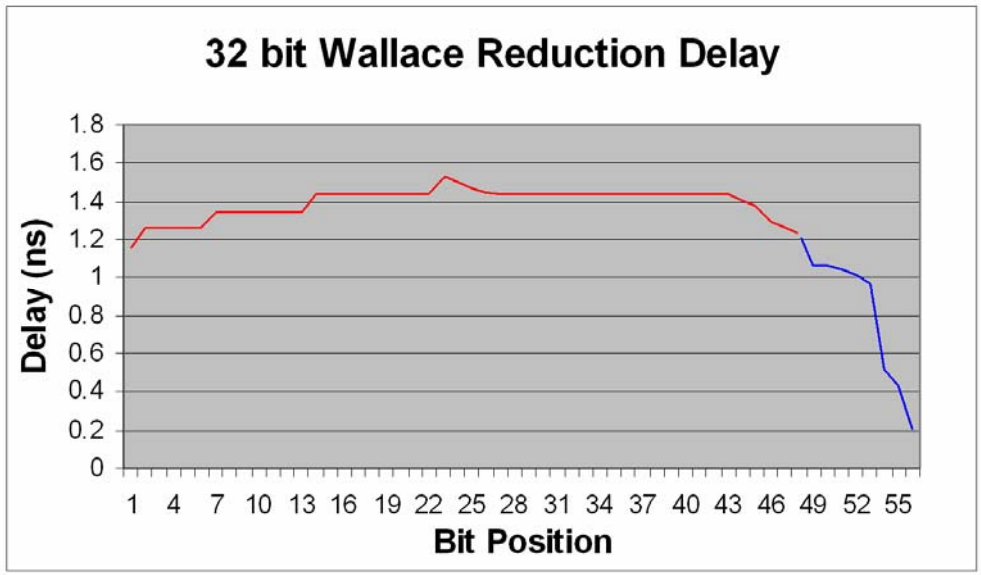


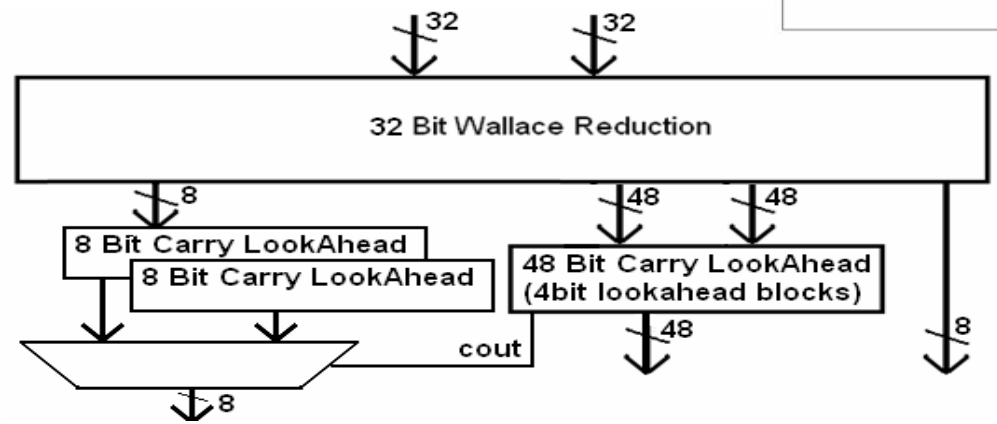
16 Bit Dadda Reduction



Naïve Wallace Hybrid Final Adder



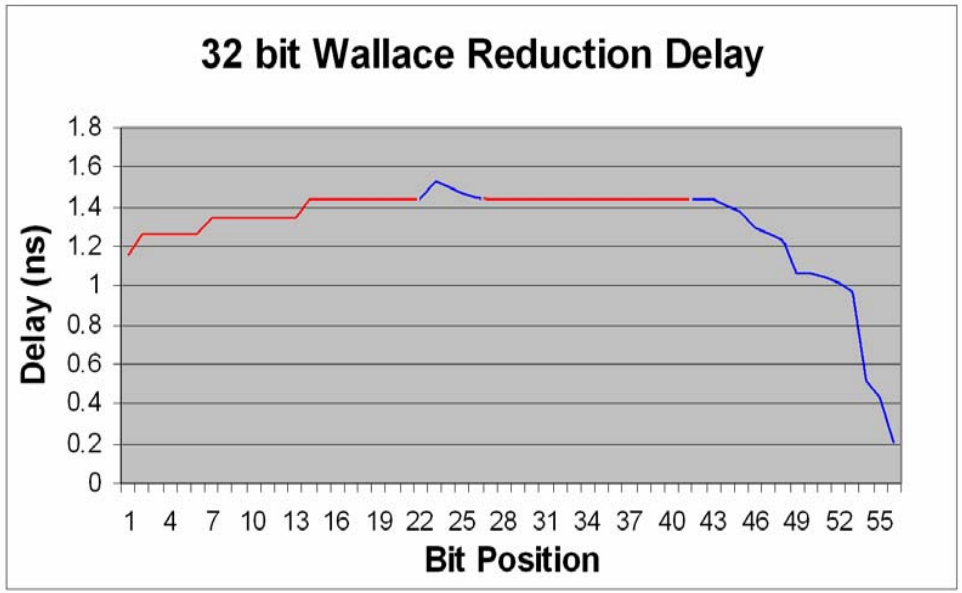


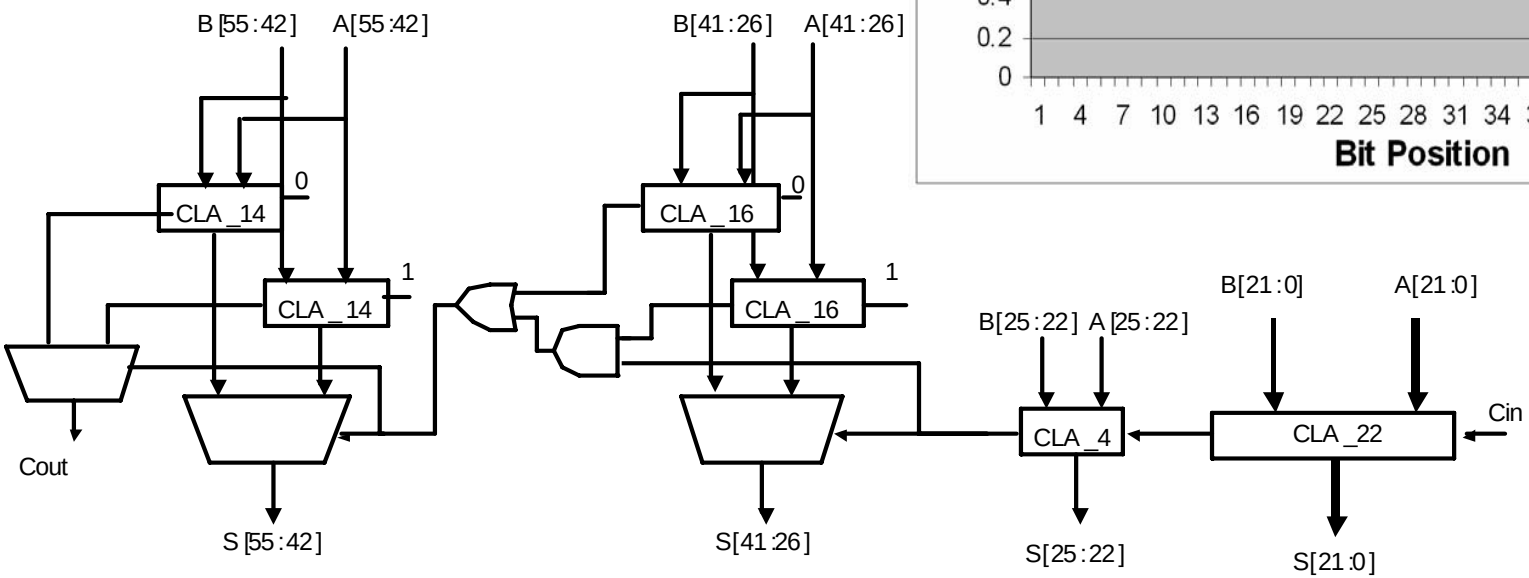


Multiplier Type	Delay
Base Line	3.04 ns
Naïve Hybrid	3.00 ns

32 Bit Wallace Hybrid Final Adder

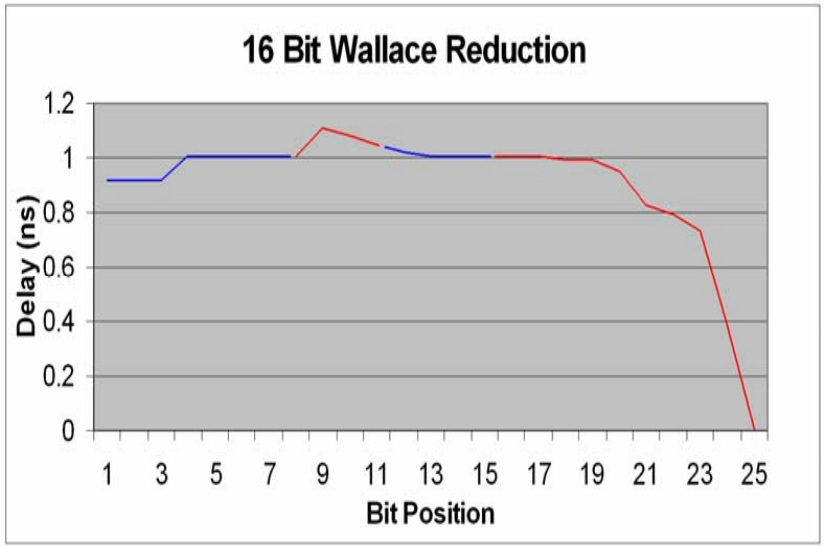
Multiplier Type	Critical Delay
Base Line Design	3.04 ns
Naïve Hybrid	3.00 ns
Hybrid Wallace	2.71 ns (1.12x)

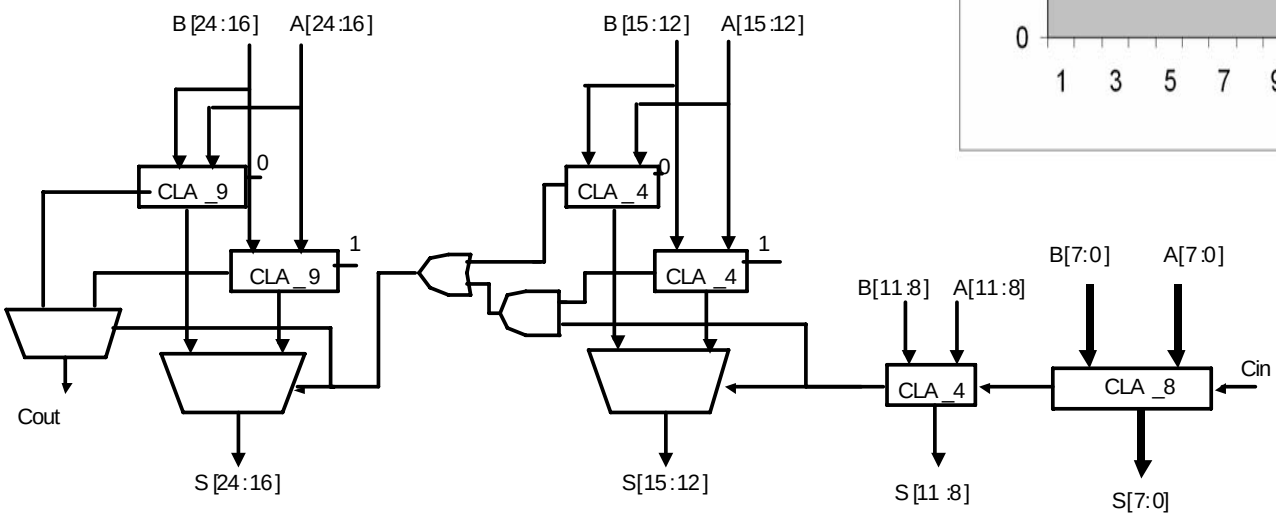




16 Bit Wallace Hybrid Final Adder

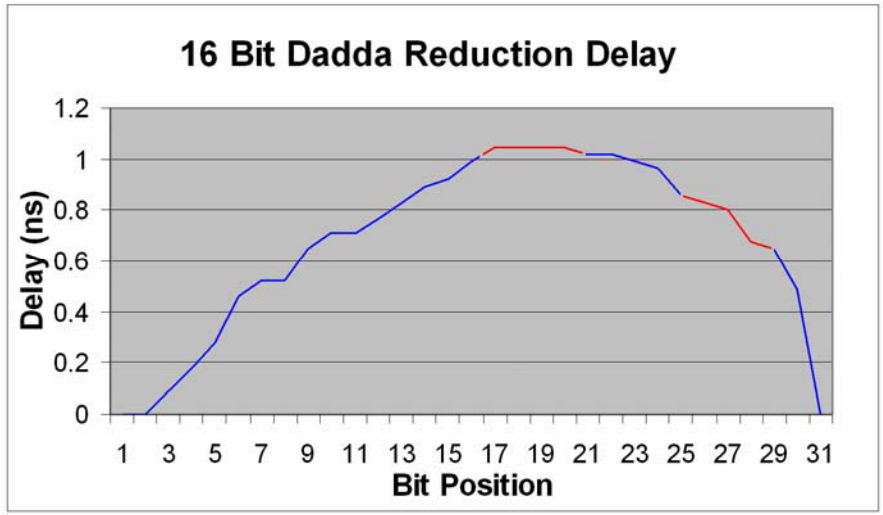
Multiplier Type	Delay
Base Line	2.44 ns
Hybrid	2.25 ns (1.08x)

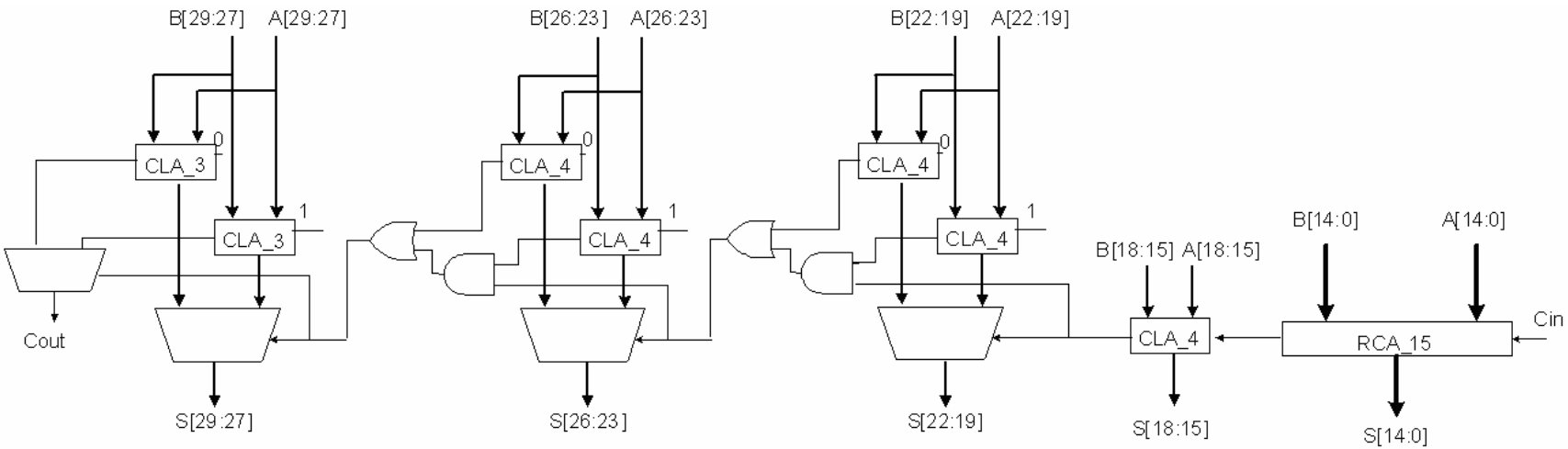




16 Bit Dadda Hybrid Final Adder

Multiplier Type	Delay
Base Line	2.38 ns
Hybrid	2.18 ns (1.09x)





Summary

- Wallace Multiplier Design
  - Similar Timing Shape for 16 & 32 Bit Versions
  - Trailing Edge Not in Critical Path
  - 32 Bit : 1.12x Improvement (CP: 2.71 ns)
  - 16 Bit : 1.08x Improvement (CP: 2.25 ns)

- Dadda Multiplier Design
  - Expect Similar Timing Shape for 32 Bit Version
  - Slope Steep Enough to Allow RCA
  - 16 Bit : 1.09x Improvement (CP: 2.18 ns)
  - Perhaps Less Hardware Than Wallace
  - Faster Critical Path

References

- [1] C. S. Wallace, “A Suggestion for a Fast Multiplier,” *IEEE Transactions on Electronic Computers*, vol. 13, 1964 pp. 14-17.
- [2] L. Dadda, “Some Schemes for Parallel Multipliers,” *Alta Frequenza*, vol. 34, 1965 pp. 349-356
- [3] L. Dadda, “On Parallel Digital Multipliers”, *Alta Frequenza*, vol. 45, 1976, pp. 574-580.
- [4] M. E. Robinson and E. E. Swartzlander, Jr., “A Reduction Scheme to Optimize the Wallace Multiplier,” *International Conference on Computer Design,* Austin, TX, October 5-7, 1998, pp. 122-127.
- [5] E. Ebrahimi and M. Sadoughi, “Standard Cell Library Characterization in 65nm Technology,” Project for VLSI 2 Course, Spring 2007. (Results can be found at: http://www.ece.utexas.edu/~ebrahimi/Delay values FO1-FO8-Worst case27-3.xls)

Questions ?