Final Report for ECE 55900: The Wallace Tree Hardware Multiplier

Aidan Prendergast and Malcolm McClymont

***Abstract*—Modern microprocessors and SoC-based accelerator modules make common use of integer, floating-point, and matrix multiplication operations within standard workloads. The emergence and dominance of AI systems in the market has placed a unique weight on matrix multiplication. All of these operations are rooted in integer multiplication, requiring blazing-fast hardware units to support gigahertz-speed pipelines and high-bandwidth multiply-accumulate operations. Combinational multipliers, in particular, require unique tricks in order to improve measurable metrics: compressor trees like the Wallace or Dadda; High-Speed final adders like the Kogge-Stone, Brent-Kung, and Han-Carlson; and encoding schemes such as Booth encoding. Tradeoffs in these designs must be evaluated when designing for speed, for area, or for power, and middle-grounds must be found when multiple metrics are desired.**

This work presents the design process and layout for a Wallace Tree-based combinational 8-bit multiplier. The design targets layout area as its parameter for minimization, undertaking a number of optimizations such as use of inverting logic, transmission gate-based cell design, and standard-cell-style layout in order to maximize performance and density.

***Index Terms*—VLSI, Cadence, Multiplier, Wallace Tree, Brent-Kung, GPDK045, Standard Cell Library, Multiply-Accumulate (MAC).**

I. Introduction

INTEGER multiplication is a core operation across microprocessors, application-specific integrated circuits, and accelerators for Digital Signal Processing, AI Compute Acceleration, or other general-purpose matrix computation hardware. Whether integer, fixed-point, or floating-point in scope, any multiplier circuit has an integer multiplication at its core.

Multipliers come in both sequential and combinational flavors. For ultra-highspeed purposes, these two approaches are often melded to enable fast, dense, pipelined multiplication. Even so, the core multiplication algorithm remains the same, drawn from the combinational multiplier architecture:

1) Calculate Partial Products
2) Compress Partial Products
3) Final Addition (If needed)

These steps are drawn specifically from a tree multiplier approach, where the N^2 partial products are compressed down with a tree of half-adders (2-to-2 compressors) and full-adders (3-to-2 compressors). The two most common approaches are the Wallace Tree [1] and the Dadda Tree [2], which compress the set of partial products in different ways. The request for this project was to use a Wallace Tree, so we avoided the Dadda approach, even though it saves area in some cases.

These trees both leave a final set of bit pairs, typically handled by some form of fast adder. The best adders in this realm include Carry-Select [3], Carry-Skip, Carry-Save, Brent-Kung [4], Kogge-Stone [5], and Carry-Lookahead. All designs take varying approaches to producing the carry-in for a given bit faster than the simple ripple-carry approach.

II. Design

A. Partial Product Generator

The partial product generator takes the two input vectors and computes the bitwise product of each pair of bits, akin to how one would compute each single-digit product when performing longhand decimal multiplication. This bitwise product is performed using an AND gate for each of the N^2 pairs. The partial product generator also arranges these bits in a convenient way for the Wallace Tree to process them, much like how each row is staggered when performing longhand decimal multiplication with numbers larger than one digit.

B. Wallace Tree Partial Product Compressor

The partial product generator outputs 64 bits that must be shifted and added to create the resulting product. Although a series of 8 adders could accomplish this, such an approach requires and excess of power and area compared to other possible strategies. Instead, the partial products can be combined in a bit-wise manner to reduce the number of partial products. This allows for a smaller, 10-bit adder to sum the remaining products.

The Wallace Tree consists of 5 layers of full adders (FAs) and half adders (HAs). Each FA and HA adds two or three bits with the same place value, producing a sum bit with the same place value and a carry bit of one higher place value. This process is visualized in Figure 6.

C. 10-Bit Brent-Kung Parallel-Prefix Adder

For the final adder, the Wallace Tree yields 10 bit-pairs to perform an addition operation on. This adder has a restricted set of input pairs such that it will never produce a carry-out from the MSB, allowing the 9th bit to have reduced hardware complexity.

Design investigation began with a Ripple-Carry Adder, as it is the maximally dense hardware for the addition operation. The worst-case time for the ideal schematic was found to be around 1000ps, which was beyond acceptable limits as the layout parasitics would threaten to bring the design above the given 3ns latency requirement. Thus, investigation moved onto the family of Parallel-Prefix Adders (PPAs), a faster approach.

Serious consideration was given to the Kogge-Stone and Brent-Kung adders, with some thought also given to Han-Carlson and Ladner-Fischer approaches, which are less well-documented. Parallel prefix adders focus on the architecture used to generate the complete carry-in signals for each full adder, typically via a tree structure.

Cells common to these trees are the Black Cell and Gray Cell, colored accordingly in the below diagrams. These are universal terms across the design of mathematical circuits, combining propagate and generate terms from adjacent ranges to encode a broader span of information.

Kogge-Stone adders require substantial duplicate hardware as they target high-speed prefix calculation, as seen in Figure 1. The circuit area is large relative to other approaches due to having 34 Black cells and 15 Gray cells for a 16-bit implementation. The excess of cells results in tree compression down to a 4-stage critical path, producing a very-high-performance adder.

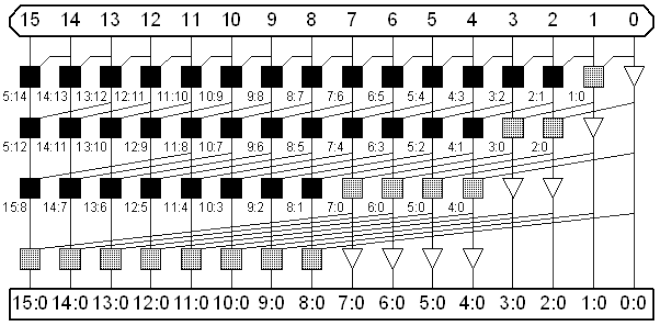


Fig. 1: The Parallel-Prefix Calculation for a Kogge-Stone Adder. Generated using IamFlea's AdderCircuitGenerator on GitHub.

Brent-Kung Adders are far less performance-focused, instead opting to decrease circuit footprint, as highlighted by Figure 2. This makes them an attractive option for an area-optimized multiplier. With only 11 Black Cells and 15 Gray Cells for a 16-bit implementation, the computation has up to 7 stages, compromising on speed. The BKA was the PPA explored first due to these attractive qualities.

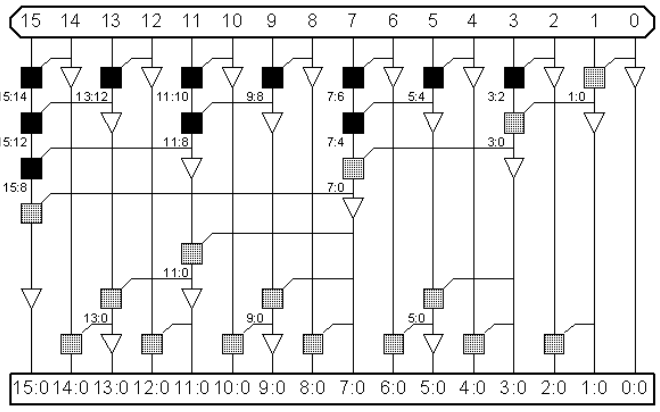


Fig. 2: The Parallel-Prefix Calculation for a Brent-Kung Adder.

The BKA was observed to have a worst case delay of 277ns without parasitics, making it sufficiently fast to meet the project requirement, even once nonidealities were added post-layout. Thus, the Kogge-Stone Adder was not explored further. The 10-bit adder schematic was connected as seen in Figure 3, with square symbols representing black and gray cells.

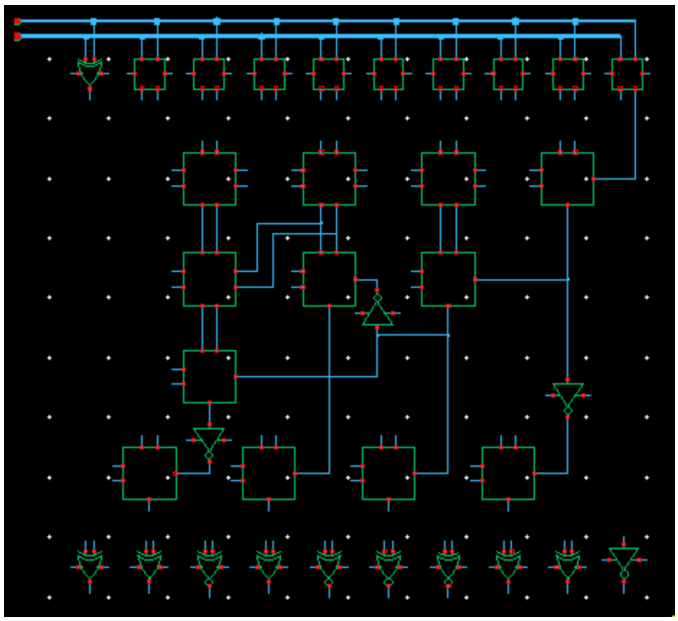


Fig. 3: Cadence Virtuoso Schematic for a 10-bit Brent-Kung Adder

D. Custom GPDK045-Based Standard Cell Library

The GPDK-045 technology node was provided as the target node for the project. Capacitors, NMOS transistors, and PMOS transistors were provided, but constructing standard cells from which to build modules was left as a requisite stepping-stone for making progress on the multiplier.

Our implementation opted for minimum-sized transistors, a 45nm gate length on this node. Although delay for a reference inverter is not equalized with this approach, causing an unoptimal delay, striving towards the minimum area target and an expectation of some breathing room on timing constraints meant minimum-sizing would be a worthy tradeoff.

Standard cells made included an inverter, nand2, nor2, and2, or2, a transmission gate, xor, xnor, aoi2x1, oai2x1, a half-adder, a full-adder, a gray cell, and a black cell. Most of these cells have very typical CMOS implementations that are dense, fast, and straightforward, but some are less clear-cut. For the non-inverting gates, AND and OR, an inverting gate was paired with an inverter.

The XOR and XNOR gates presented a unique design challenge: traditional CMOS XOR gates use 12 transistors for a 2-input logic function. As the design heavily emphasizes half and full adders, XOR gates are a large portion of any critical path. Thus, their implementation must carefully considered for a successful multiplier design. Through research into works such as [6], [7], and wiki resources, an 8T Transmission-Gate XOR approach was found. This approach uses 4 fewer

transistors in the main body of the cell, which decreases the area due to less overhead for spacing gate contacts. The 8T version also presents with faster transitions in all 16 transition cases except the 11→01 case (not the worst case delay), with a lower average delay by almost 30ns. Its schematic is displayed in Figure 4.

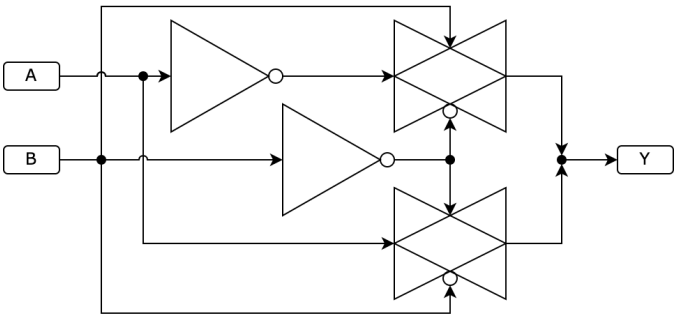


Fig. 4: Schematic for a Transmission-Gate XOR

The full adder is another complex gate to design, and many approaches have been documented [6] [7]. Our sum logic used a standard XOR-XOR approach with transmission gates, but our carry logic is nonstandard. The $AB + AC + BC$ majority logic term can be re-expressed more efficiently when the $A \oplus B = X$ term already is present in the circuit from the sum calculation. This rearrangement results in $CX + A\bar{X}$ for the carry-out, or alternatively $CX + B\bar{X}$, noting that A and B are equal in the case that $\bar{X} = 1$.

Using this logic results in a design as seen in Figure 5.

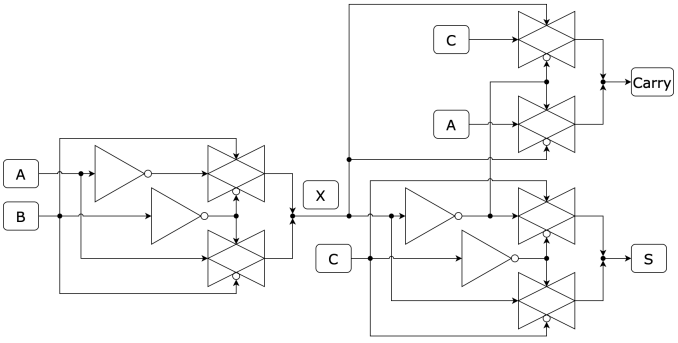


Fig. 5: Schematic for a Transmission-Gate Full Adder

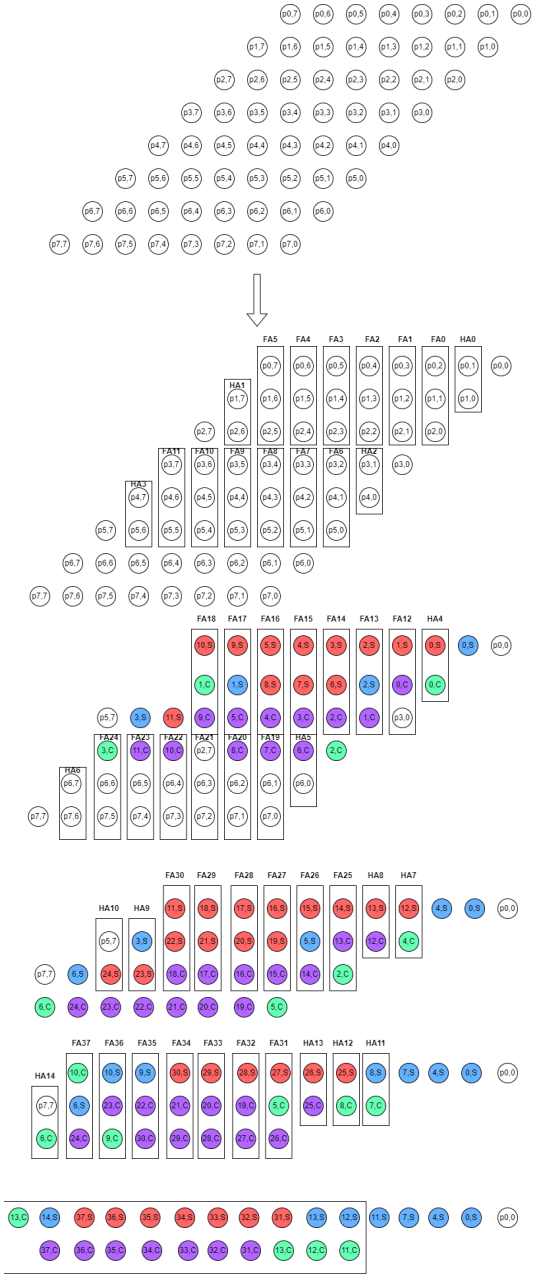


Fig. 6: Algorithm for a 16-bit Wallace Tree. Blue and green outputs correspond to HA sum and carry out bits respectively. Red and purple outputs correspond to FA sum and carry out bits.

E. Layout

A similar motivation dominated the choices made regarding layout to decisions for architecture and transistor sizing: as minimum area was the ultimate goal of the design, NMOS and PMOS transistors were put as close together as possible for the reference inverter, and the power rails were packed a minimal possible distance as well. This resulted in a cell height of 0.8µm with transistors in a vertical-gate orientation. This measurement constituted the row height when stacking cells together.

Our area goal also provoked a preference for going up rather than out when dealing with crossing or adjacent wires. Thus, we iterated and arrived at the following system:

- Poly, Metal 1, Metal 2: Used for laying out primitives (NOT, NAND, NOR, TX, AOI, OAI) and other routing when convenient for higher-level cells.
- Metal 3: Used for internal routing in more complex cells (XOR, XNOR, Gray Cell, Black Cell) and other higher-level routing when convenient.
- Metal 4: Used for internal routing in the most complex cells (HA, FA) and at the module level.
- Metal 5, Metal 6: Used for Vertical and Horizontal global routing, respectively.

The understanding was that this approach would constitute

a performance penalty due to contact and via resistances and capacitances. The design density would also make necessary small, high-resistance wires, further decreasing performance, but still minimizing area.

With rigorous, robust cell design, and learned heuristics, i.e. keeping wires a minimum of $\frac{1}{2}$ the PDK's minimum spacing from cell borders, layout was made far easier than a haphazard approach.

III. Optimizations

The multiplier design we chose includes 110 XOR gates and an additional pseudo-XOR included in each of the 38 full-adders. Thus, most of the effort put towards optimizing the design revolved around these gates, or towards global optimizations.

The first optimization occurred in the design phase, when the internal $\bar{X}$ signal of the FA's sum computation was able to be reused for the carry logic's TG enables. This saved an external inverter on the full adder, reducing its size from 22T to 20T.

There exist a class of pass transistor-based 4T-XOR and 4T-XNOR designs [6], and these were implemented within the standard cell design as an attempt to save even more area. However, their reliance on pass transistors make them slow under certain conditions, and they have issues with weak pull-up to V_{dd} using NMOS. Additionally, under capacitive load, select transitions were seen to be in the realm of 550ps, invalidating them as a candidate. A variation that used a 4T XNOR with an inverter on the output was also explored, but failed to produce a significant speedup, even with only a single inverter loading the pass transistor.

One observation made was the number of inverters contained within cells. Standard half-adders use AND gates for carry logic, requiring an inverter. Gray cells and black cells used and-or trees, comprised of an AOI and an inverter [8]. The partial product generator was entirely built of AND gates. Creating an alternating tree structure in the Wallace Tree, the parallel prefix tree, and constructing the partial product generator to produce inverted outputs using NAND cells could greatly reduce the area. Thus, we designed half-adder, full-adder, black cell, and gray cell variants for Positive-Input/Negative-Output and Negative-Input/Positive-Output combinations. These additional cells were instrumental in saving $\sim$150 inverters ($\sim$71.4 μm^2) within the design. In no case did the inverter count increase for an individual cell, as the XOR and XNOR designs are of the same size and could be swapped. At the module level, a few spots required re-addition of the previously removed inverter to preserve polarity, but the change still resulted in large overall savings.

Many other optimizations were considered but not pursued either due to project scope constraints or due to meager benefits yielded from substantial effort. For the Wallace tree, 4-to-2 and larger compressors were investigated as a way to shrink the tree size. However, a 4-to-2 compressor requires 2 FAs, negating any significant benefits. Additionally, 4-to-2 compressors introduce a layer of complexity that complicates the addition of more beneficial optimizations. Low-Voltage-Threshold transistors were considered as a potential improvement for pass transistor designs, but lacking time, an uncertainty surrounding layout specifics of LVT models in the GPDK045 technology node, and focus on other optimizations prevented exploration of this unique approach. Lastly, we found gate-input optimization unnecessary, as each individual structure proved fast enough to abstain from further path/cell-to-cell optimization for the design.

IV. Energy-Latency-Area analysis

A breakdown of these metrics for standard cells can be found in section A. As for the higher-level modules, the following was extracted from our work:

Cell	Area (μm^2)	Delay	Energy
Partials	44.672	29 ps	10fJ
Wallace Tree	328.96	1.24 ns	170fJ
Final Adder	74.035	694ps	59.39fJ
Top-Level	407.108	1.76ns	479fJ

This design is within the given target metrics given of 3ns maximum delay and 1000fJ of power draw. The worst input vector occurs when bit 4 of input A is high and bit 3 of input B rises. Increasing the total number of transitions increases the energy consumption. Therefore, the worst-case input vector is 10 00 $\rightarrow$ FF FF. In this case, a propagation delay of 1.76 ns is observed, and the design consumes 478.566 fJ.

Due to substantial optimizations and deft design, the layout of the multiplier has an area of 407 μm^2. When compared with a non-inverting version, the updated inverting schematic consumed less power on top of proving faster and lower area.

V. Conclusions

This project served as both a great success and a fantastic learning experience. To wrap up simply: The design worked and hit substantially below the required metrics. With a worst-case latency of 1.76ns and an energy usage of 479fJ, the design uses between 45% and 60% of the delay and power budget available to it. With such margins available between the requirements and results, there is yet more redesign that could be undertaken to continue to shrink the area footprint of the design further.

We are particularly curious regarding how the low-voltage transistors would affect the pass-transistor XOR operation. More delay would be acceptable for an area reduction, as there are $\sim$ 1.25ns to sacrifice in exchange for lower cell area. Additionally, there is likely a layout density improvement we could find, as shown by the fully integrated multiplier having small holes in it. We estimate the design area could be reduced by 5-10% by exploiting these regions for better compression of the standard cells. Lastly, we may be able to use this extra timing delay to revert to a simpler adder, such as a ripple-carry or a simple carry-select.

We are proud of our efforts, achievements, and the payoff of the complicated decisions we spent time mulling over. We'd like to thank Professor Sumeet Gupta for his excellent instruction of a compelling, well-designed VLSI course.

VI. Contributions

Aidan Prendergast focused his efforts primarily on the adder design, standard-cell design, layout rule construction, and topological optimization of cells and inverters. This included construction of a SystemVerilog model of the Brent-Kung Adder to verify its functionality, comparison of Ripple-Carry and Brent-Kung adders in Spectre simulation, Standard Cell schematics and layout, Standard Cell re-layout to optimize cell interconnectability, half-adder and full-adder design, and additional design of inverting variants of cells.

Malcolm McClymont focused primarily on the schematics and layout of the high-level modules, especially the partial product generator and Wallace Tree. Both modules were modeled and their connectivity verified in SystemVerilog. He also worked on the bring-up of NanoTime scripts and evaluation tooling for Static Timing Analysis and Worst-Case Input Vector Extraction. He performed the re-schematic of the Wallace Tree to cut out inverters and laid out the Partial Product Generator, Wallace Tree, and Top-Level Multiplier.

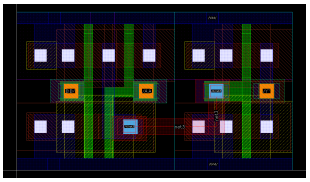
Appendix A
Standard Cell Metrics

All cells occupy one 0.8µm-tall row, including power and ground rails. Metrics were calculated from cell schematic sizing and worst-case input vector computation. Cells with more than one entry correspond to different variants, such as the Negative-to-Positive and Positive-to-Negative half adder layouts.

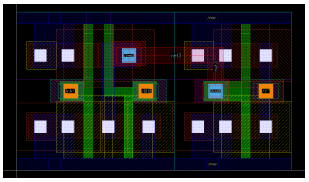
Cell	Area(s) (μm^2)	Worst Delay (ps)
NOT	0.476	12
NAND2	0.64	19
NOR2	0.64	25
TG	0.584	6
AND2	1.116	25
OR2	1.116	33
AOI	0.82	37
OAI	1.012	35
XOR	2.12	29
XNOR	2.12	29
Gray	0.82, 1.012	37
Black	1.46, 1.652	37
HA	2.76, 3.236	29
FA	5.408	62

Appendix B
Standard Cell Layouts

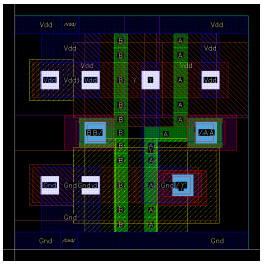
The following are imagery detailing the layouts used for each of the designs required for the Wallace Tree Multiplier. Much larger designs, from the full-adder to the top-level, are exhibited on subsequent, auxiliary pages.



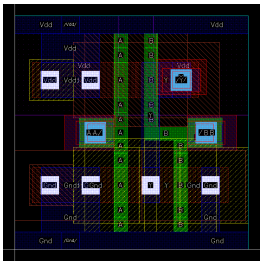
AND Gate



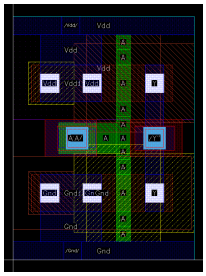
OR Gate



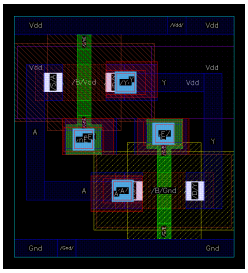
NAND Gate



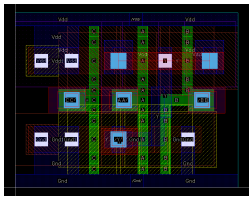
NOR Gate



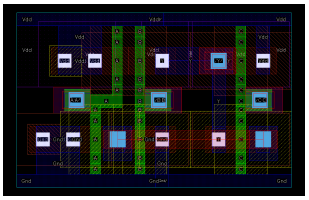
NOT Gate



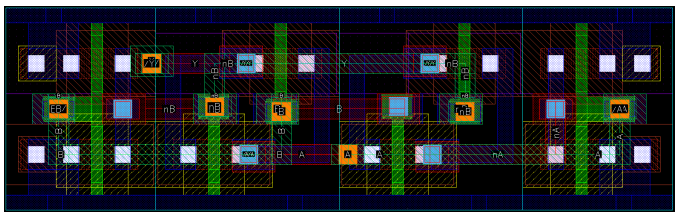
Transmission Gate



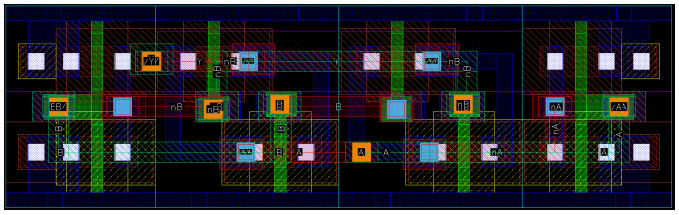
AOI Gate



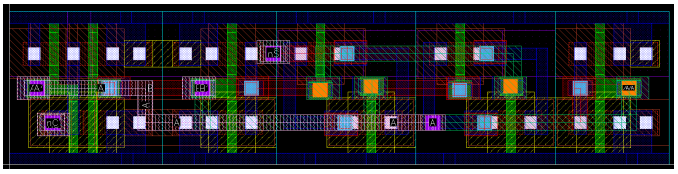
OAI Gate



XOR Gate



XNOR Gate



Half-Adder Module

APPENDIX C
PROOF OF FUNCTIONALITY

The images presented in Figure 11 and Figure 12 are the Design Rule Check and Layout-Versus-Schematic (including Electrical Rules Check and Extraction Steps) results, proving the Wallace Tree design follows the Layout and Connectivity requirements set forth by the GPDK045 Technology Node and Schematic Design of the module.

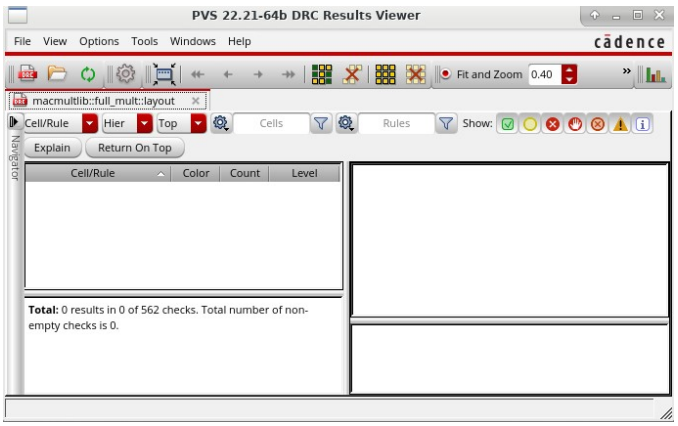


Fig. 11: Design Rule Check Results for the Full Multiplier

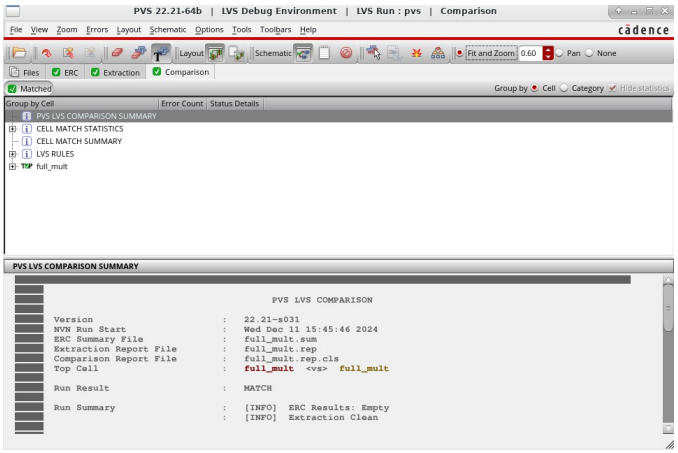


Fig. 12: Layout-Versus-Schematic Results for the Full Multiplier

REFERENCES

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

[2] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, 1965. [Online]. Available: https://www.ece.ucdavis.edu/~vojin/CLASSES/EEC280/Web-page/papers/Arithmetic/Dadda_mult.pdf

[3] O. J. Bedrij, "Carry-select adder," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 3, pp. 340–346, 1962.

[4] Brent and Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

[5] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

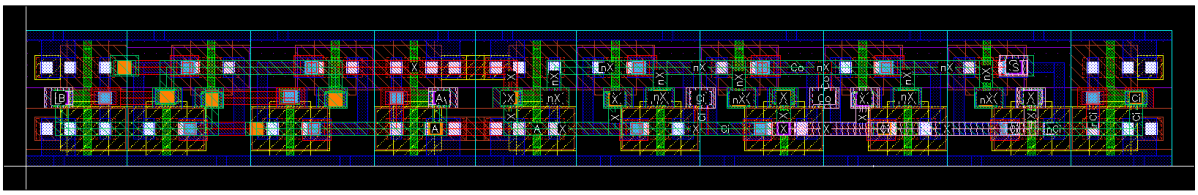
[6] J.-F. Lin, Y.-T. Hwang, M.-H. Sheu, and C.-C. Ho, "A novel high-speed and energy efficient 10-transistor full adder design," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, pp. 1050 – 1059, 06 2007.

[7] R. W.-K. Lam and C.-K. Li, "Optimized transmission gate-based cmos full adders: design and analysis," *International Journal of Electronics*, vol. 88, no. 9, pp. 1001–1013, 2001. [Online]. Available: https://doi.org/10.1080/00207210110058175

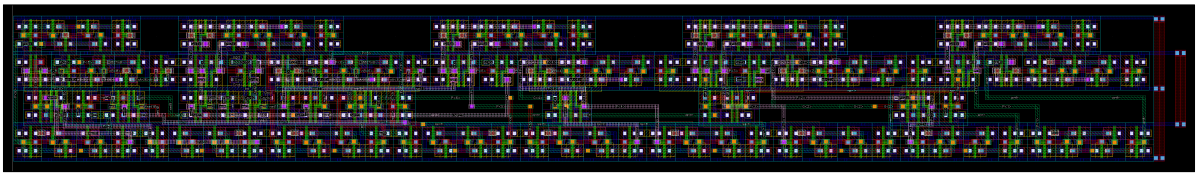
[8] D. Saha, A. Chatterjee, S. Chatterjee, and C. K. Sarkar, "Row-based dual vdd assignment, for a level converter free csa design and its near-threshold operation," *Advances in Electrical Engineering*, vol. 2014, no. 1, p. 814975, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2014/814975

[9] R. P. Rajput and M. S. Swamy, "High speed modified booth encoder multiplier for signed and unsigned numbers," in *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, 2012, pp. 649–654.

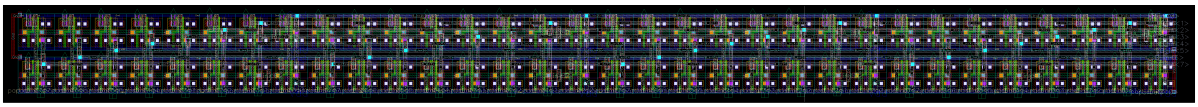
[10] G. Rani and S. Kumar, "Delay analysis of parallel-prefix adders," *International Journal of Science and Research*, vol. 3, no. 6, pp. 2339–2342, 2014. [Online]. Available: https://www.ijsr.net/archive/v3i6/MDIwMTQ3ODc=.pdf



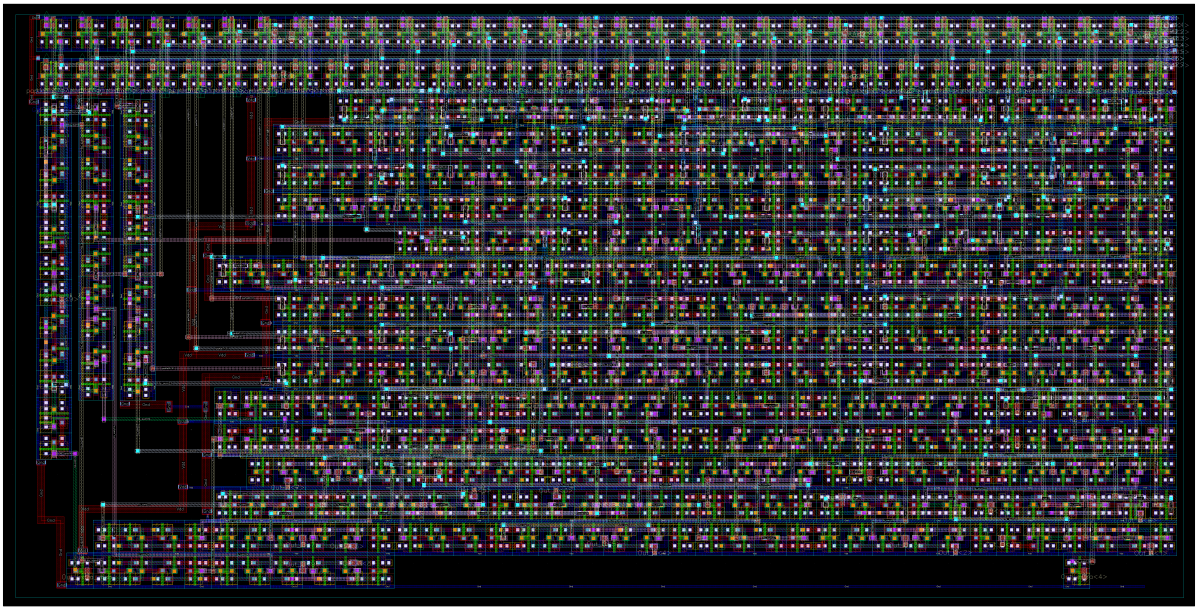
Full-Adder Module



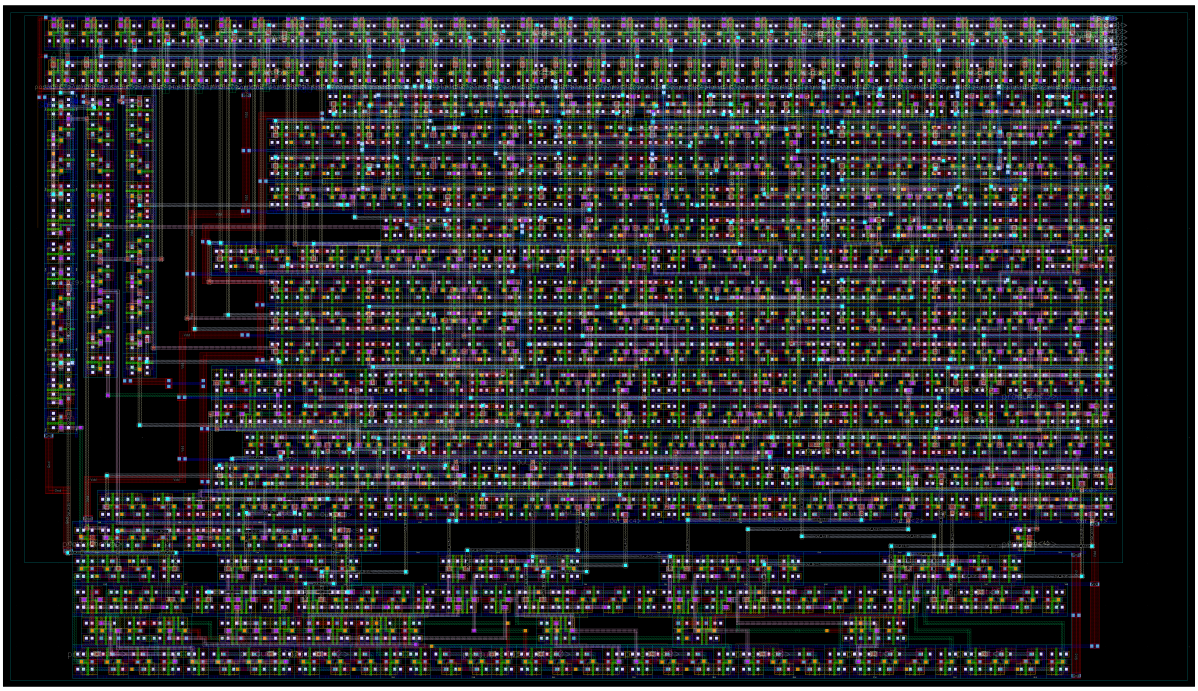
Brent-Kung Adder Module



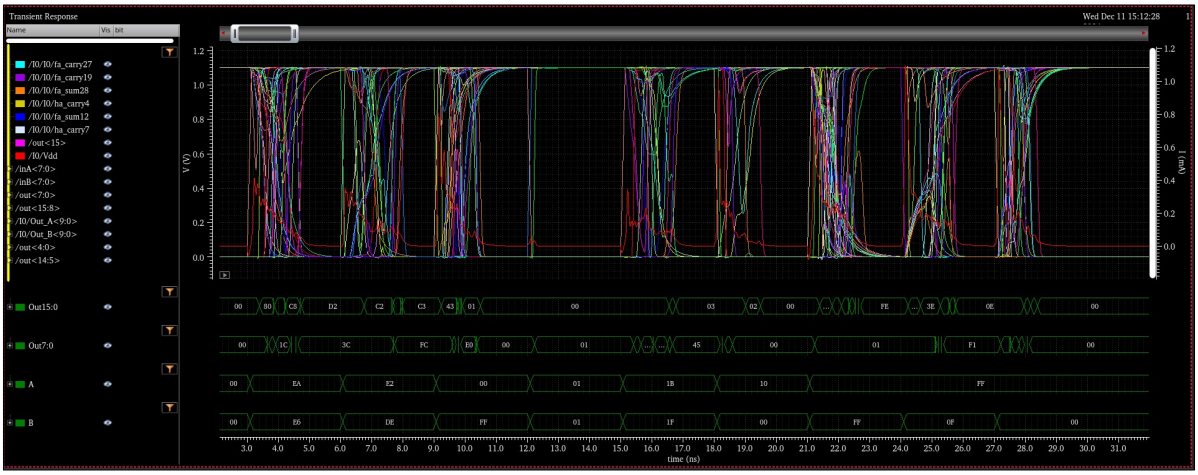
Partial Product Generator



Wallace Tree



Fully-Integrated Wallace Tree Multiplier



Wallace Tree Multiplier Functionality Verification