Labs and Assignments for an Advanced VLSI Course
EECS 499 Final Project Report

Advisor: Prof. Seda Memik

Rebecca Nevin
MS. Electrical and Computer Engineering ‘12
Department of Electrical Engineering and Computer Science
Northwestern University

I. Introduction

The goal of this project was to develop a collection of projects that could be assigned as labs in a 12-week course focusing on advanced VLSI design and analysis. The proposed course would be a supplement to the existing curriculum provided in EECS 391 - VLSI Systems Design and would further students knowledge of modern VLSI design challenges and considerations while improving their CAD, analysis and simulation skills. The added experience and knowledge gained in this course would help Northwestern students be competitive candidates for careers in high-tech chip design at distinguished companies such as Intel, AMD, IBM, Texas Instruments, Apple and a number of other companies including defense contractors.

II. Overview

In the following sections I provide a complete working curriculum for two advanced VLSI projects, each containing two parts. The first asks students to design and compare 8-bit adders using 4 different commonly-used topologies: ripple-carry, cary-look-ahead, carry-select and Kogge-Stone. The second lab asks students to use their best adder to design and analyze an 8x8 multiplier based on Booth encoding. For each assignment I have written prompts that could be given to students that outline the project requirements and I also present the solutions which the results could be graded against. In addition I have included some observations made while I worked on these projects that may help professors and teaching assistants guide their students.

I have also suggested topics for class lectures that would supplement these labs and provide practical VLSI design lessons.

Note: These projects assume that students have access to a working library of basic low-level logic gates, presumably built during EECS 391. Otherwise, assembling and testing these components may prove too time consuming for students to complete the more advanced material.

III. Changes From Proposal

Throughout the course of this project, I made several adjustments from the originally proposed outline. Initially we intended to have students build 32-bit adders. Once I began working, it quickly became clear that this would require a lot of time spent doing repetitive copy-paste work and would not actually develop new skills. In response, I decided 8-bit adders would sufficiently challenge students and serve the purpose of the comparison exercise, without weighing them down with mindless repetition.

A third lab involving a radix-4 divider was also proposed at the initiation of this project. However, after researching several architectures I decided that the complexity of the circuit was not realistic for manual schematic design in a short period of time. Almost all the literature designed this adder using RTL and automated synthesis tools rather than manually in a CAD tool.

Lastly, personal experience in previous courses lead me to introduce the ‘research and planning’ portions of each lab. In the past, many students have struggled to complete working designs due to errors early on, such as having the wrong block diagram or using the wrong testing scheme. To combat this, the professor or TA can provide feedback on Part 1 before the students begin building their design, giving them a better opportunity to spot and correct any major issues. The research portion also introduces students to academic literature, which they likely would not encounter during 300-level courses.

IV. New Outline

Below is a brief summary of the material covered in the labs and presented in sections V and VI below.

- Lab 1: Comparison of 4 Common Adder Topologies
  - Part 1: Research, Plan and Hypothesize
  - Part 2: Design Schematics, Simulate and Analyze
    - A. Ripple-Carry
    - B. Carry-Look-Ahead
    - C. Carry-Select
    - D. Kogge-Stone

- Lab 2: Booth Multiplier
  - Part 1: Research and Plan
  - Part 2: Design Schematic, Simulate and Analyze

V. Lab 1: Comparison of 4 Common Adder Topologies

V.1 Assignment

Adders are one of the most heavily studied units in processor design. They are involved in almost every process throughout the system and often lie in the critical path. For this reason, improving adder performance could result in major savings in the overall system performance. In this lab, you will study four different commonly used adder topologies and compare their benefits and weaknesses.

Part 1: Research, Plan and Hypothesize

Research four common adder topologies: Ripple-Carry, Carry-Look-Ahead, Carry-Select and Kogge-Stone. Write a report that summarizes your findings. It should include the following:

- A block diagram or algorithm for each topology
- A paragraph clearly stating your hypothesis for how the adders will compare in terms of speed and area. (This should be based on data and not merely a guess.)
- Propose a testing plan (input vectors and expected outputs) to prove complete functionality and to determine the worst-case delay of your adders. One plan should work for all four topologies. (Hint: It is possible to complete both goals with one set of input vectors)
- Correct citation of your sources

Part 2: Design Schematics, Simulate and Analyze

Use Cadence to build 8-bit versions of each adder. Use the Analog Design Environment to simulate each adder to verify its functionality and determine the worst-case delay. Present your findings in a report that includes the following:

- Images of each schematic. If you built any sub-components, include these images as well.
- Images of timing plots proving functionality for each adder. Use as many frames as necessary.
- Images of timing plots proving the worst-case delay for each adder. Use the marker tool to show time values.
- In prose, compare and contrast the adder topologies in terms of key design trade-offs such as speed, area, power and complexity. Use data from your designs as necessary to support your conclusions.
- Your images must be clear and properly labeled to receive full credit.

V.2 Solutions

Part 1: Research, Plan and Hypothesize

- Block Diagrams - Please see Appendices A-D for block diagrams of each adder topology.

- Hypothesis - The adders ranked in terms of worst-case delay from fastest to slowest will be Kogge-Stone, Carry-Select, Carry-Look-Ahead and Ripple-Carry. This is due to the number of "stages" or low-level gates in the longest possible path. In ripple-carry, the carry bit is propagated through every stage, so the worst-case system delay is the delay of one full-adder (max 3 gates) multiplied by the number of bits the adder can calculate (in this case 8) so our total is 24. In carry-look-ahead, to maintain a reasonable fanout, we can cascade blocks of 4-bit adders. In each the carry bit propagates through 3 gates, so the longest possible path is only 6 gates. In carry-select, we can exploit the 4-bit carry-look-ahead and then use muxes so that the longest possible path is only 5 gates. In Kogge-Stone, the longest path passes through stage one once (max one gate) and stage two three times (max two gates) for a total of seven gates. However, since these stages are parallel, there is no need to wait for the previous bit to finish calculating, resulting in a faster overall performance.

- Test Vectors - To prove complete functional, you must test every bit, including the carry-in and carry-out. To do this, we can use the following scheme
    - Set A = 10101010, B = 01010101, CIN = 0 >>> The expected output is SUM = 11111111, COUT = 0.
    - Now, leave A and B the same but switch CIN =1 >>> The expected output is SUM = 00000000, COUT = 1.

Part 2: Design Schematics, Simulate and Analyze

- Schematics and Timing Plots - Please see Appendices A-D

- Compare and Contrast - Table 1 reports the worst-case timing results and transistor counts for each 8-bit adder. Transistor count is a quick way to estimate area and provides insights to power consumption and complexity. Naturally, more transistors use more area, consume more dynamic power, release more leakage power, require more wires to connect them, and cost more. However, designs that can be arranged from smaller blocks are much quicker to design. Therefore, choosing the right adder depends significantly on the application. It is up to the designer to set priorities, weigh the tradeoffs and select the design that is most suitable. Although it is not a cascaded design, Table 1 makes it clear why Kogge-Stone is a popular choice. It's impressive performance and reasonable transistor count make it the best option for many highly-integrated systems.

Topology	Worst-Case Delay (pico-seconds)	NMOS Transistors	PMOS Transistors
Ripple-Carry	222	168	168
Carry-Look-Ahead	179	240	240
Carry-Select	164	403	403
Kogge-Stone	102	279	279

Table 1: Design properties for four different 8-bit adder topologies

V.3 Notes and Troublshooting

There were several challenges I encountered throughout the design process that are likely to arise again:

1. The carry-look-ahead schematic diagram that most students will find on google is very difficult to read. There is a significant amount of congestion and there were several errors in mine the first time I assembled it. In addition, students who have not studied advanced VLSI topics may not be familiar with buffers, and therefore may think that buffers in the schematic are actually inverters. Clearly this destroys the logic. Since buffers are highly topical to practical VLSI design, it would be good to cover them in class, and if possible before this project gets largely underway.

2. The Kogge-Stone algorithm is rather challenging to follow. Most students will use the algorithm as outlined on Wikipedia. This is fine as long as they understand exactly what each term means. When I was having difficulty I wrote out several paths by hand to verify that the logic made sense. Students should make sure that carry-in is propagated by using the formula G0 = (A0 AND B0) OR CIN.

VI. Lab 2: Booth-encoded Multiplier

VI.1 Assignment

In this lab you will design a multiplier that uses Booth encoding - a popular algorithm for fast multiplication. You will need to make use of one of the adders designed in Lab 1, so chose wisely and defend your decision. You may need to make several adjustments or build some other subcomponents, so start early and test them before implementing into the complete design.

Part 1: Research and Plan

Research Booth's encoding and decoding algorithms as well as multiplier architectures. Write a brief report that includes the following:

- Block diagrams of a single encoder unit, a single decoder unit, and the overall multiplier architecture.
- Taking the architecture into consideration, choose an adder topology from Lab 1 and explain your decision. (Hint: discuss design tradeoffs).
- Propose a testing plan (input vectors and expected outputs) that will prove the complete functionality of your multiplier as well as determine the worst-case delay.
- Proper citation of your sources.

Part 2: Design Schematic, Simulate and Analyze

Use Cadence to build 8x8 booth-encoded multiplier. Use the Analog Design Environment to simulate your design to verify its functionality and determine the worst-case delay. Present your findings in a report that includes the following:

- Images of the schematic. If you built any sub-components, include these images as well.
- Images of timing plots proving functionality. Use as many frames as necessary.
- Images of timing plots proving the worst-case delay. Use the marker tool to show time
- Your images must be clear and properly labeled to receive full credit.

VI.2 Solutions

Part 1: Research and Plan

- Block diagrams - see Appendix E.

- Adder Choice - I used the the Kogge-Stone adder because it was the fastest. However, for this design, which used a 12-bit adder rather than the 8-bit that was previously built, there was a significant amount of work involved in updating the Kogge-Stone. Since carry-lookahead works well in multiples of four, this would have been much simpler to implement, at the cost of performance.

- Testing Plan - One key feature of Booth's algorithm is that it is signed. For complete functional verification, we must guarantee that negative values produce the expected result. However, we no longer have to worry about the propagation of the carry bit. I chose -13 (A = 11110011) x -95 (B = 10100001) with the expected output being 1235 (P = 0000010011010011). A better testing plan would exercise all of the digits (a negative times a positive would suffice).

Part 2: Design Schematic, Simulate and Analyze

Please see Appendix E for schematics and waveforms.

The worst case delay was 382ps.

VI.3 Notes and Troubleshooting

If students are able to locate the correct documentation, this Lab should not be too challenging. My main hold-up was with sign-extension in Cadence. Unlike Mentor Graphics, Cadence does not allow you to short rips from two different busses. I ended up building sign-extension units that use buffers to transfer from the input bus to the output bus.

VII. Additional Work and Class Topics

Since I have significant experience with Cadence from other independent projects, I am uncertain exactly how long these assignments would take students to complete. In theory, students should be comfortable from completing EECS 391. Should these projects be too little, there are an assortment of other advanced topics I considered.

1. Memory Cells - As the other significant component of any integrated system, it would benefit students to have a better understanding of different memory structures and how to to build them. However, HSPICE and the Analog Design Environment within Cadence make it challenging to properly simulate the bi-directional property of SRAM data inputs/outputs. I built several versions of a 1-bit SRAM cell, but was unable to simulate it in the traditional methods.

2. Practical layouts - When I took 391 (Winter 2012) we did not simulate our layouts, meaning that most of us built circuits that would never actually work. It would be good practice for students to take some basic gates from their library and adjust the layouts until they simulate with parasitics taken into consideration.

3. RTL Synthesis - Students who take EECS 355 will get decent exposure to writing, synthesizing, and simulating VHDL and Verilog code. However, the projects in that class emphasize programming and interfacing with a variety of I/O devices. It would benefit students to gain more experience seeing the complete flow of RTL to schematic to layout using automated tools, since this is how the majority of work gets done in the

industry.

4. Testable designs - Different testing methods are touched on in EECS 391, however it was brief and lacked any practical application (homework or projects). An easy supplement to one of these projects would be to introduce features to simplify testing.

5. Low-power design

6. Low-temperature design

7. Mixed-signal design

Acknowledgements

I would like to thank Matthew Morton and Patrick Schnettler for allowing me to use designs we built in a previous project. Special thanks to Brett Newkirk for his advice on subject matter for these projects. Lastly, thanks to Prof. Seda Memik and the rest of the Computer Engineering department at Northwestern for their continuous support.

References

[1] http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html
[2] http://www.sciencedirect.com/science/article/pii/S0026271410003318
[3] http://web.mit.edu/6.111/www/f2005/tutprobs/arithmetic_answers.html
[4] http://venividiwiki.ee.virginia.edu/mediawiki/index.php/Group_name:_NAND
[5] http://people.ee.duke.edu/~jmorizio/ece261/F08/projects/MULT.pdf

APPENDIX A - Ripple-Carry Adder

A.1 Block Diagram

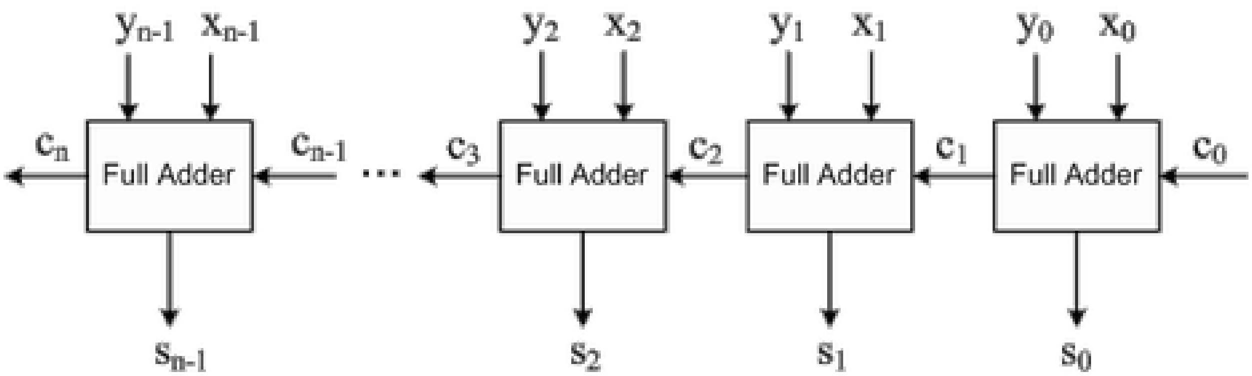

Figure 1: Block diagram of a ripple-carry adder [1].

A.2 Schematics

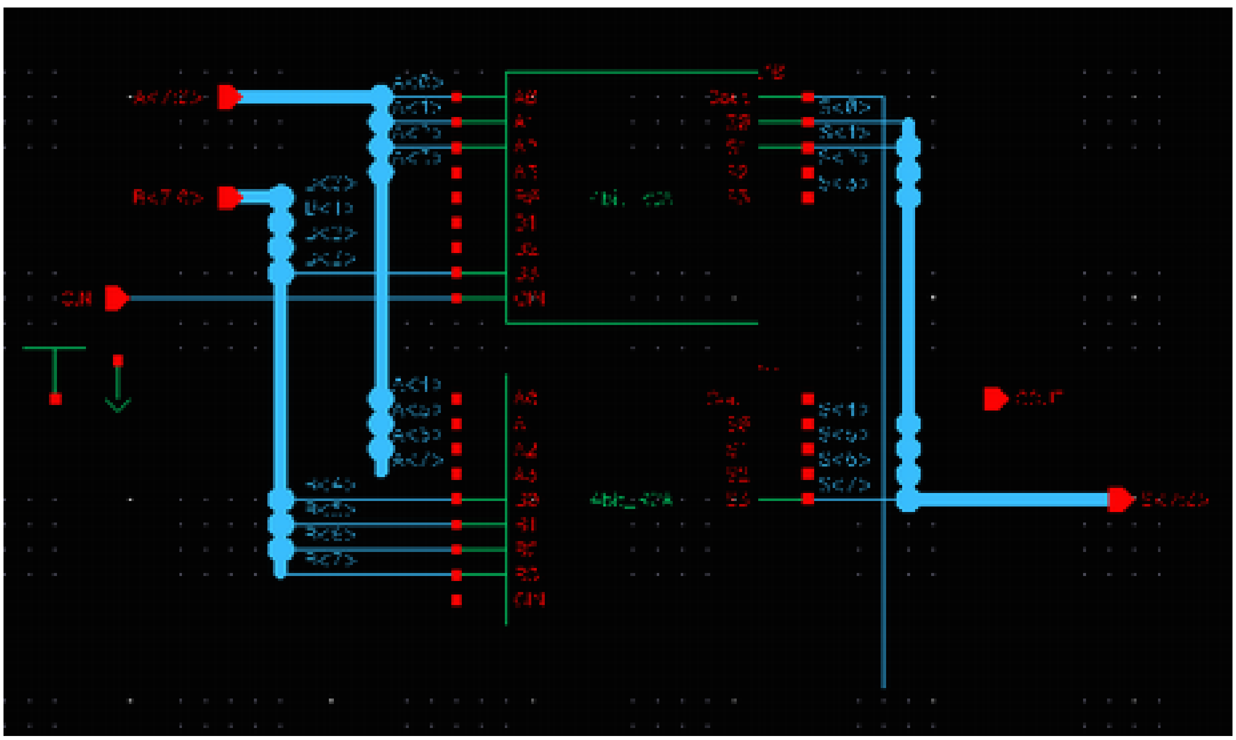

Figure 2: 8-bit ripple-carry using two 4-bit blocks.

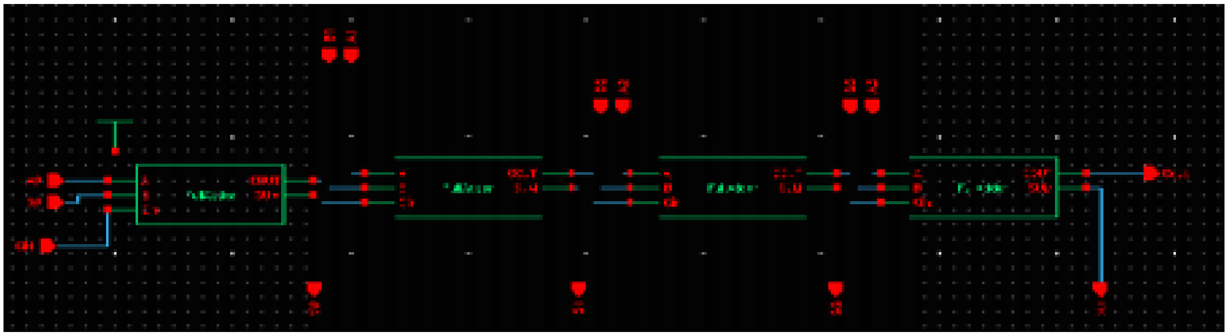

Figure 3: 4-bit ripple-carry block using full-adders.

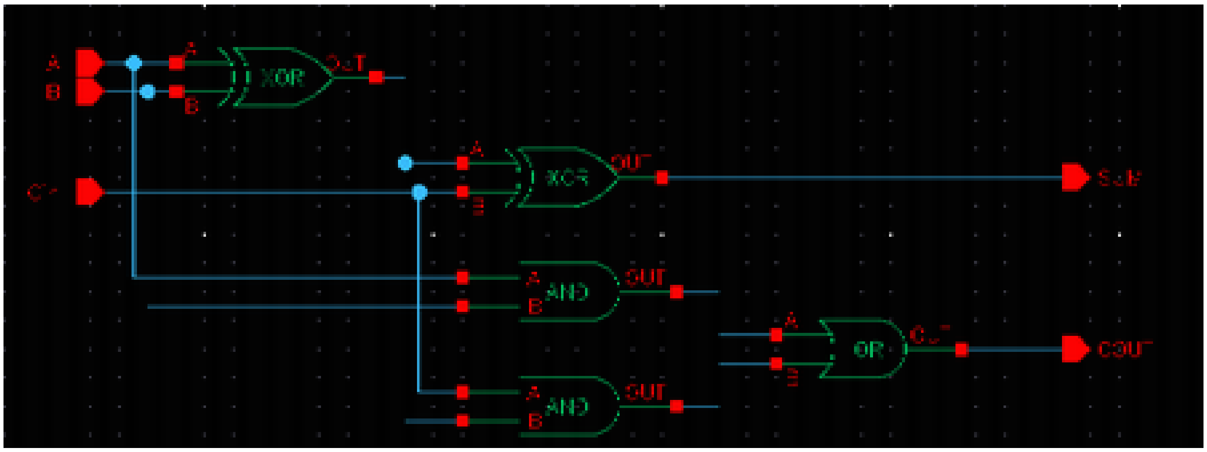

Figure 4: Full-adder .

A.3 Functionality Waveforms

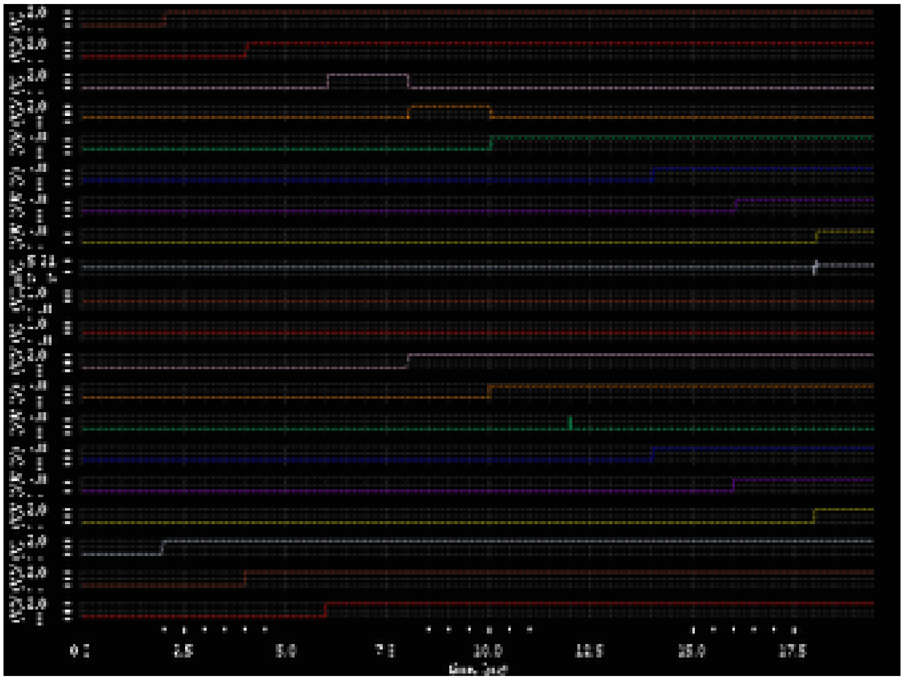

Figure 5: 8-bit ripple-carry adder - proof of functionality.

A.4 Worst-Case Delay Waveforms

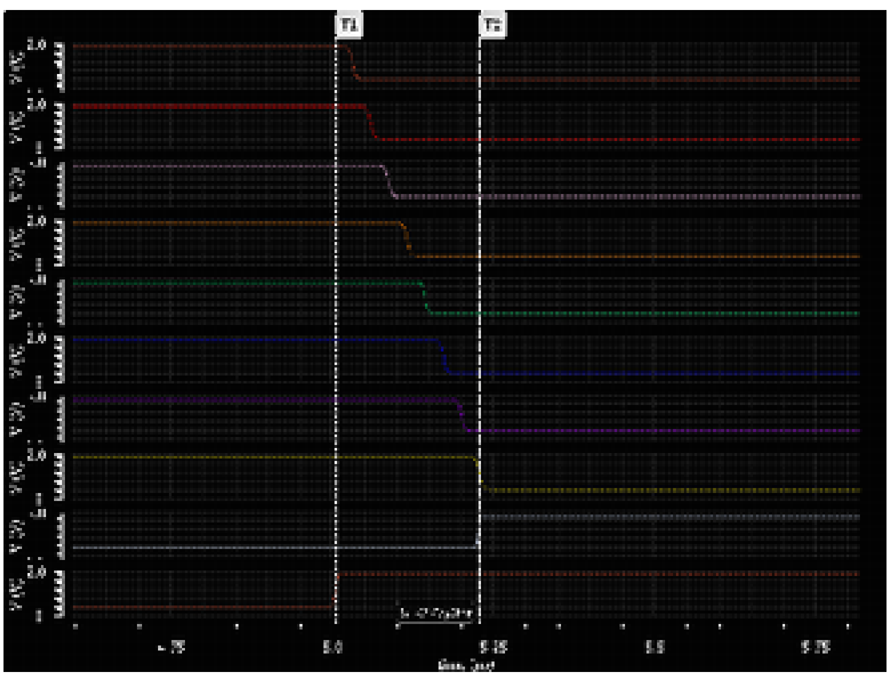

Figure 6: 8-bit ripple-carry adder - worst-case delay = 222ps.

APPENDIX B - Carry-Look-Ahead Adder

B.1 Block Diagram

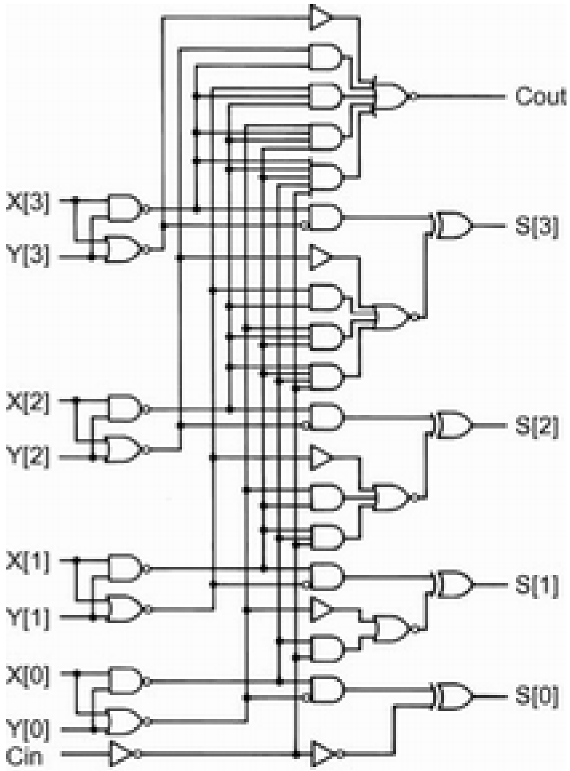

Figure 7: Block diagram of 4-bit carry-look-ahead adder [2].

B.2 Schematics

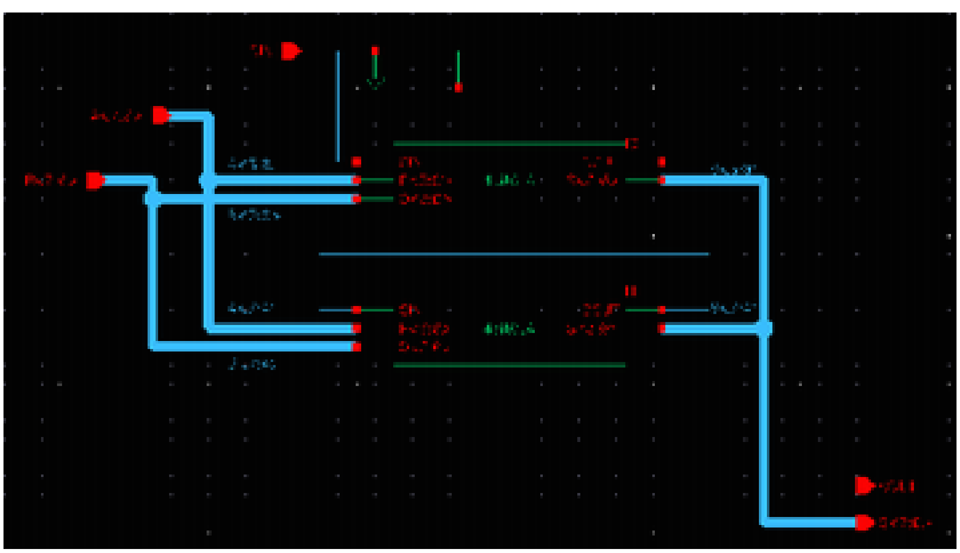

Figure 8: 8-bit carry-look-ahead adder using cascaded 4-bit blocks.

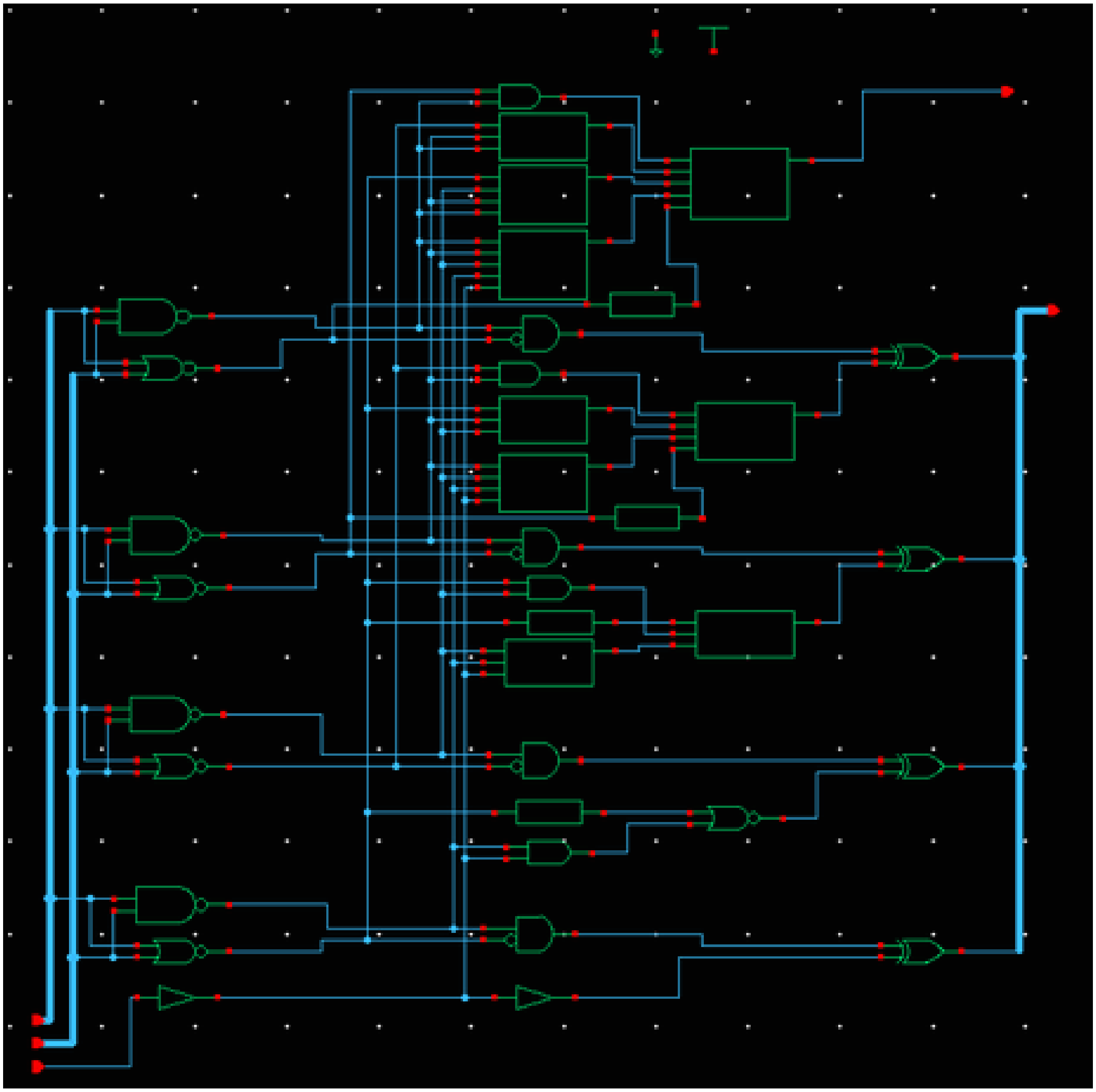

Figure 9: 4-bit carry-look-ahead adder block. This is an exact replica of the block diagram shown in Fig 7.

B.3 Functionality Waveforms

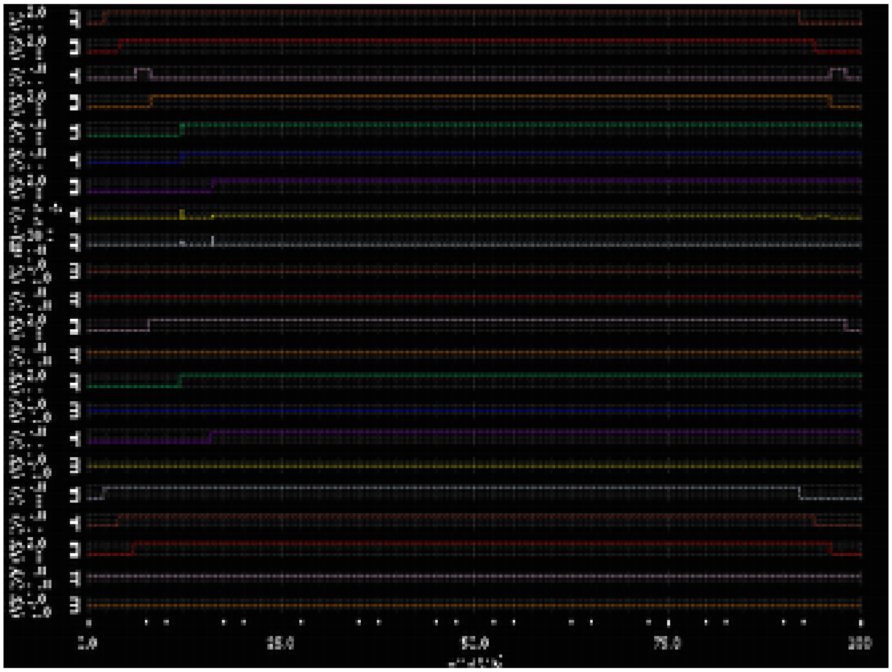

Figure 10: 8-bit carry-look-ahead adder - proof of functionality.

B.4 Worst-Case Delay Waveforms

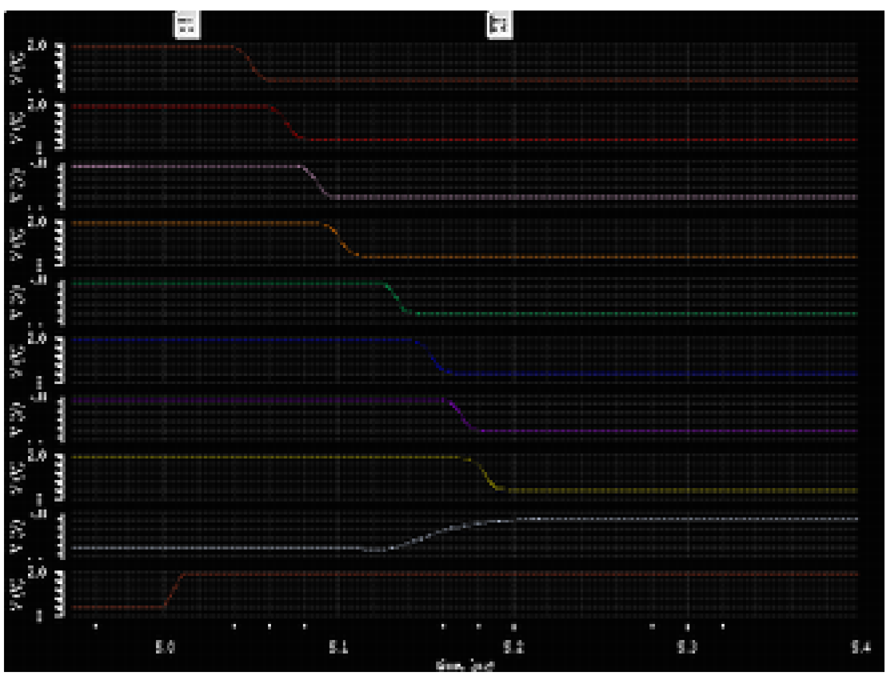

Figure 11: 8-bit carry-look-ahead adder - worst-case delay = 179ps.

APPENDIX C - Carry-Select Adder

C.1 Block Diagram

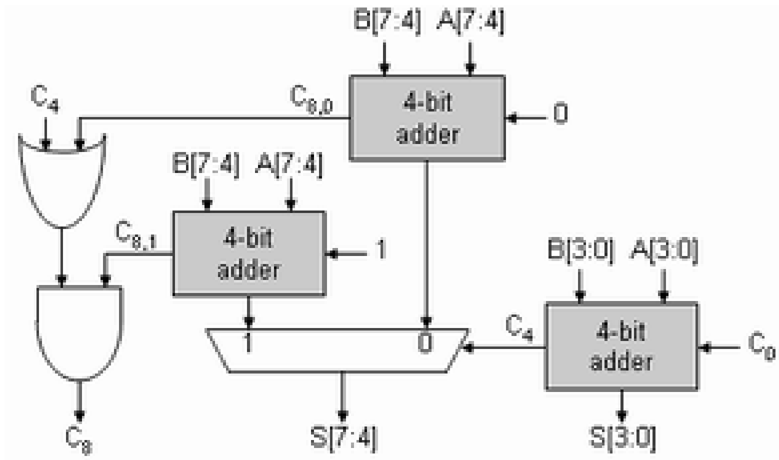

Figure 12: Block diagram of an 8-bit carry-select adder [3].

C.2 Schematics

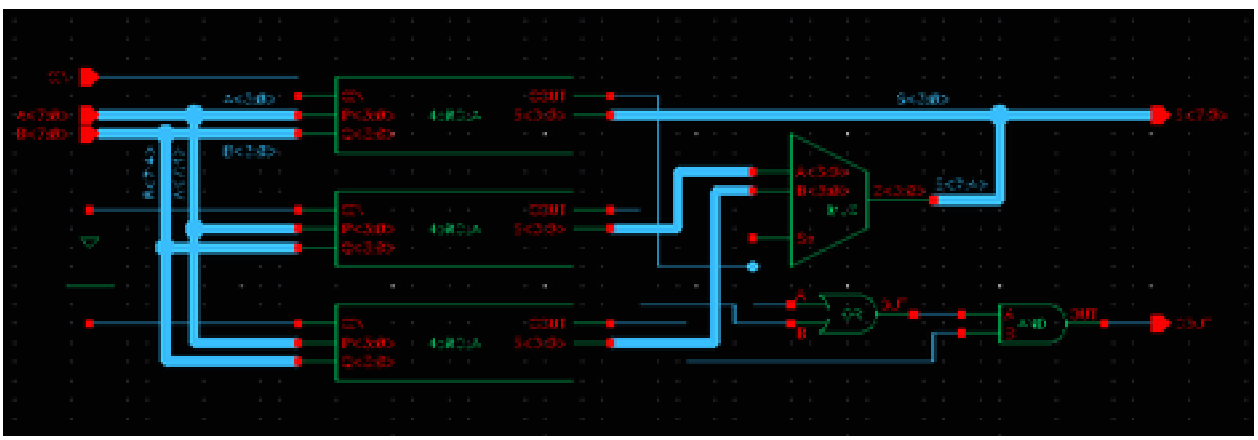

Figure 13: 8-bit carry-select adder using 4-bit CLA (see Appendix B.2).

C.3 Functionality Waveforms

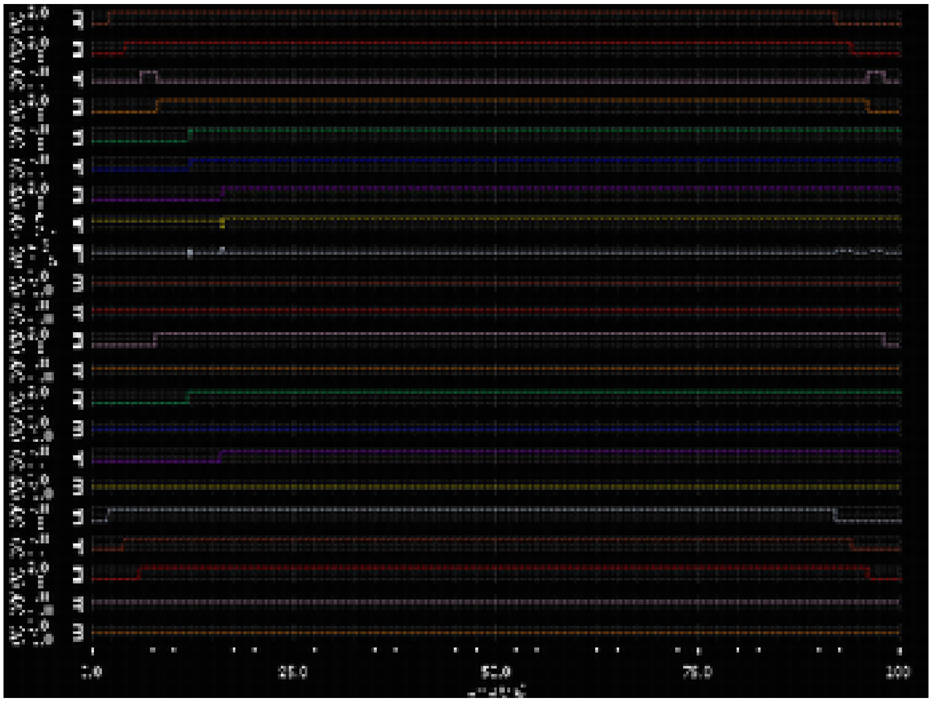

Figure 14: 8-bit carry-select adder - proof of functionality.

C.4 Worst-Case Delay Waveforms

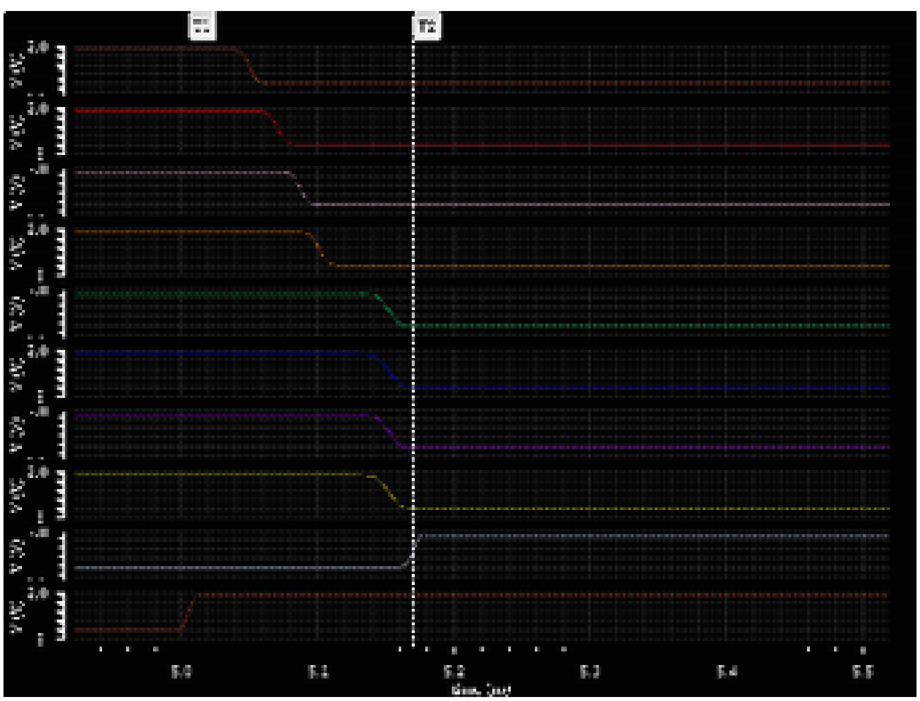

Figure 15: 8-bit carry-select adder - worst-case delay = 164ps.

APPENDIX D - Kogge-Stone Adder

D.1 Block Diagram

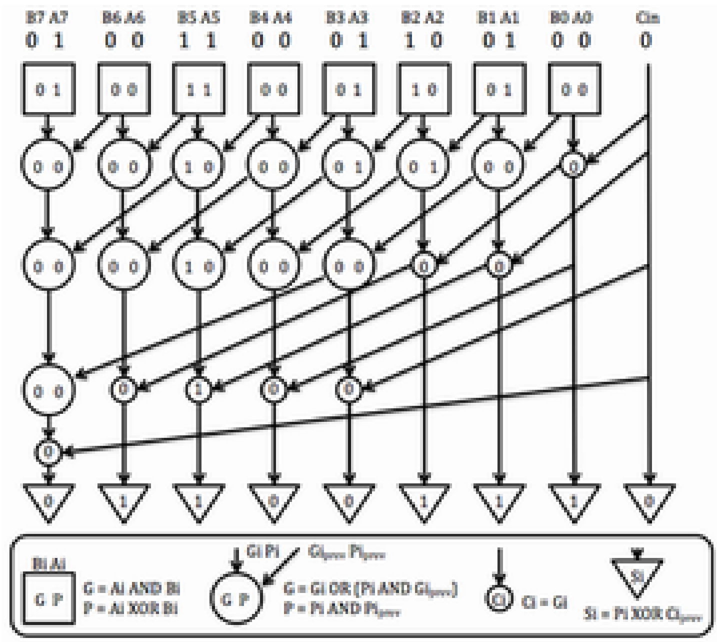

Figure 16: Algorithmic representation of an 8-bit Kogge-Stone adder [4].

D.2 Schematics

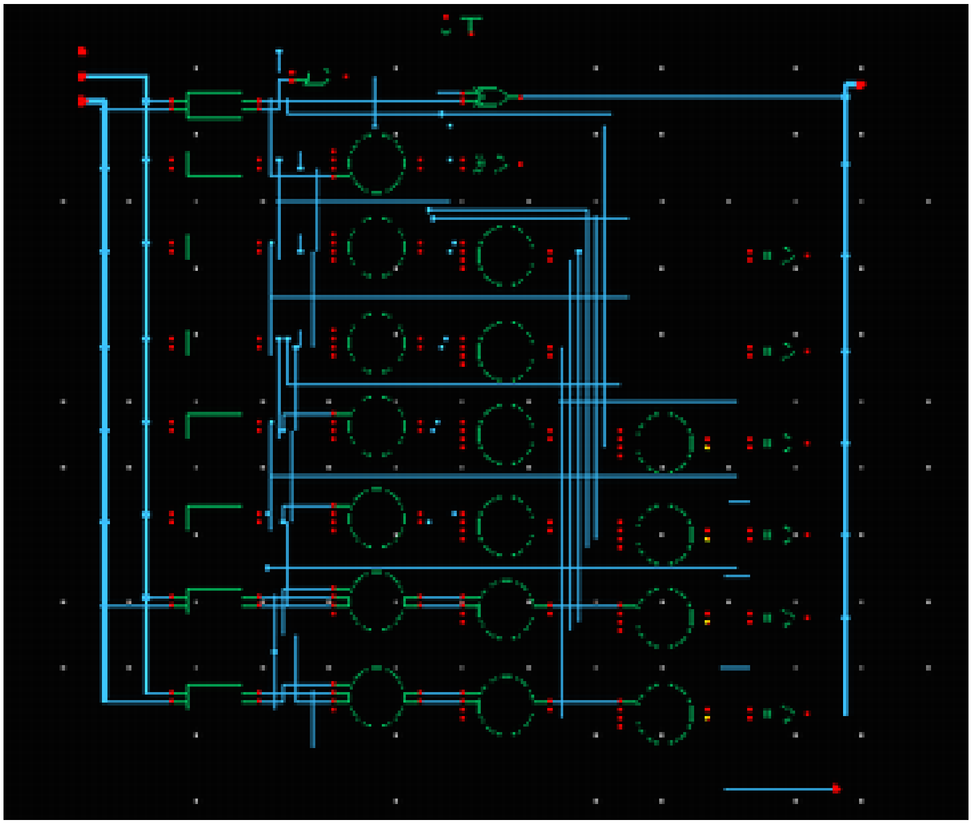

Figure 17: 8-bit Kogge-Stone adder. The rectangular blocks implement the Stage 1 algorithm and the circular blocks implement the Stage 2 algorithm (see Figs 18 and 19).

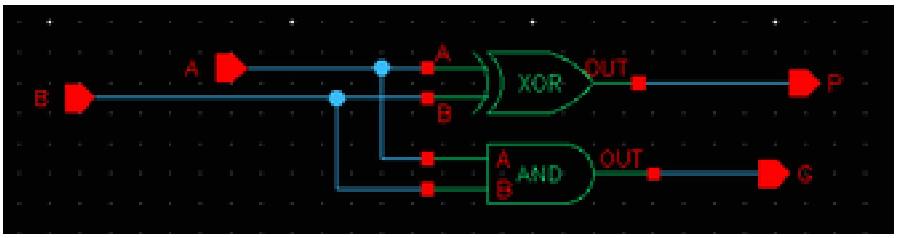

Figure 18: Kogge-Stone Stage 1 block.

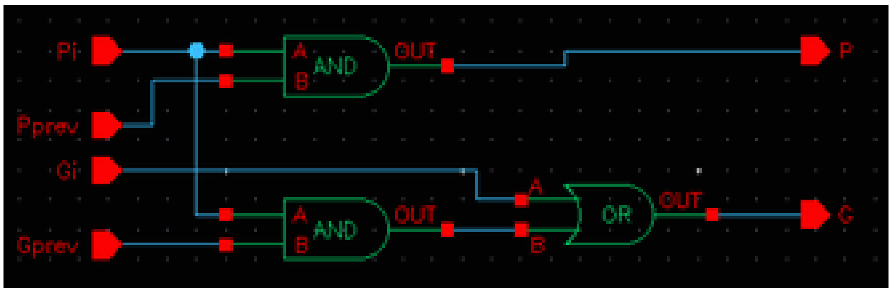

Figure 19: Kogge-Stone Stage 2 block.

D.3 Functionality Waveforms

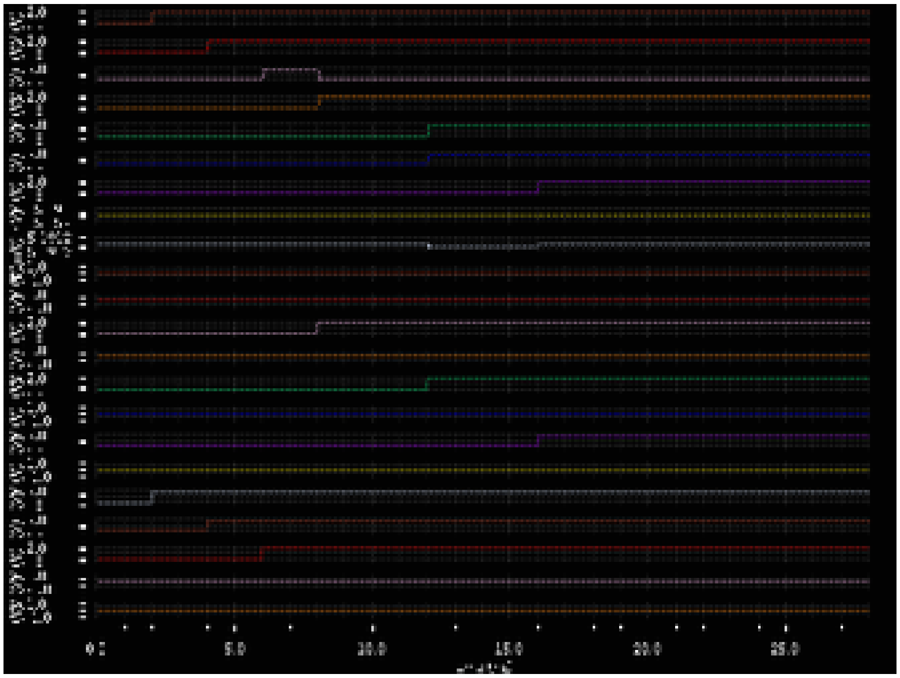

Figure 20: 8-bit Kogge-Stone adder - proof of functionality

D.4 Worst-Case Delay Waveforms

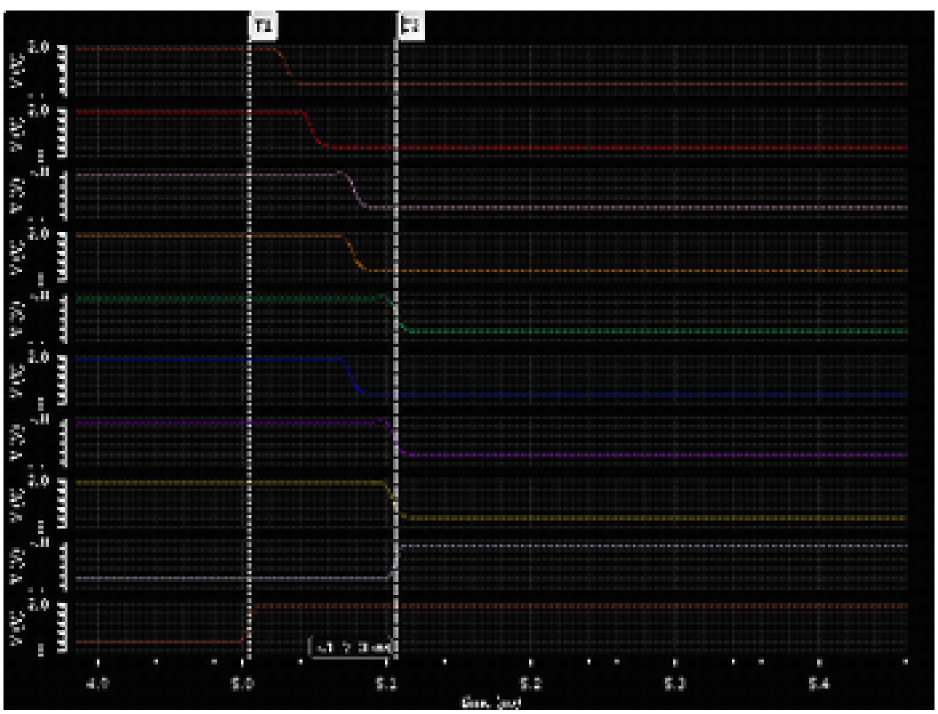

Figure 21: 8-bit Kogge-Stone adder - worst-case delay = 104ps.

APPENDIX E - Booth Encoding Multiplier

E.1 Block Diagram

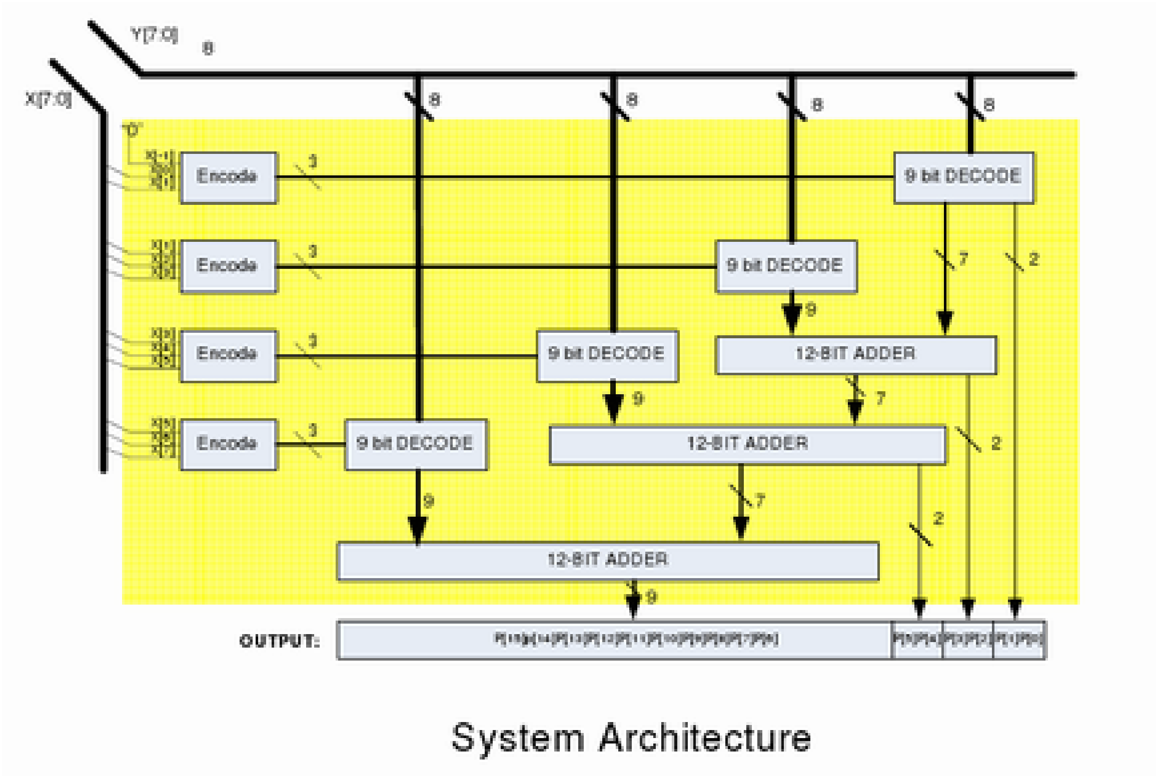

Figure 22: Block diagram for 8x8 booth-encoding multiplier [5].

E2. Schematics

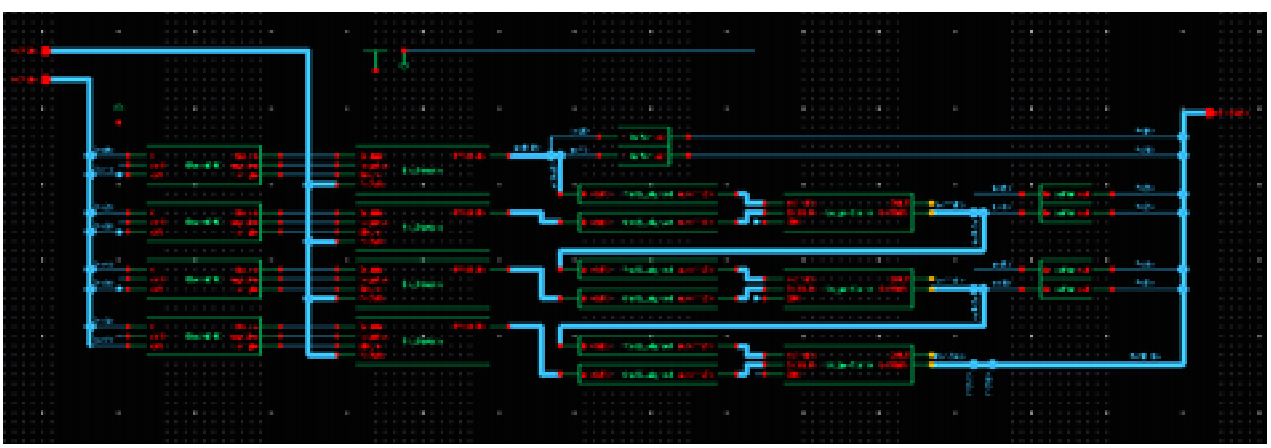

Figure 23: 8x8 Booth encoding multiplier using 12-bit Kogge-Stone adders. Other components in this schematic include four Booth encoders (see Fig. 24), four 9-bit decoders (see Figs. 25 and 26), sign-extension units and buffers.

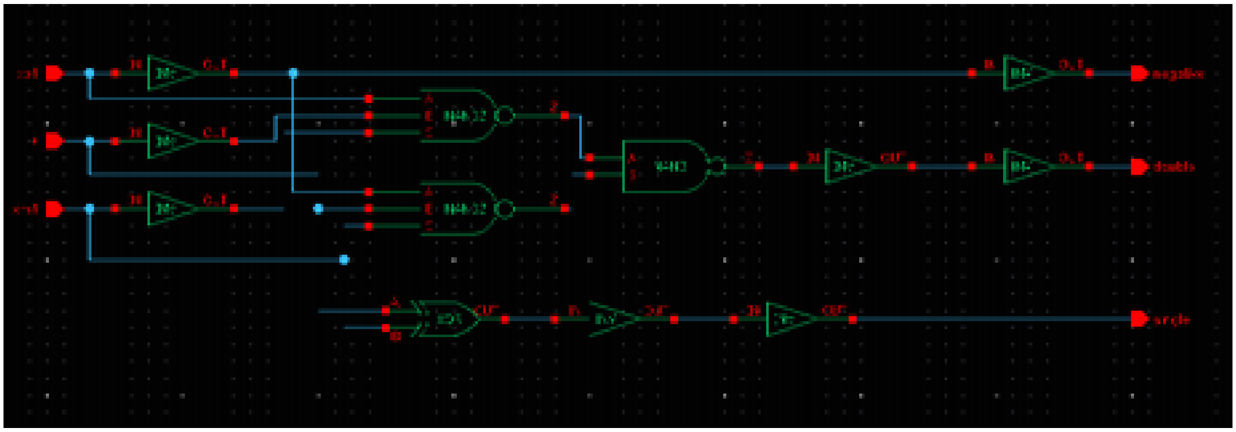

Figure 24: Single Booth encoding unit.

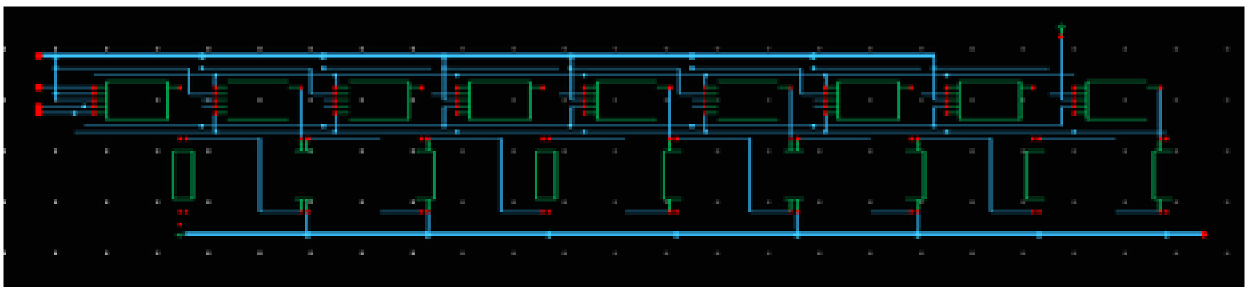

Figure 25: 9-bit Booth decoding unit.

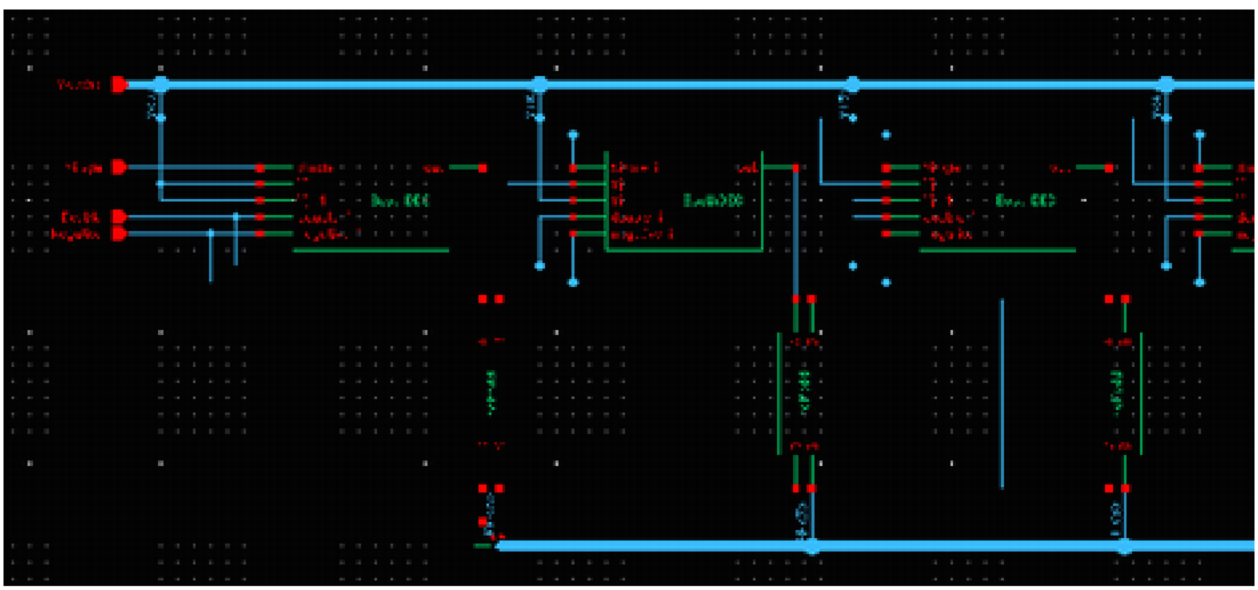

Figure 26: Close-up of a portion of the 9-bit booth decoding unit showing single booth decoder units (see Fig. 27) and half adders (see Fig. 28).

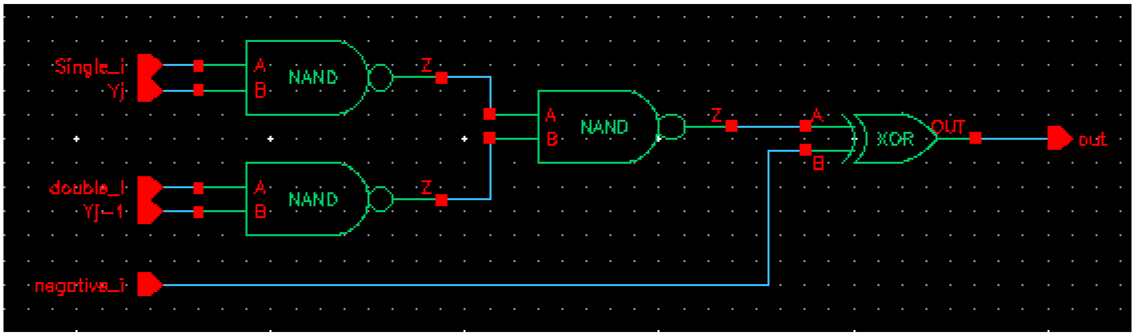

Figure 27: Single booth decoder unit.

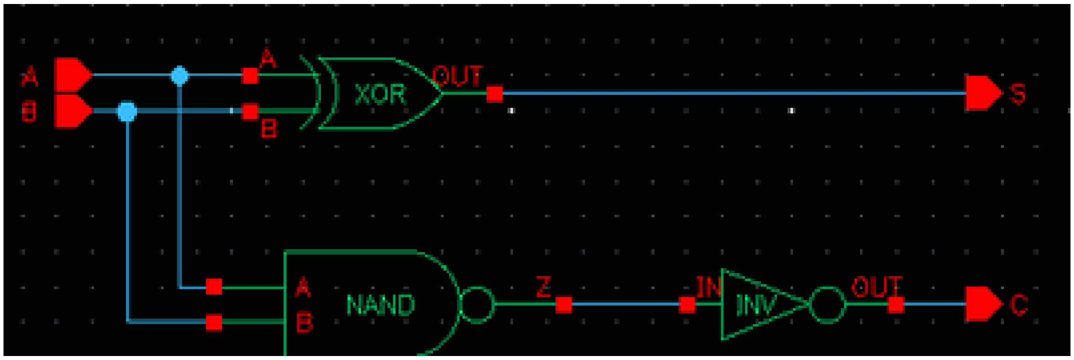

FIgure 28: Half adder.

E3. Functional Waveforms

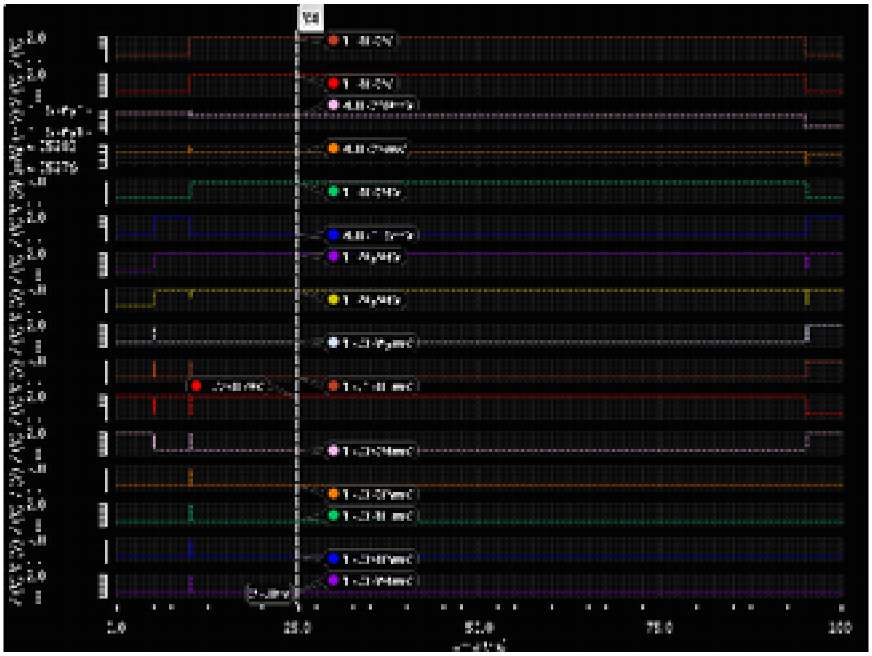

Figure 29: 8x8 Booth multiplier - proof of functionality. (-13 x -95 = 1235)

E4. Worst-case Delay Waveforms

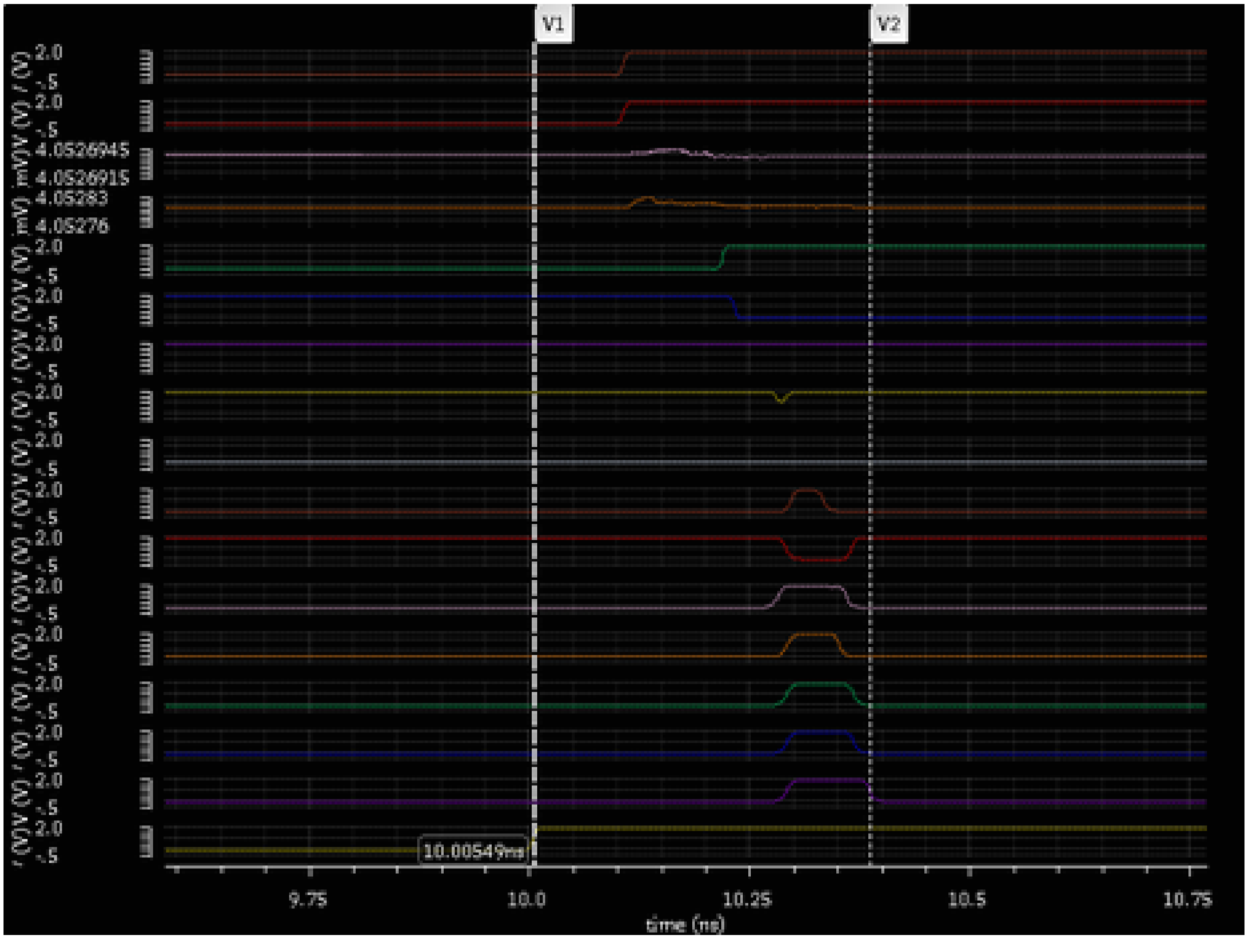

Figure 30: 8x8 Booth multiplier - worst-case delay = 382ps.