UNSIGNED 16-BIT INPUT MODIFIED BOOTH MULTIPLIER

Developed by Edison Chen, Eric Dexheimer and Jason Glass
Instructed by David Zaretsky
Submitted On: 6/6/16

Contents

List of Figures

List of Tables

Executive Summary

This report documents the development of an unsigned 16-bit input multiplier circuit which incorporates modified booth encoding and the Three-Dimensional-Method to improve performance. This design also employs 4:2 compressor trees and a 32-bit carry-ripple adder for the summation of the final output, the product.

This circuit was developed using the Cadence Virtuoso software suite. This team's design flow was to first complete a gate-level schematic of the top-level multiplier, then to complete the associated transistor level layout. The schematic phase was designed from the bottom-up, starting with the smallest unit of operation, an inverter, and culminating in the schematic of the final multiplier. The layout was conducted in the same manner.

The final layout extracted circuit performs multiplicative calculations with a maximal delay of 5.68 ns and with an energy consumption of 12.33 pJ. The final area was 170 um x 170 um, 0.289 mm^2.

Body

Introduction

Our design goal was to implement an unsigned 16-bit input multiplier which utilized modified booth encoding to reduce the number of partial products. Our design also incorporated the Three-Dimensional-Method (TDM) to manage the delay propagation of each partial product. The TDM incorporates 4:2 compressor trees to further improve the performance of the multiplier.

Using Cadence our team built the gate-level schematic of the design with a bottom-up design flow. Starting with inverters all basic logic gates such as AND, OR and XOR were designed. Following this the Booth algorithm hardware and TDM hardware were developed concurrently. Once the top-level multiplier schematic was developed, testing was conducted to confirm successful operation.

Upon completion of testing the development of transistor level layout in Cadence began. This process was identical to the bottom-up approach of schematic design. Starting with inverters all basic logic gates were developed. After this the layouts of the Booth algorithm hardware and TDM hardware were conducted concurrently.

When all the modules of the top-level multiplier had been completed the final routing was conducted to complete the top-level layout of the multiplier.

This multiplier was developed using theories and designs originating from *CMOS VLSI Design* by Weste and Harris [1], and designs from the Tufts Design group D'Angelo and Smith [2].

Broader Considerations

Multiplier circuits are a basic operational unit used in just about any modern hardware. Before the mass production and development of integrated multiplier circuits most multiplicative math was performed using a series of shifts and additions, a slow and inefficient process at the hardware level.

In 1950 Andrew Donald Booth developed his Booth Algorithm. This process allows for the number of partial products in a multiplication to be reduced by increasing factors of two. The algorithm manipulates 2's complement binary representations of numbers to determine the partial products as a function of Y the multiplicand.

The algorithm was useful for Booth at the time, predating modern computing, as the desk calculators in use at the time performed shifts faster than additions. At the lowest level of the algorithm, radix-4, all operations are based off of Y, the multiplicand. Any partial product will be 0, Y, 2Y, -Y or -2Y and thus shifts can be used to implement any multiplication operation at this level.

His algorithm has since been translated into a hardware implementation known as modified booth encoding which uses groupings of 3-bit from the multiplicand to encode the future partial products into a Single, Double and Negative single. This generalization only applies to radix-4, the algorithm used in this project.

Dedicated multiplier circuits in the modern day are heavily used in the signal processing field, a field which relies heavily on speed as a trope. The design and development of increasingly fast dedicated multiplier circuits is integral to increasing the speed and efficiency of decoding increasingly complex signals, such as those NASA decodes from space.

It is difficult to assess the commercial feasibility of this design due to a combination of factors, primarily a lack of publically available multiplier VLSI designs to compare this design too, and the size of this multiplier. The industry standard is 54-bit input; this design is only 16-bit.

Design Constraints and Requirements

This design follows the standard design rules that govern the realization of a chip from layout form to taping and printing at a chip fabrication center. These rules define how close layers of metal may be to each other, their width, length and other various factors.

There are also general design conventions which need to be followed to give a chip any chance at being commercially successful. One such rule would be the strategy of implementing the power rails vdd and vss using the lowest level of metal, metal1. Figure 1 presents a sample of these rules [1].

MOSIS SUBM design rules (3 metal, 1 poly with stacked vias & alternate contact rules)			
Layer	**Rule**	**Description**	**Rule (λ)**
N-well	1.1	Width	12
	1.2	Spacing to well at different potential	18
	1.3	Spacing to well at same potential	6
Active (diffusion)	2.1	Width	3
	2.2	Spacing to active	3
	2.3	Source/drain surround by well	6
	2.4	Substrate/well contact surround by well	3
	2.5	Spacing to active of opposite type	4
Poly	3.1	Width	2
	3.2	Spacing to poly over field oxide	3
	3.2a	Spacing to poly over active	3
	3.3	Gate extension beyond active	2
	3.4	Active extension beyond poly	3
	3.5	Spacing of poly to active	1
Select (n or p)	4.1	Spacing from substrate/well contact to gate	3
	4.2	Overlap of active	2
	4.3	Overlap of substrate/well contact	1
	4.4	Spacing to select	2
Contact (to poly or active)	5.1, 6.1	Width (exact)	2 × 2
	5.2b, 6.2b	Overlap by poly or active	1
	5.3, 6.3	Spacing to contact	3
	5.4, 6.4	Spacing to gate	2
	5.5b	Spacing of poly contact to other poly	5
	5.7b, 6.7b	Spacing to active/poly for multiple poly/active contacts	3
	6.8b	Spacing of active contact to poly contact	4
Metal1, Metal2	7.1, 9.1	Width	3
	7.2, 9.2	Spacing to same layer of metal	3
	7.3, 8.3, 9.3	Overlap of contact or via	1
	7.4, 9.4	Spacing to metal for lines wider than 10 λ	6
Via1, Via2	8.1, 14.1	Width (exact)	2 × 2
	8.2, 14.2	Spacing to via on same layer	3
Metal3	15.1	Width	5
	15.2	Spacing to metal3	3
	15.3	Overlap of via2	2
	15.4	Spacing to metal for lines wider than 10 λ	6
Overglass Cut	10.1	Width of bond pad opening	60 μm
	10.2	Width of probe pad opening	20 μm
	10.3	Metal3 overlap of overglass cut	6 μm
	10.4	Spacing of pad metal to unrelated metal	30 μm
	10.5	Spacing of pad metal to active or poly	15 μm

Figure 1: Design Rules [1]

Design Description

This design can be broken down into 3 modules, Booth hardware, TDM hardware and the final adder. This section will give an overview of the schematic and layout of these modules. For schematics and layouts of smaller submodules such as logic gates see Appendices A and B respectively.

Booth Hardware

The Booth hardware require the encoder and decoder to be built in 1-bit, 17-bit and 9 product versions. The 1-bit versions of the encoder and decoder, seen in figures 2 and 3 respectively, are the basic units of the booth hardware. The 3-bit encoder takes a grouping of 3 bits and outputs the Single, Double and Negative signals. The 1-bit decoder takes these signals and outputs the appropriate bit 0 or 1. Note that in many booth hardware setups the decoder has an embedded 1 bit Left-Shift module. This team however referenced a design from Tufts [2], which uses a sequence of decoders and half adders to shift Y when necessary, this is seen in Appendices A and B as the 17-bit Decoder.

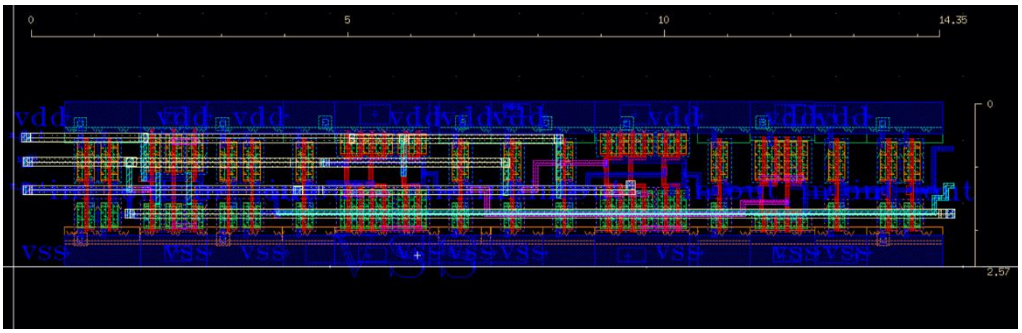


Figure 2: 3 Bit Encoder

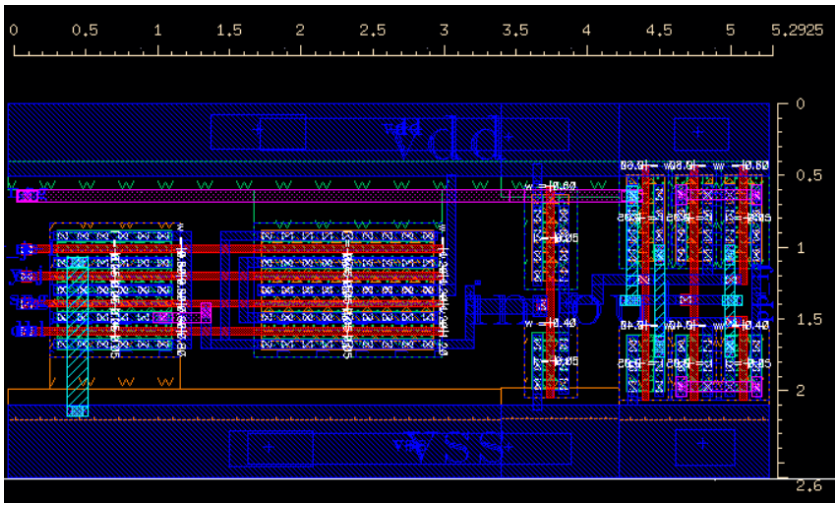


Figure 3: 1 Bit Decoder

The next level of Booth hardware are the 9 product decoder and encoder seen in figures 4 and 5 respectively. The 9 product encoder turns the 16-bit input into 9 groupings of Single, Double and Negative which are outputted to the 9 product decoder. The 9 product decoder takes these signals and outputs the 9 partial products which are summed in the column additions.

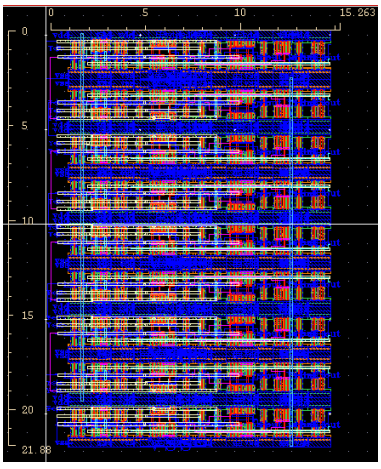


Figure 4: 9 Product Encoder

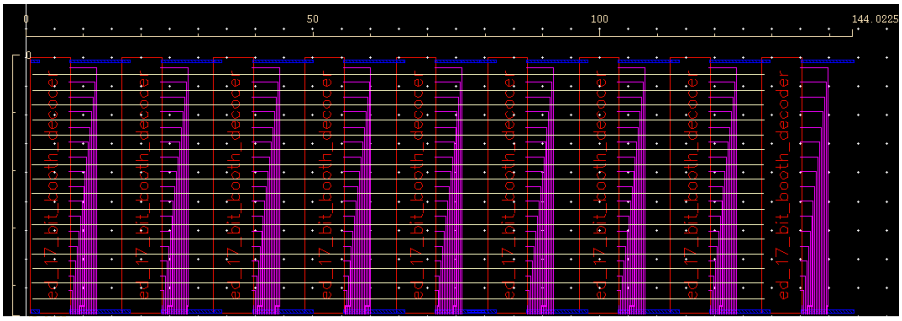


Figure 5: 9 Product Decoder

Three-Dimensional Method

The following figure shows the final layout of the column additions which define the TDM. For information on how these columns work and their purpose in improving this design see the appropriate section in Design Optimization.

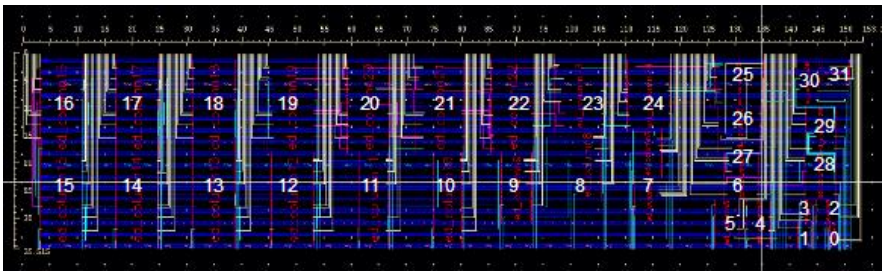


Figure 6: Column Addition

The Top Level layout and schematic can be seen in figures 7 and 8 respectively.

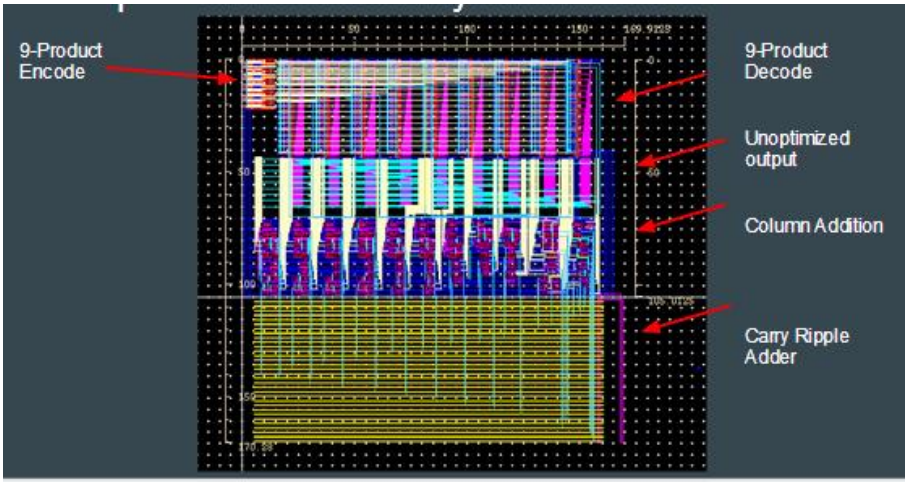


Figure 7: 16-Bit Multiplier

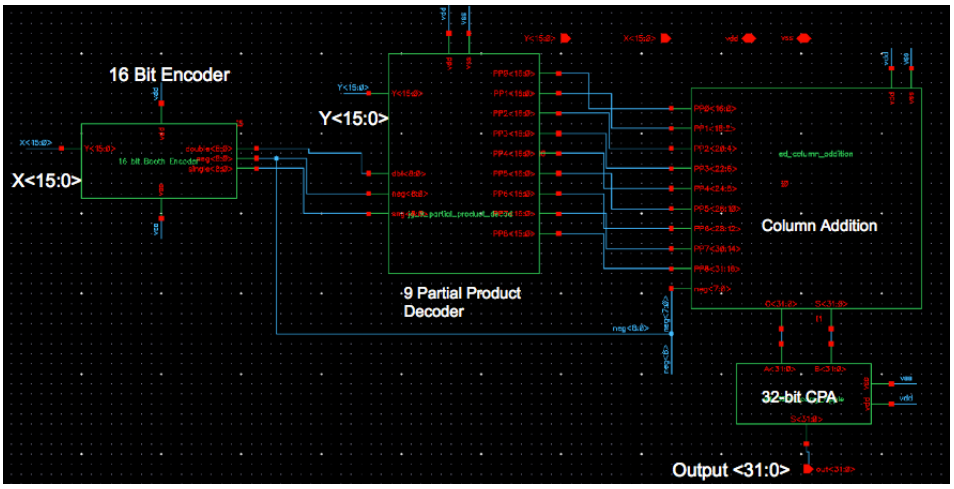


Figure 8: 16 Bit Multiplier

Design Optimization

This design utilizes two main levels of optimization as previously detailed, the Booth algorithm and the Three-Dimensional-Method. This section will explain in greater detail how these optimizations work and how they were realized in hardware.

Booth Algorithm

The Booth algorithm in hardware works by encoding groupings of bits in the multiplicand, Y, and then employing decoders to interpret each set of bits as being either 0, Y, 2Y, -Y or -2Y in the respective partial product. Figure 9 shows the encoding table for this process.

Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	SINGLE_i	DOUBLE_i	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0\ (=0)$	0	0	1

Figure 9: Booth Encoding Table [1]

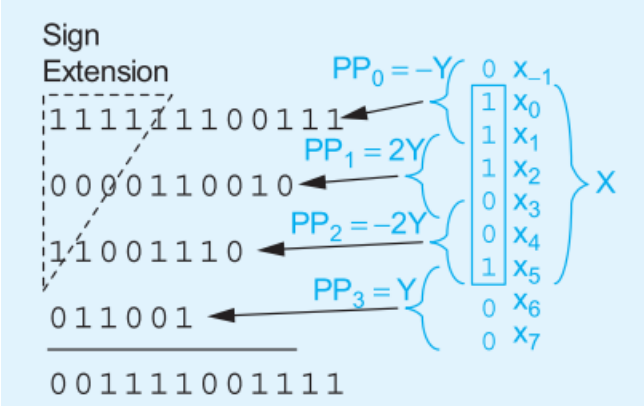


Figure 10: Booth Encoding Example [1]

Figure 10 shows an example of the Booth algorithm being used to group bits together to acquire the appropriate multiple of Y. Note the sign extension. Each level of the Booth Algorithm beginning with "radix-4" reduces the input by a factor of $\log_2(\text{radix})$, so radix-4 reduces the number of partial products by a factor of 2.

Three-Dimensional Method

The partial product outputs of the decoders are arranged into the addition shown below:

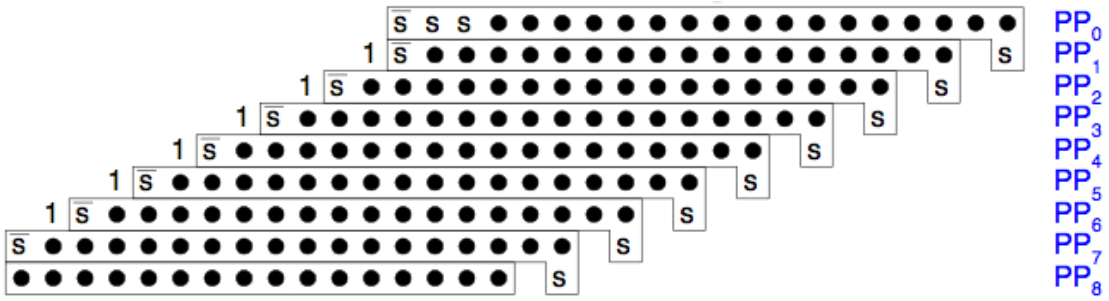


Figure 11: Sign Extension [1]

Since the partial products are signed, the sign bits from the encoder were also needed. The form above reduces the sign extensions so that each sign bit at most four times, see Appendix C more information on Sign Extension.

The column additions were completed using 4:2 compressor trees, carry-save adders, half-adders, and inverters. The sum and final carry of each column are sent to the 32-bit adder as the final step. Each column may also produce multiple carries, with the rest propagating to the next column as additional inputs. By waiting to add the sign bit to the LSB of each partial product in the column additions, a chain of ripple-carry adders can be avoided in the decoder when converting to the two's complement form for -Y and -2Y. This parallel computation helps reduce the worst case delay. An example of a column addition is shown with Column 18, which contains 8 partial product inputs, the sign bit from partial product #0, and six carry bits from Column 17. Three compressors and one CSA were used in this case.

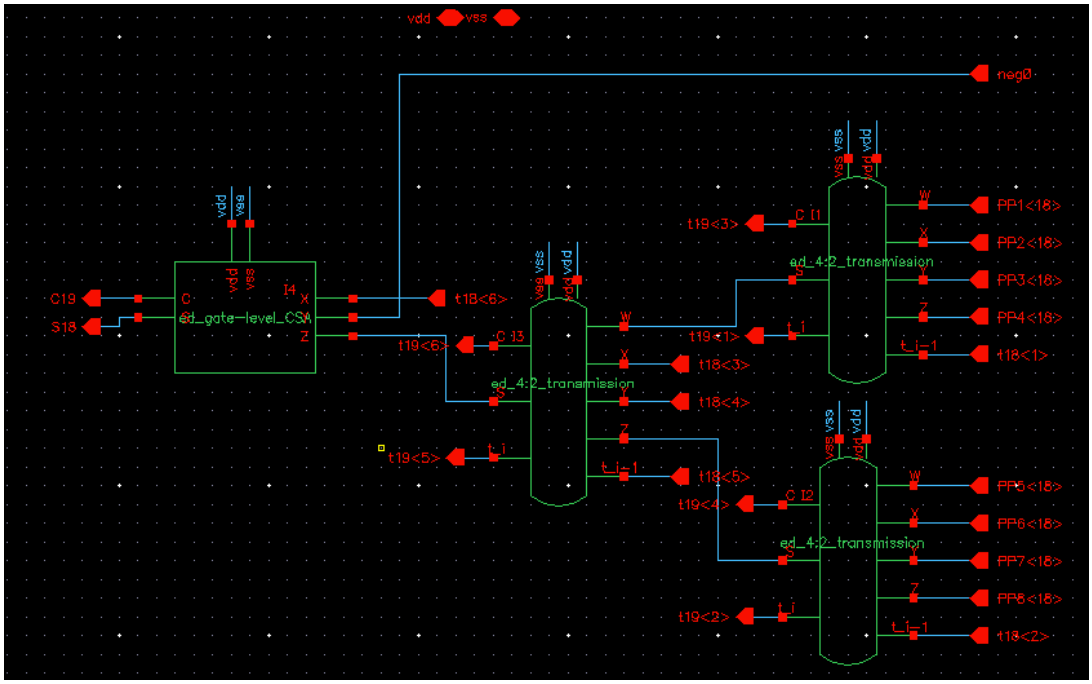


Figure 12: Column 18

The columns are designed to rank the delays of the carry bits between columns so that a third dimension of time could be accounted for instead of just rows and columns, known as the Three Dimensional Method. Using this method, the critical delay can be reduced as slow carries are routed to fast inputs in the CSA and 4:2 compressor architectures. An example of compressor tree levels taking in carry-in bits arriving late is shown below. The numbers shown represent relative arrival times.

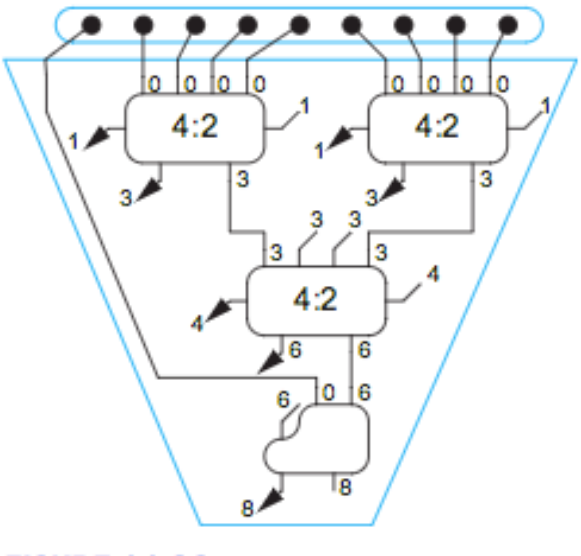


Figure 13: Propagation Delay Chart [1]

Lastly, the sum and carry bits act as inputs to the final 32-bit adder. For the simulations in this project, a 32-bit ripple carry adder was used, which prioritizes power over delay. However, the final adder could be easily replaced to accommodate other design preferences.

The column addition with compressor trees increased the complexity of the design, complicating both the schematic and layout, but decreased delay with the use of the Three Dimensional Method to coordinate the delays of the carry bits between columns with the logic levels of the compressors.

Testing and Simulation

Testing was conducted at both the schematic and the layout level. Schematic level testing verified the functionality of the design while giving rough estimates of the power consumption and critical delay.

Layout testing gives the real delay and consumption that would exist in a physically manufactured chip.

The following table shows the final delays and power consumptions of our design.

Table 1: Test Results

	Schematic	Initial Layout	Improved Layout
Critical Delay	1.59 ns	7.30 ns	5.68 ns
Maximum Energy (10 ns period)	4.63 pJ	13.71 pJ	12.33 pJ
Maximum Power	4.63 mW	13.71 mW	12.33 mW

Implementation

N/A – VLSI group

Conclusions

Future work includes fixing the vdd and vss rails, improving the layout density, designing a final adder for minimizing both power and delay, and optimizing the 3-Dimensional connections.

Improving the vdd and vss rails will improve the layout critical delay quickly and efficiently by ensuring all can be connected with metal1. However, to fully improve these paths, the column additions must all be altered so that no metal2 to metal3 vias are placed within the vdd and vss rails. After this has been completed, metal10 and metal9 could be used to connect all power rails.

Due to time constraints, the final layout is somewhat inefficient. There are large spaces exclusively for routing so that debugging was not as challenging. However, not all metal layers were used, so higher

level metals could have been run over the column additions and decoder. This improvement would take the longest amount of time to fully condense the layout.

In this design, a 32-bit ripple carry adder is used, which has minimal power and area, but a very slow delay compared to all other adders. Originally, the multiplier would have been integrated into the ALU group's Kogge-Stone adder for the final addition, which would have very large power consumption, but an extremely fast delay. Although this would be very helpful for a fast multiplier, the fact that the column additions are completed at different times due to the number of logic levels increasing and then decreasing from column 0 to column 31. Thus, the arrival of inputs to the final adder could be observed so that the full parallel nature of the Kogge-Stone is not needed. One example of such an adder is shown below.

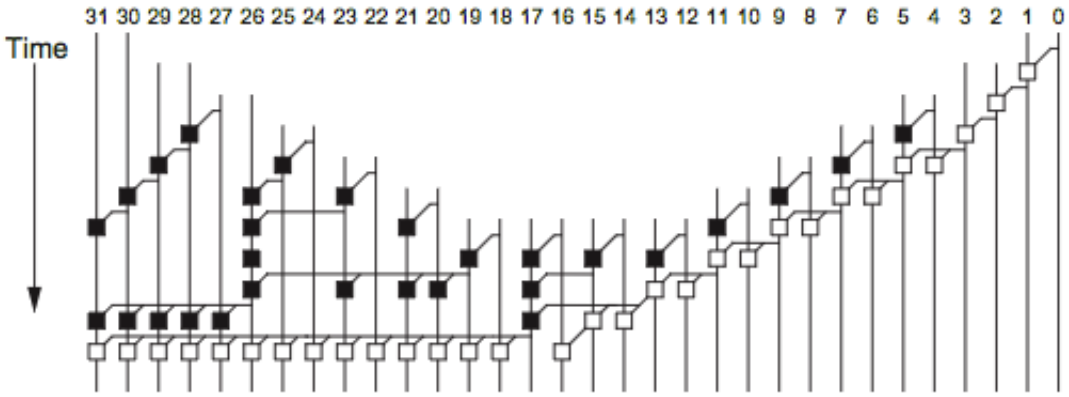


Figure 14: CPA Prefix Network [1]

The delay will not be much different from a Kogge-Stone adder, and the power is greatly reduced. The middle inputs, which will take the longest to arrive from the column additions, only need about 4-levels of cells, which is the same as from a 32-bit Kogge-Stone adder. However, the outer levels have a larger number cell levels, but the final arrival time of the multiplier outputs will be similar.

Lastly, the 3-Dimensional connections could be further tested to ensure that each carry bit is ranked properly. The levels of CSAs and 4:2 compressors could also be improved to ensure that all partial products are inputs to the greatest levels of XOR gates.

References

[1] Weste and Harris, *CMOS VLSI DESIGN: A Circuits and Systems Perspective*. 4th ed. Pearson, 2011.

[2] D'Angelo and Smith, *Modified Booth Multiplier*. Tufts, 2011.

Appendices

Appendix A of schematics

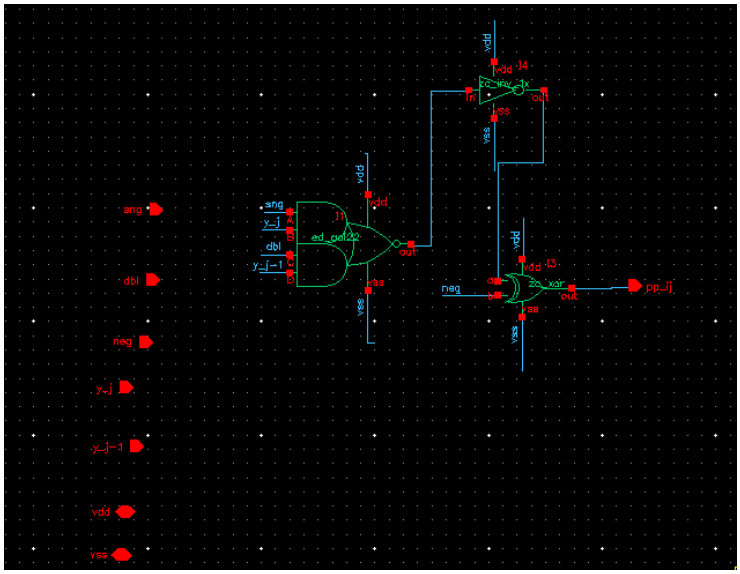


Figure 15: 1 Bit Decoder

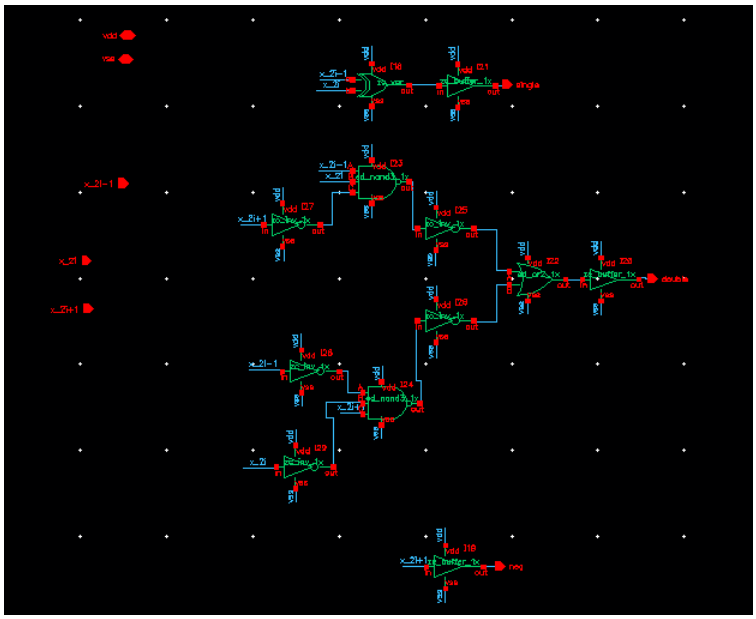


Figure 16: 1 Bit Encoder

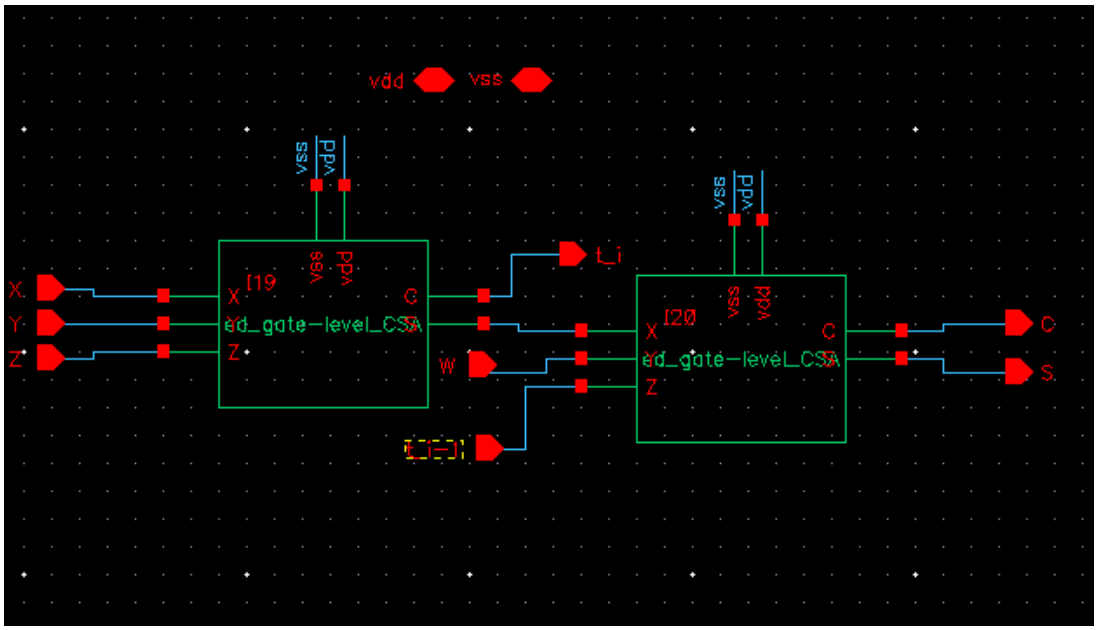


Figure 17: 4:2 Compressor

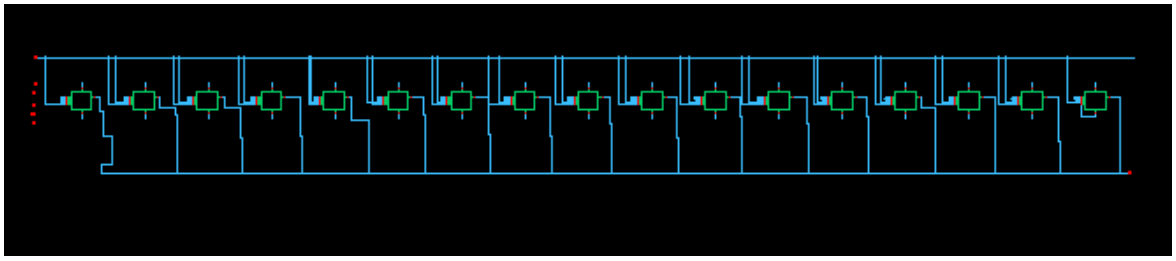


Figure 18: 17 Bit Decoder

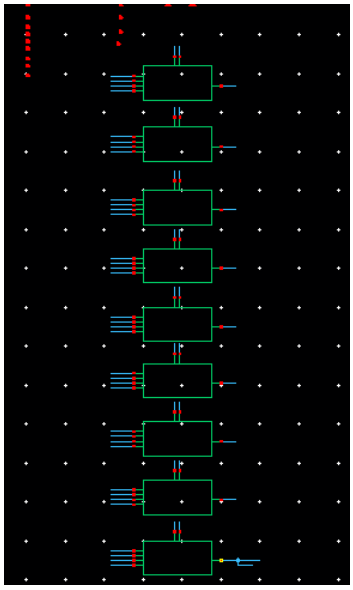


Figure 19: 9 Product Decoder

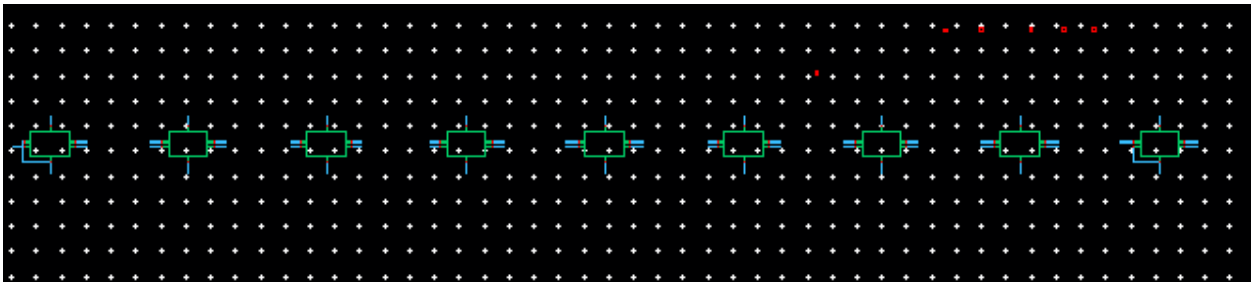


Figure 20: 9 Product Encoder

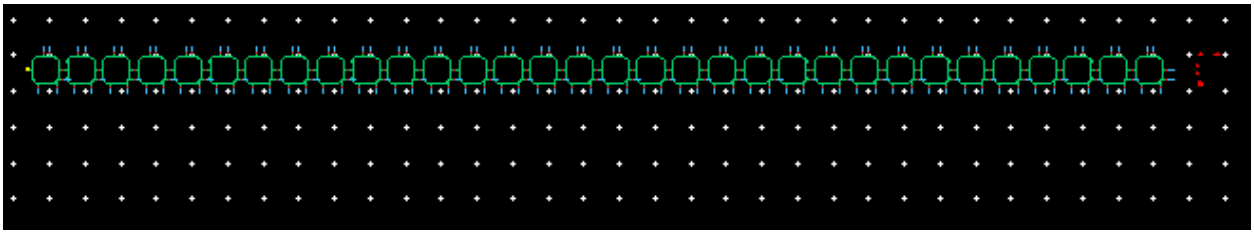


Figure 21: 32-Bit Carry Ripple Adder

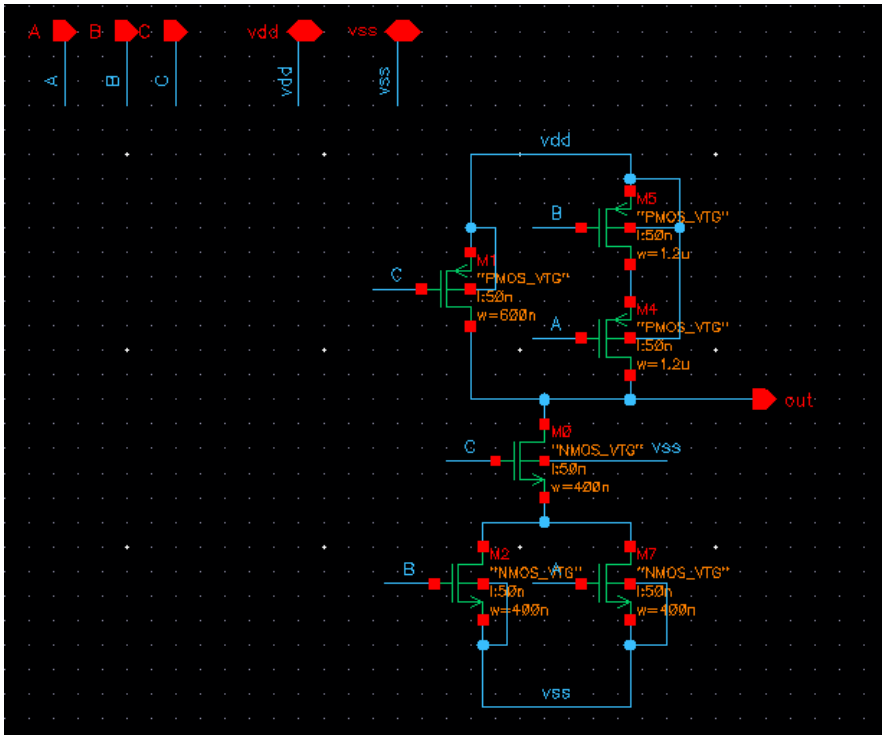


Figure 22: AOI

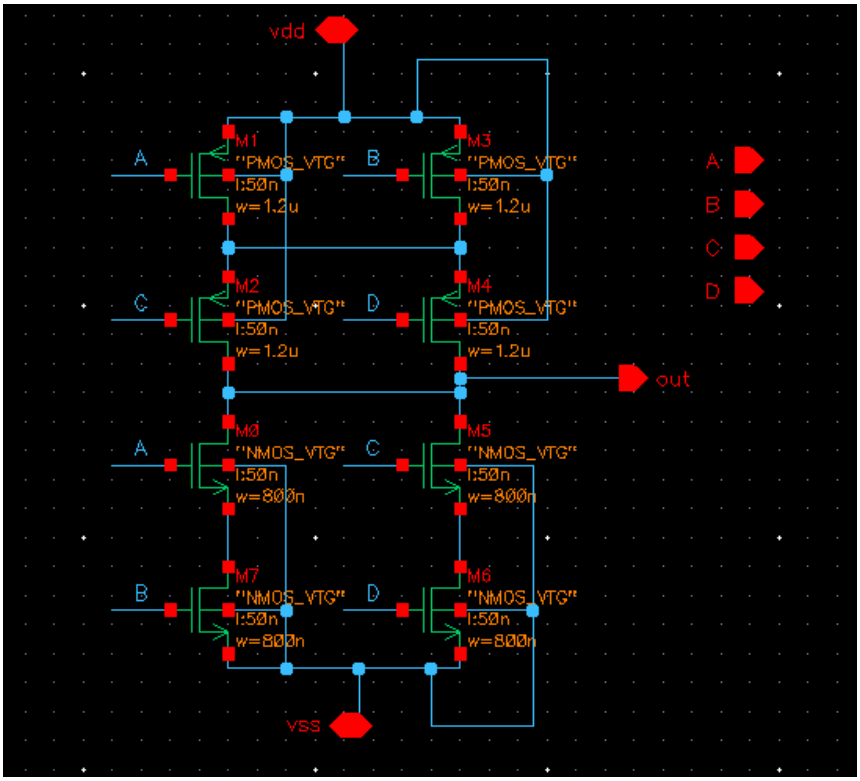


Figure 23: AOI22

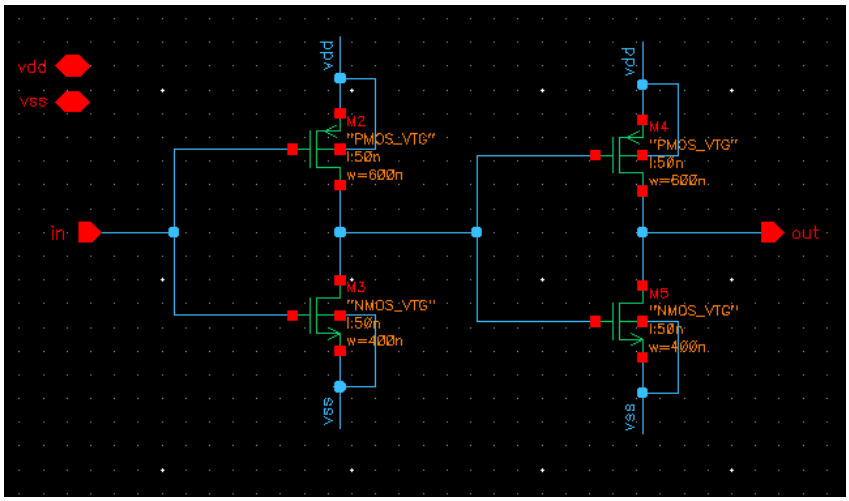


Figure 24: Buffer

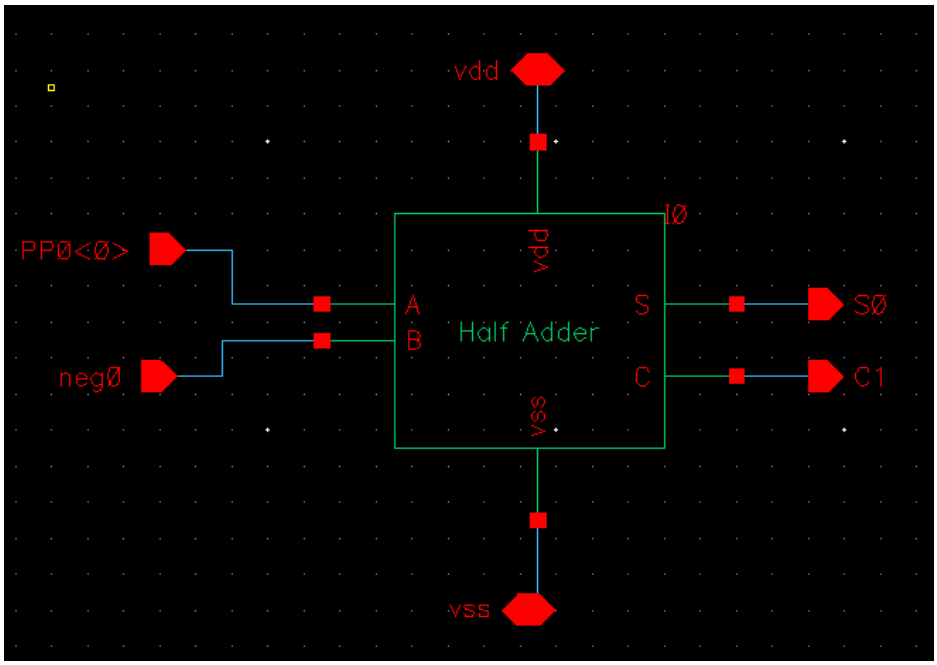


Figure 25: Column 0

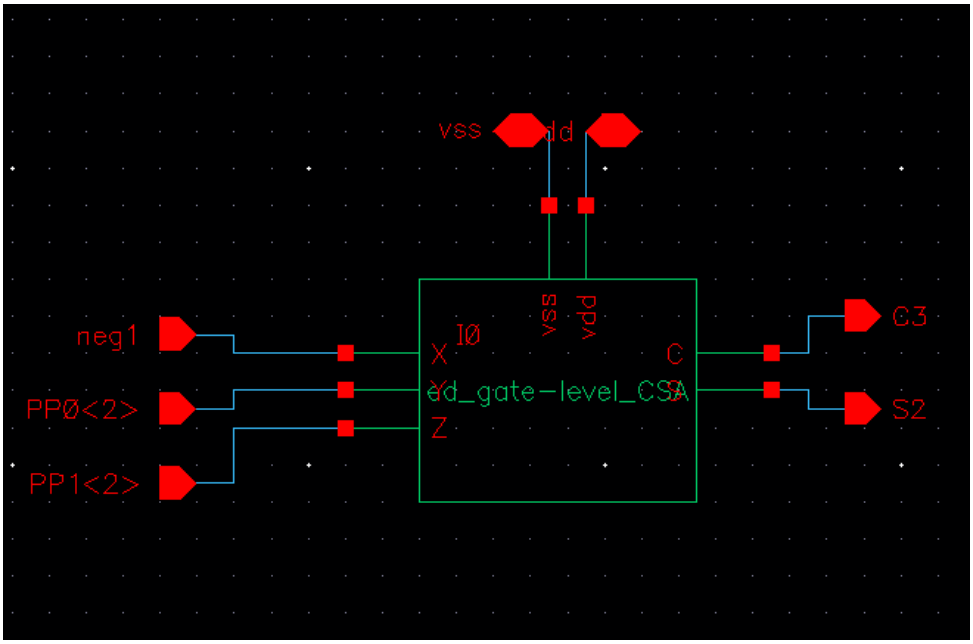


Figure 26: Column 2

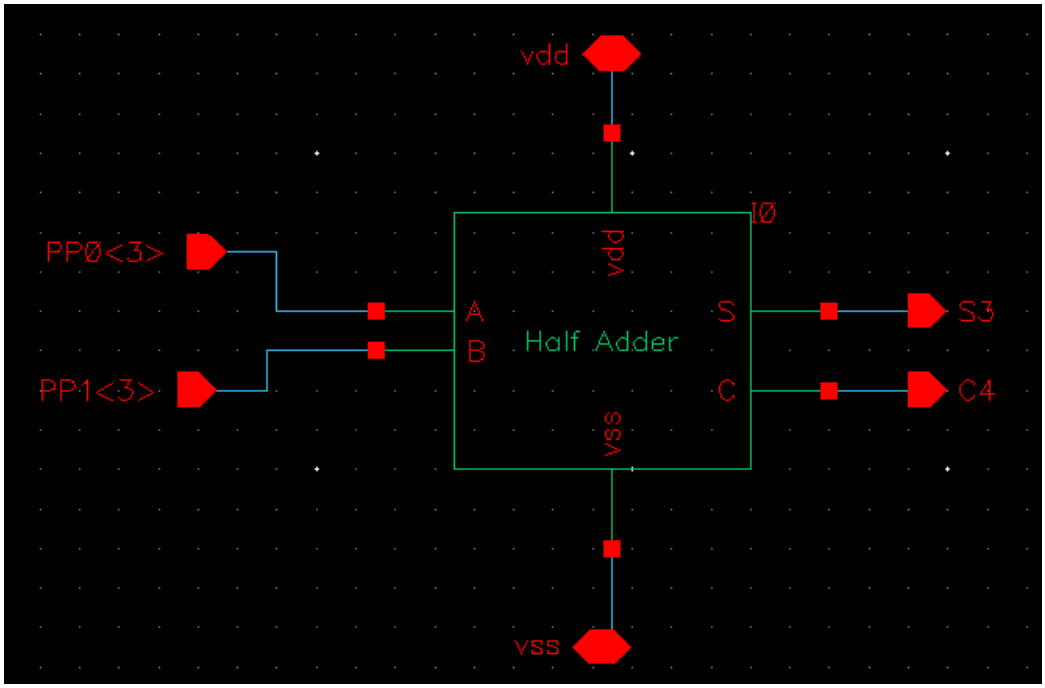


Figure 27: Column 3

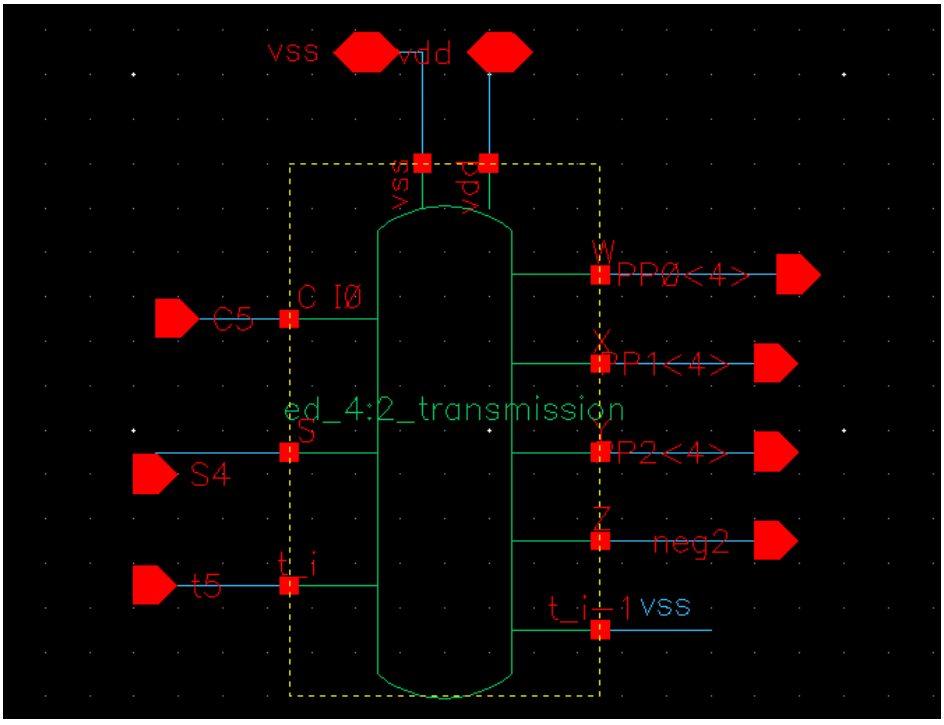


Figure 28: Column 4

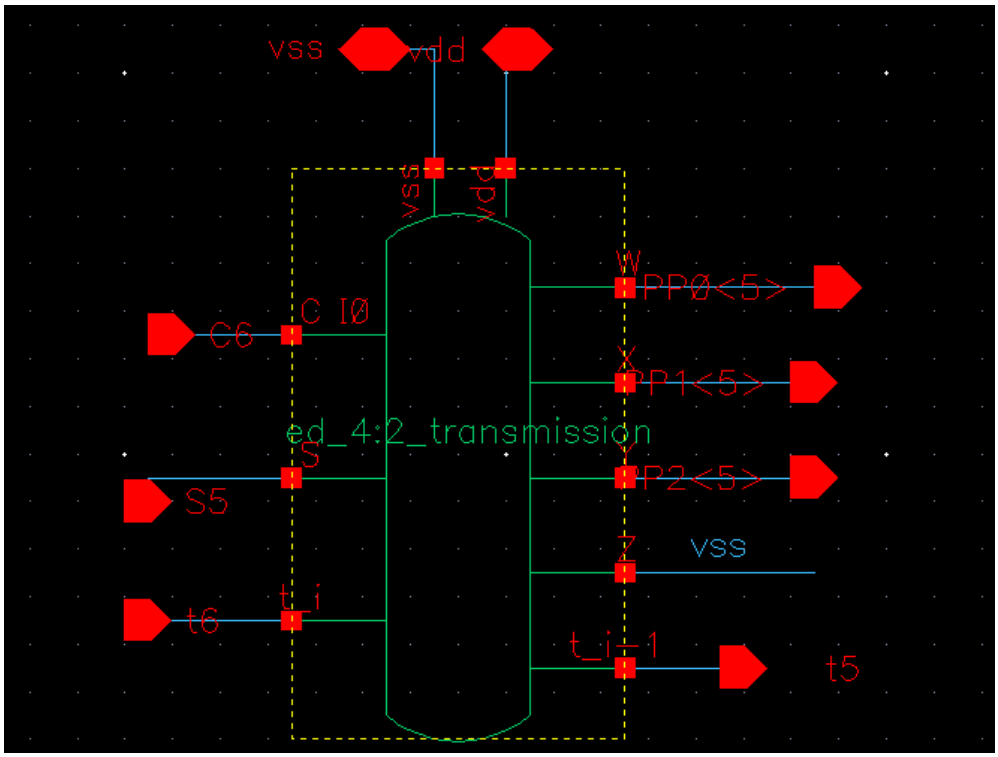


Figure 29: Column 5

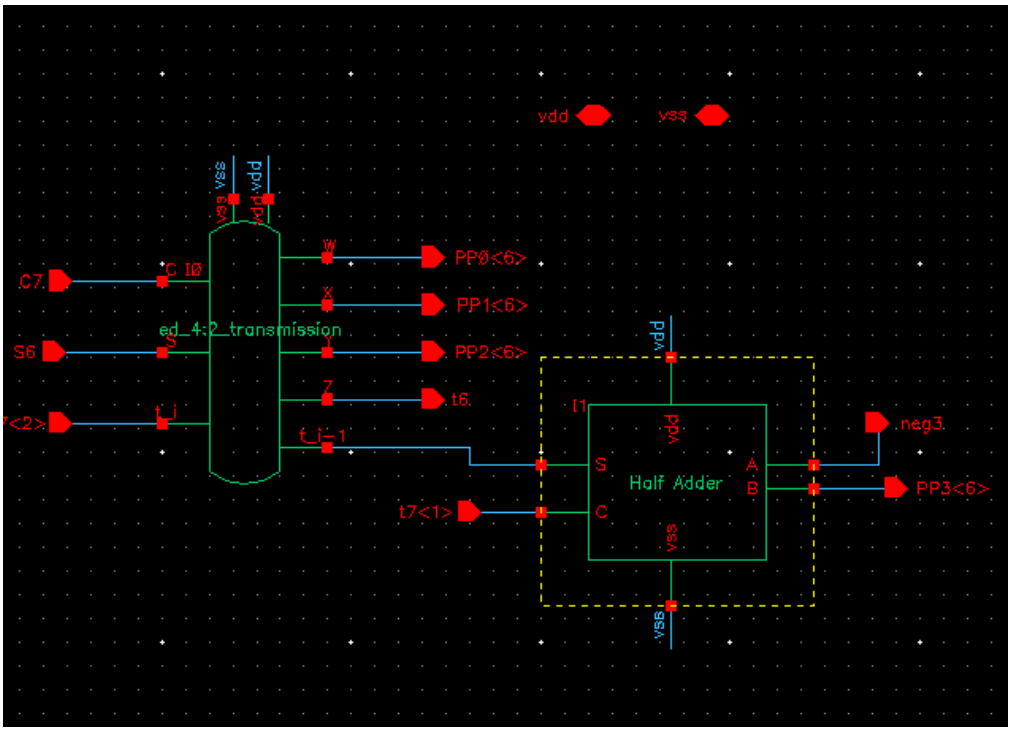


Figure 30: Column 6

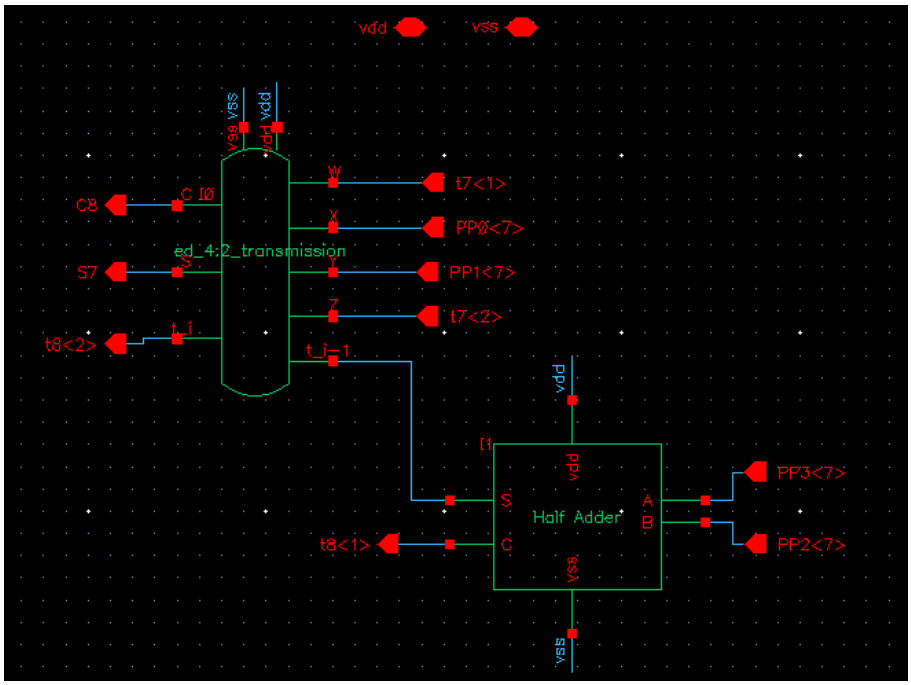


Figure 31: Column 7

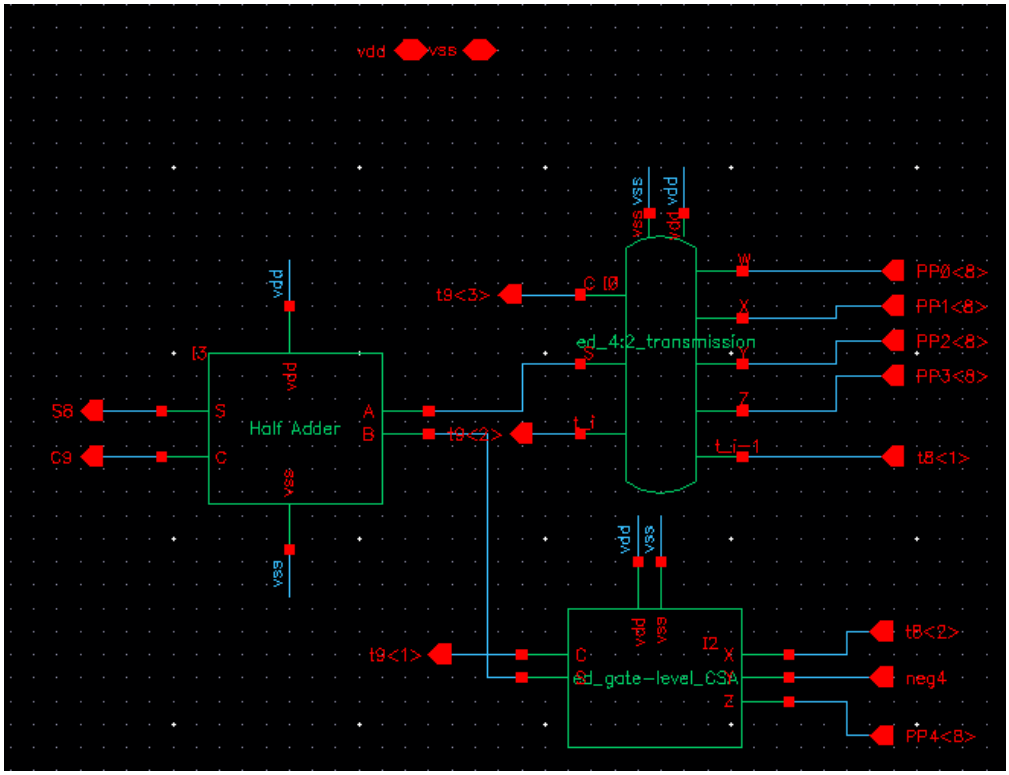


Figure 32: Column 8

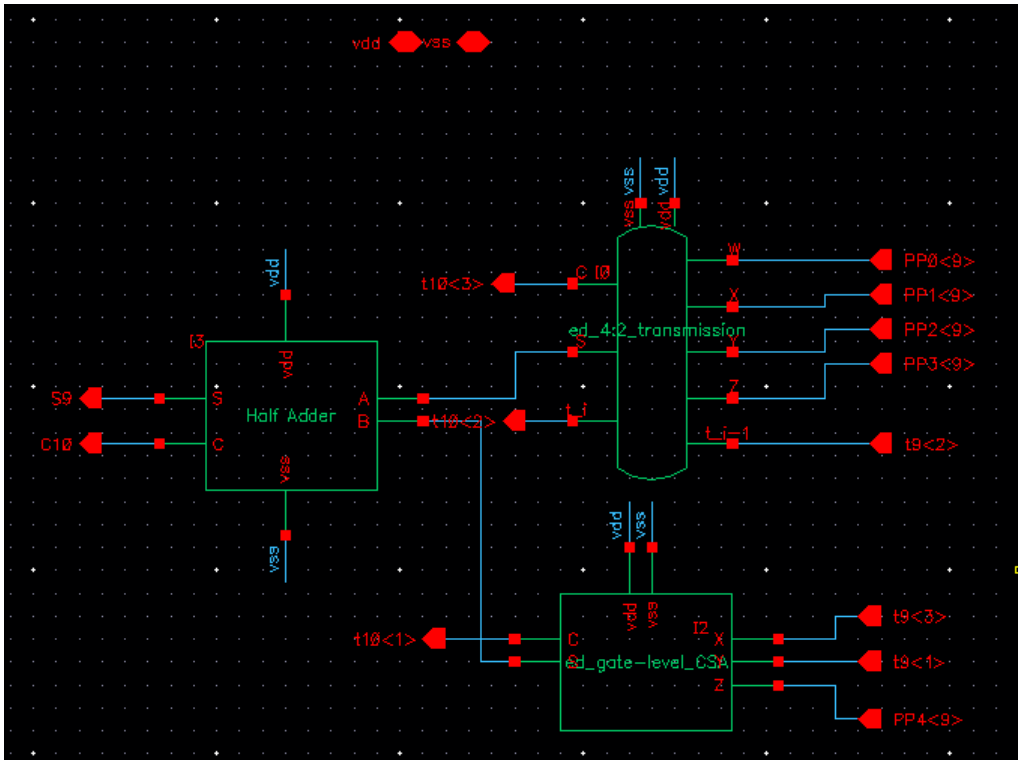


Figure 33: Column 9

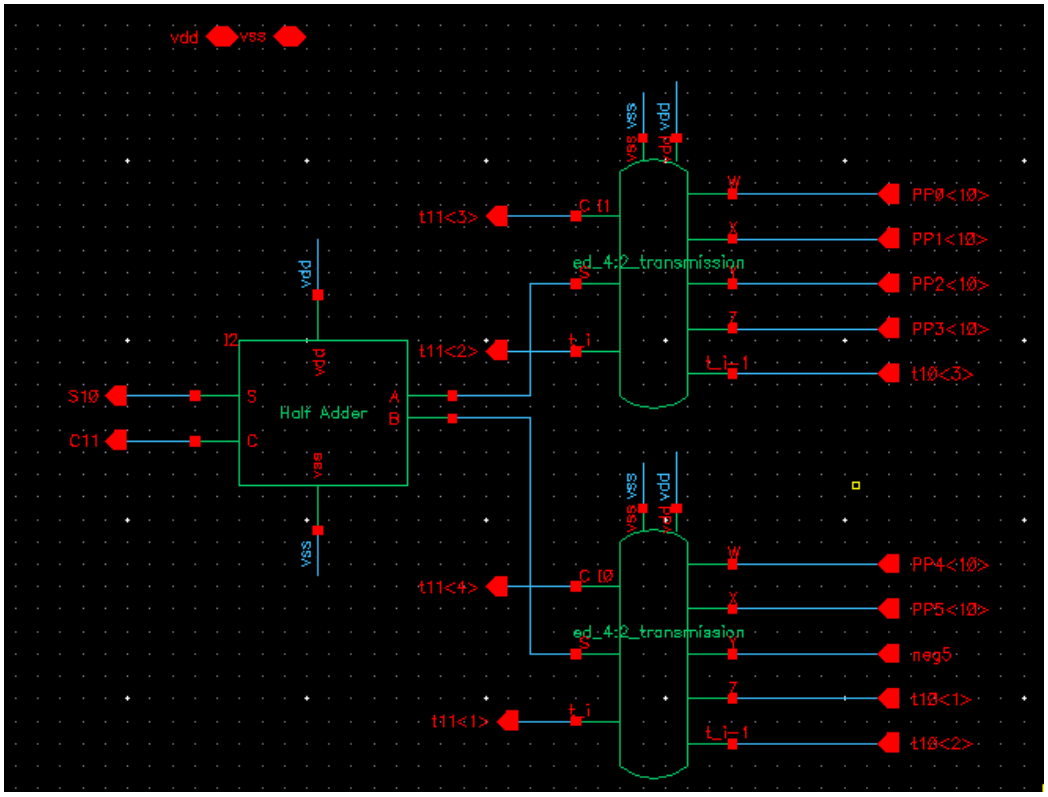


Figure 34: Column 10

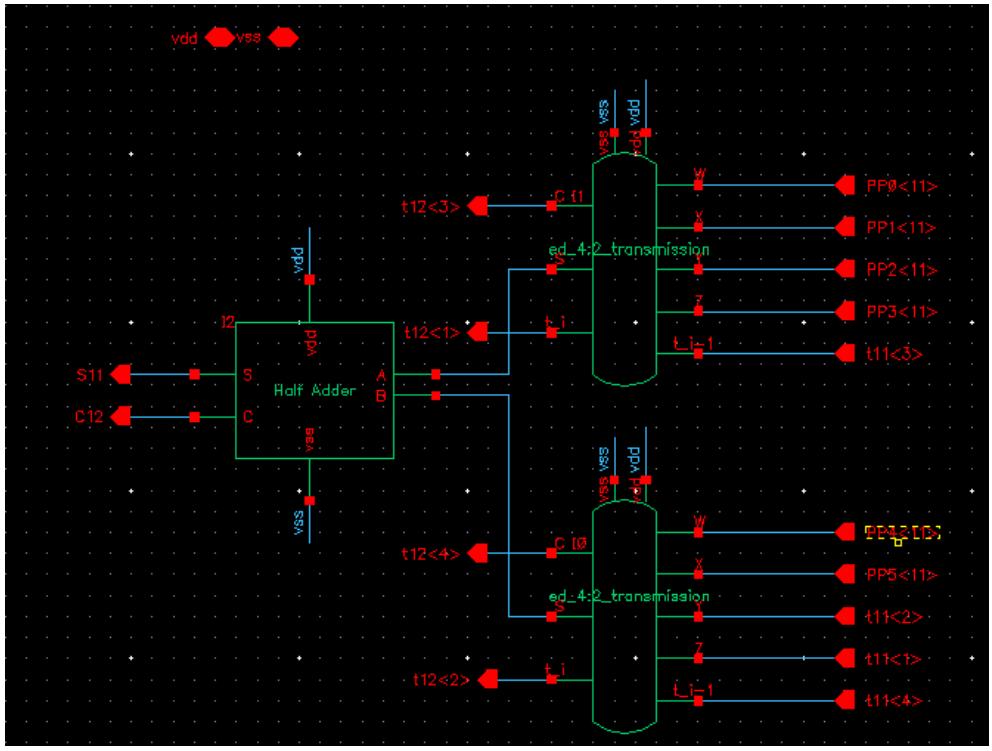


Figure 35: Column 11

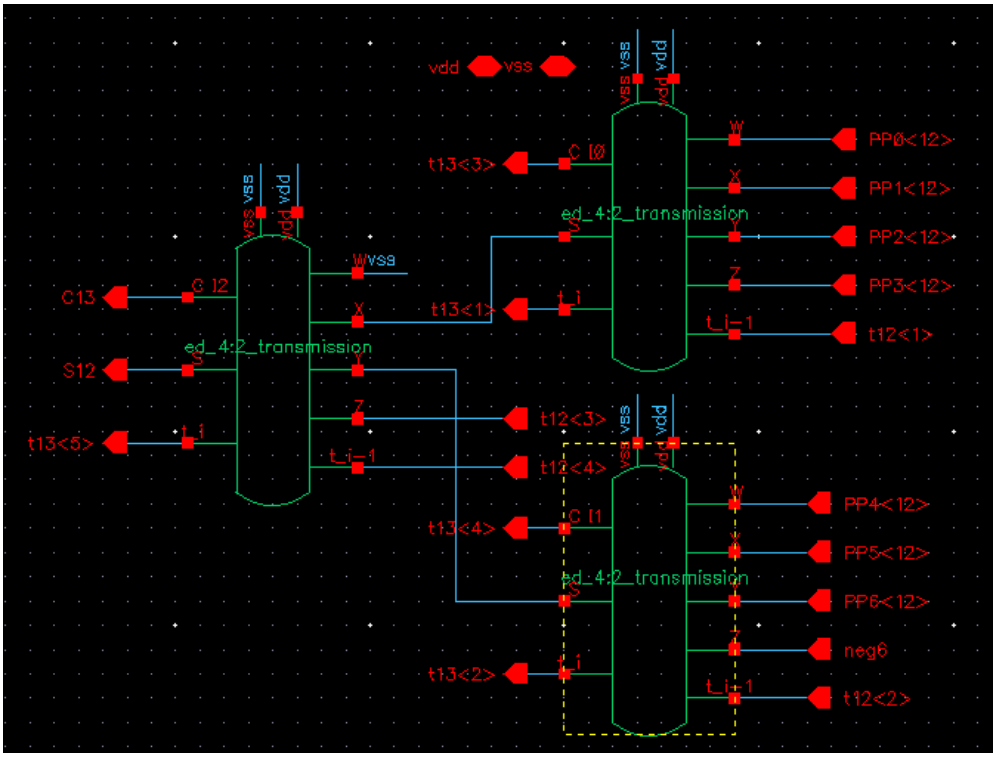


Figure 36: Column 12

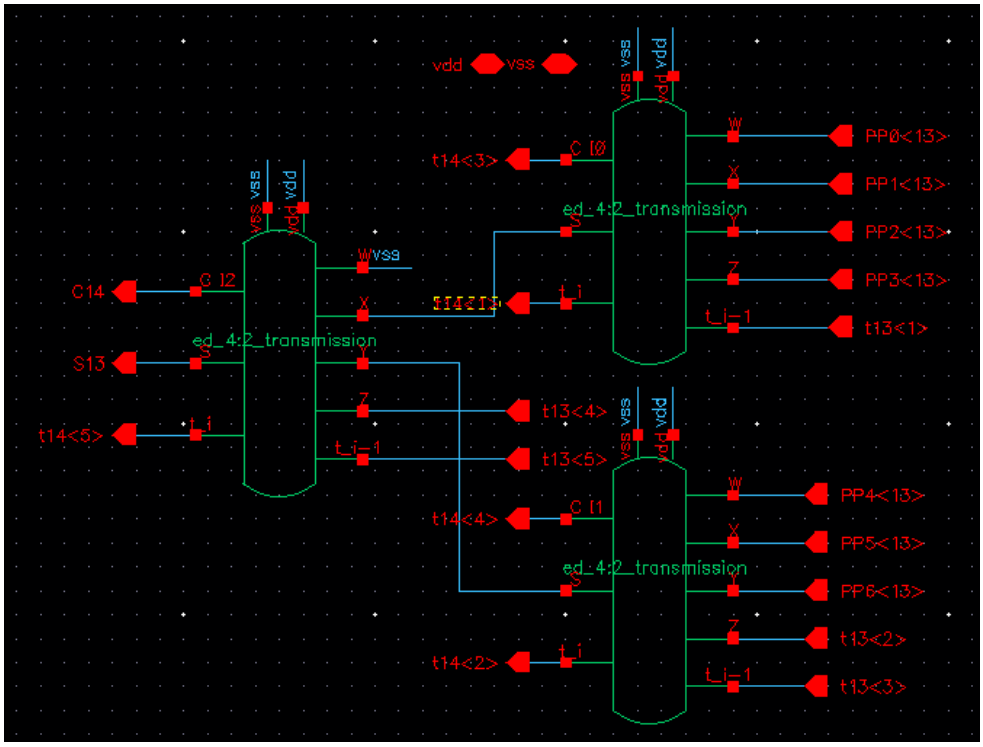


Figure 37: Column 13

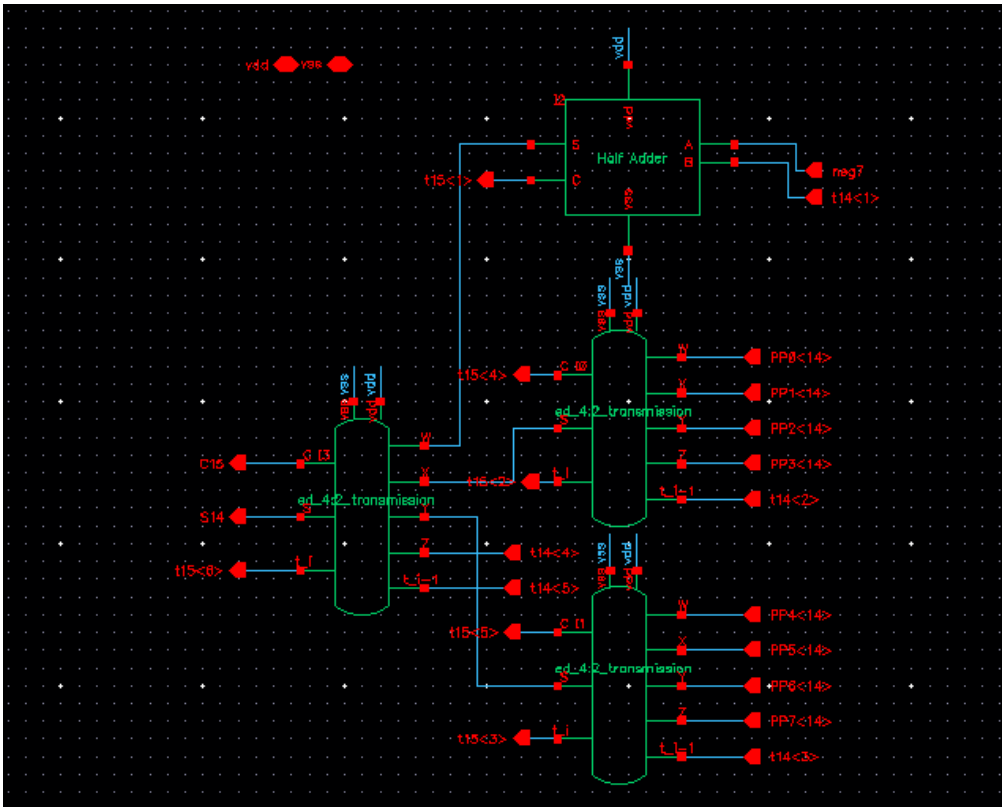


Figure 38: Column 14

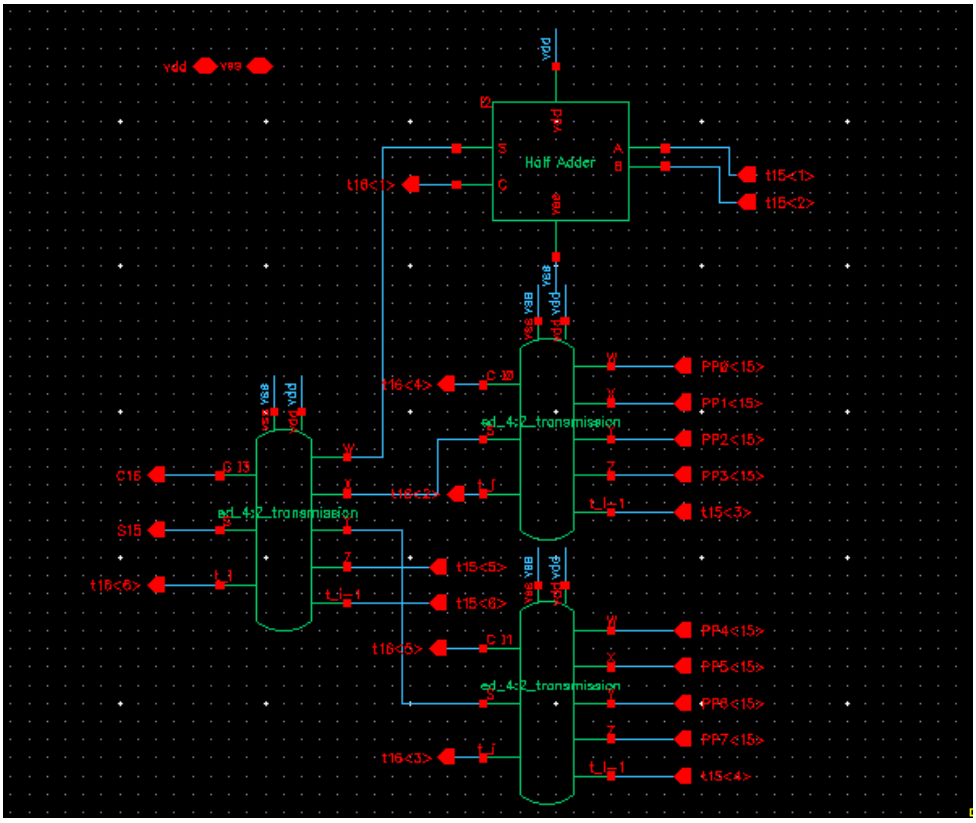


Figure 39: Column 15

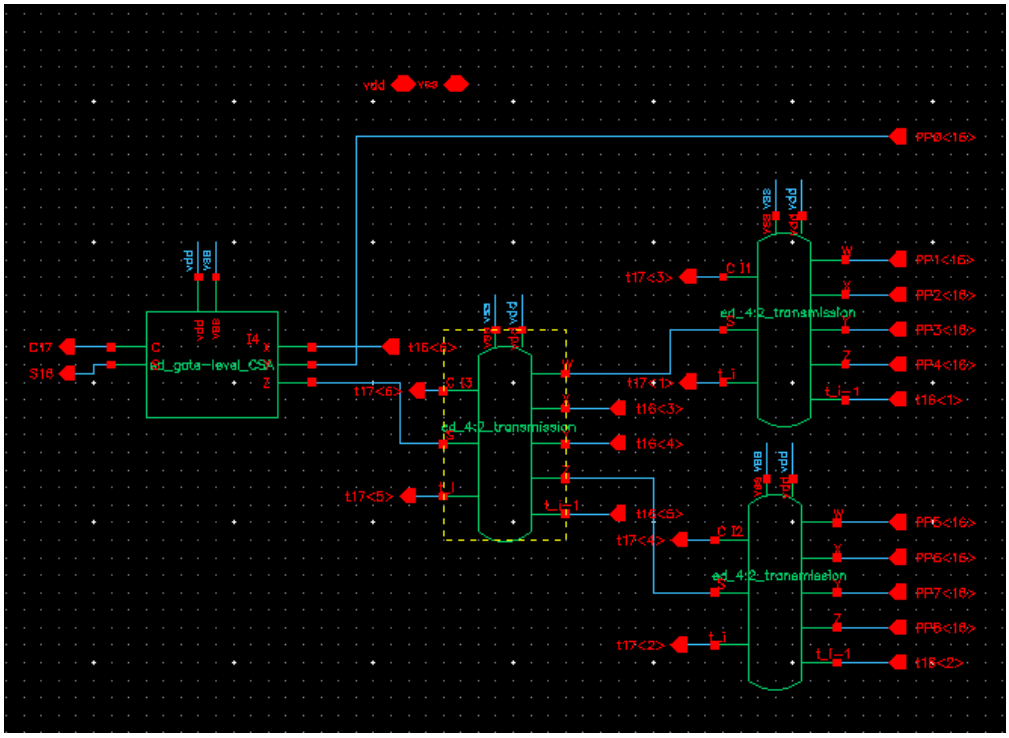


Figure 40: Column 16

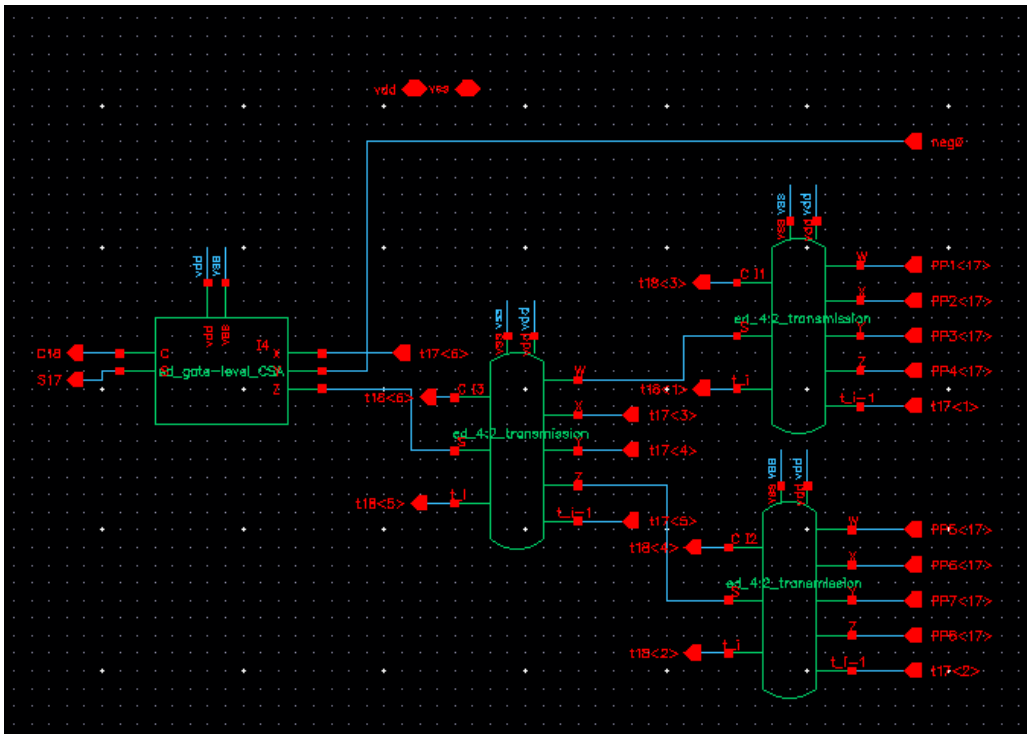


Figure 41: Column 17

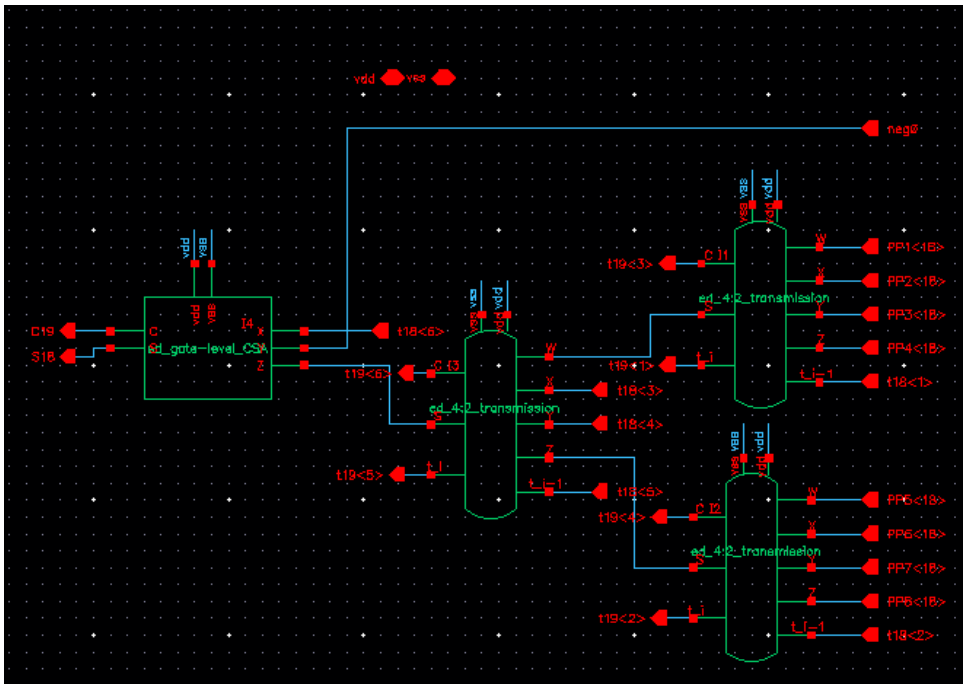


Figure 42: Column 18

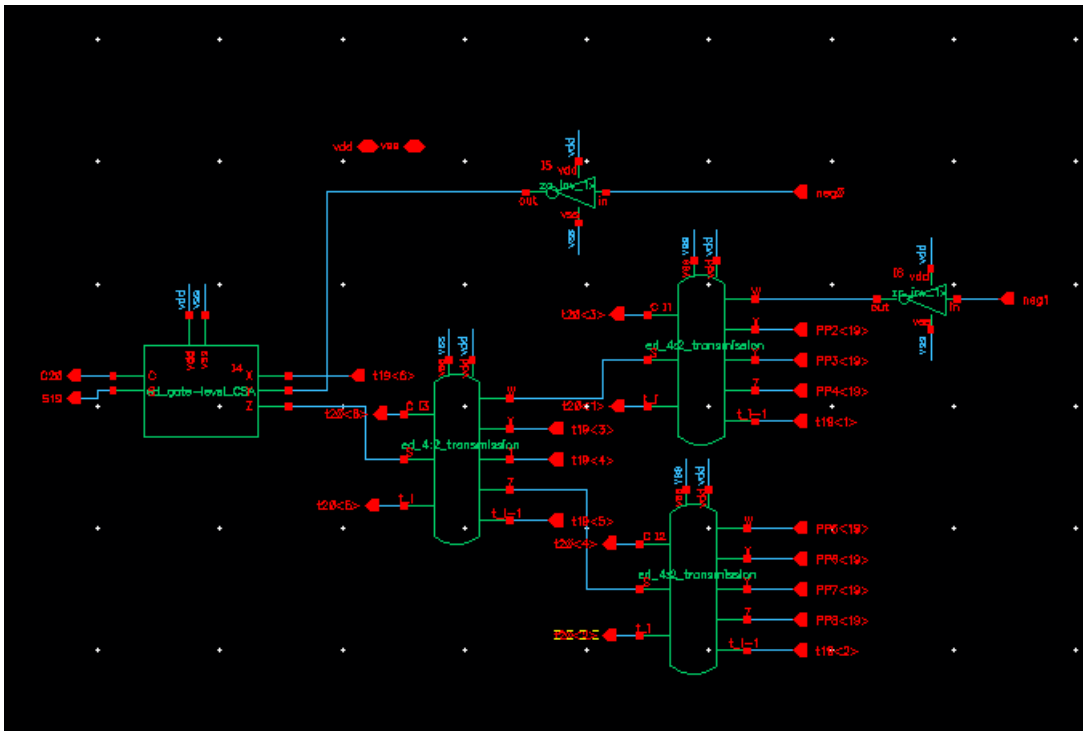


Figure 43: Column 19

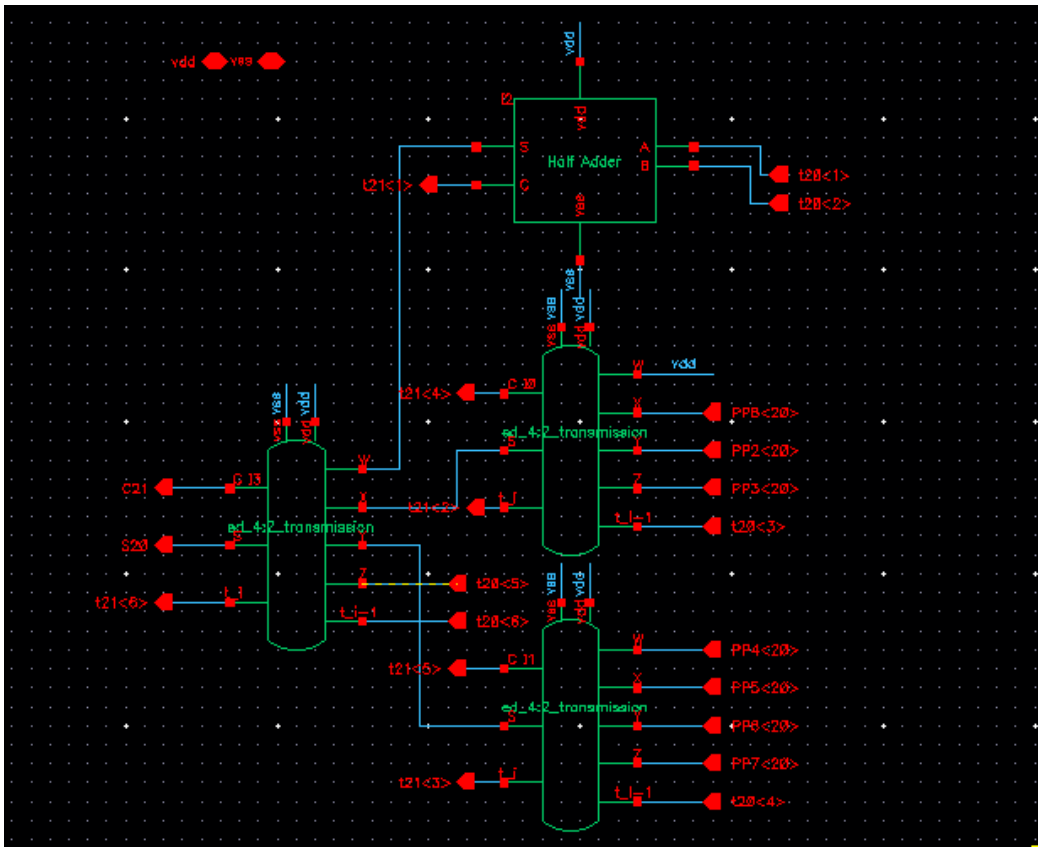


Figure 44: Column 20

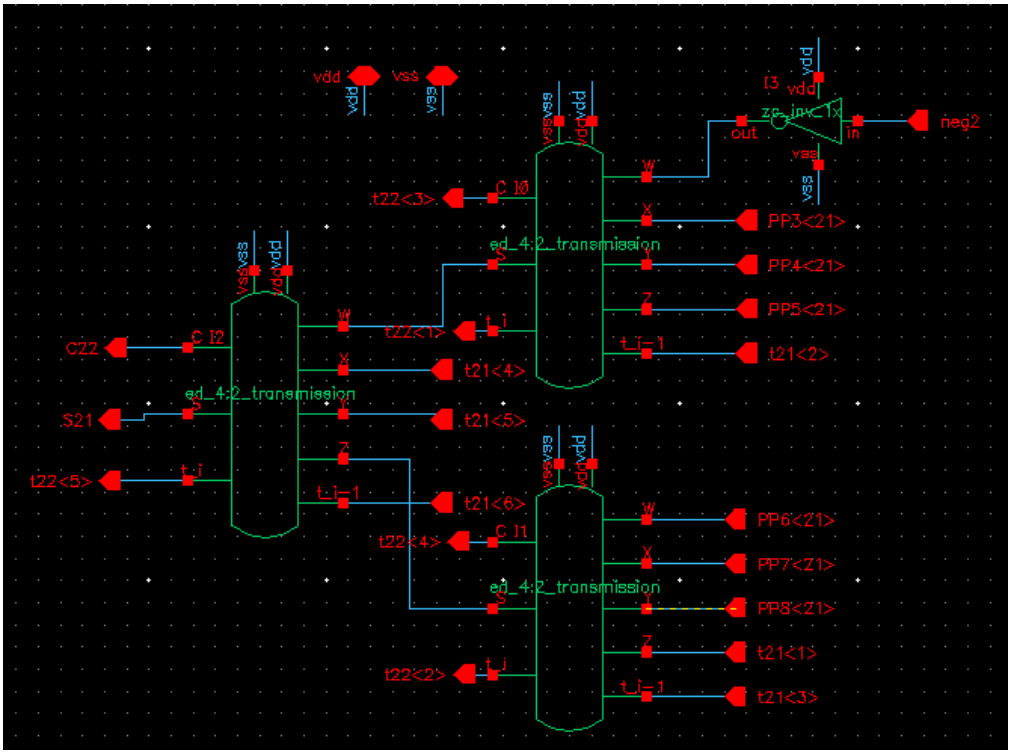


Figure 45: Column 21

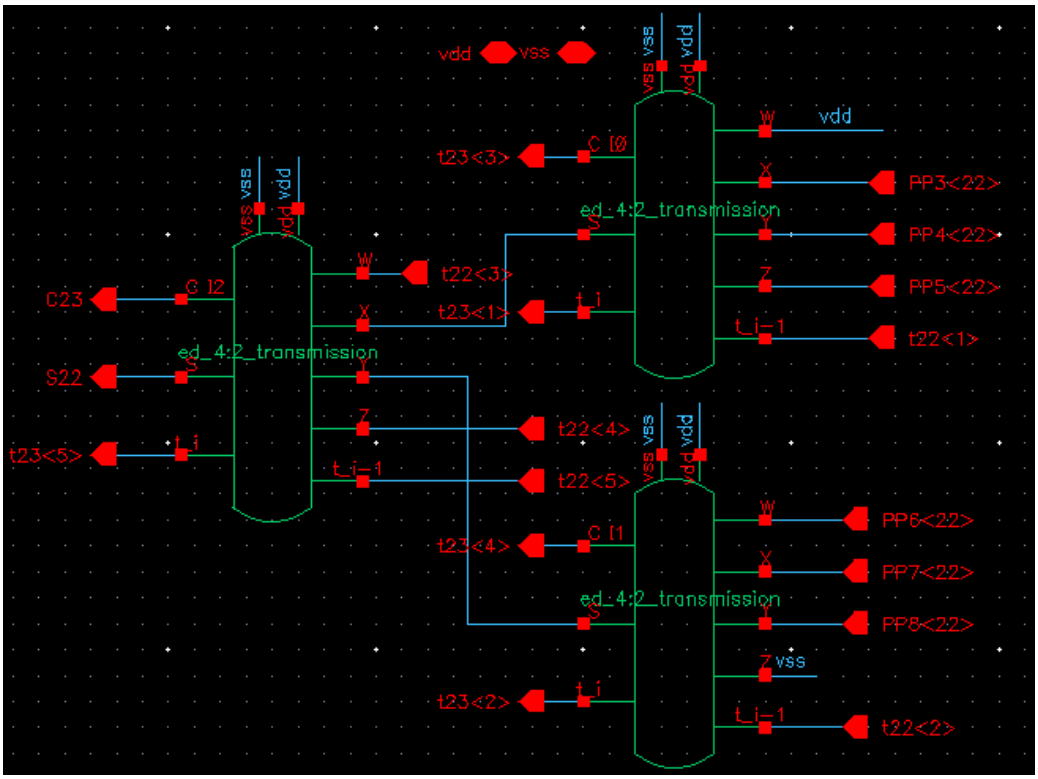


Figure 46: Column 22

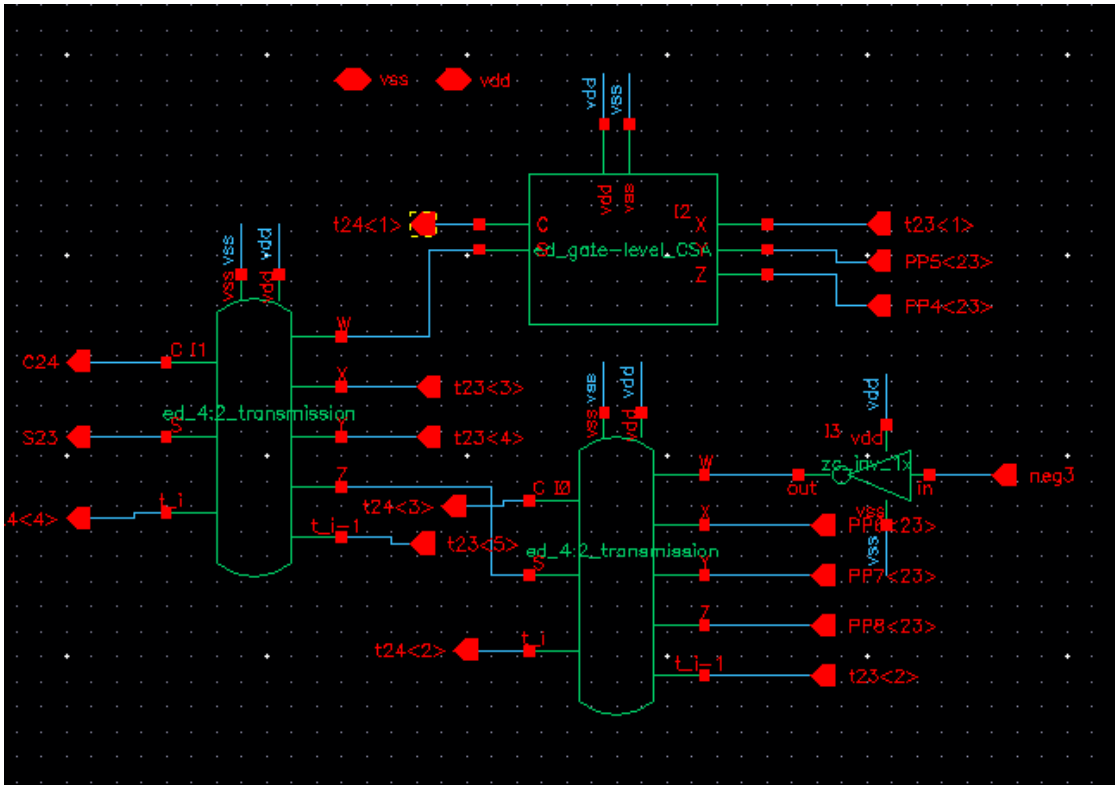


Figure 47: Column 23

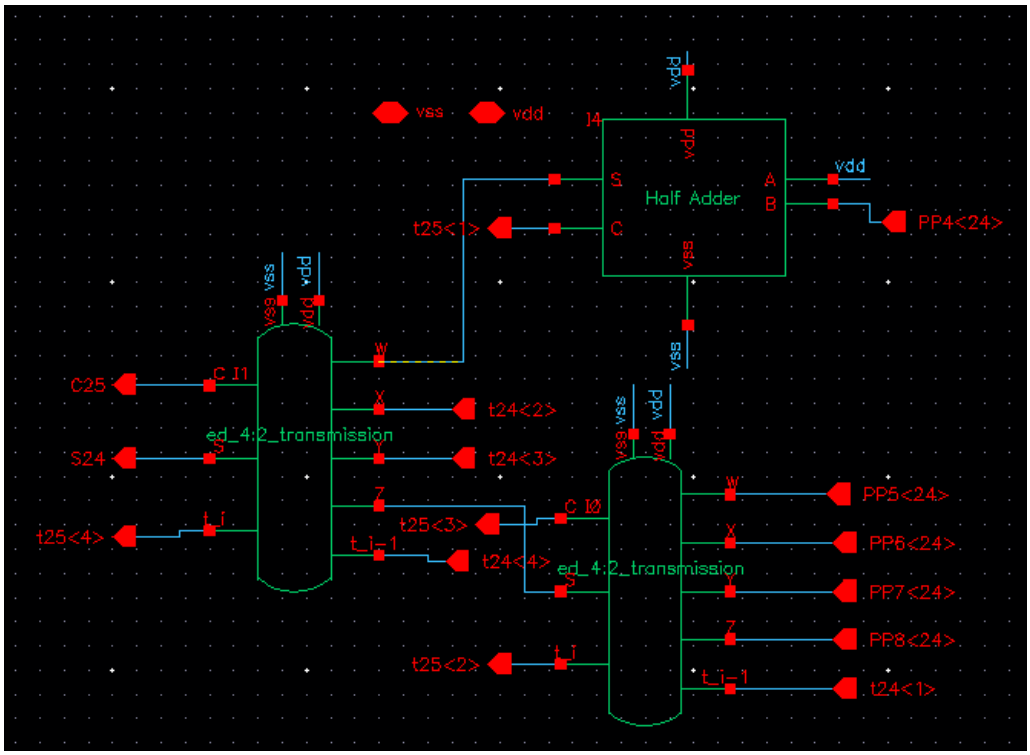


Figure 48: Column 24

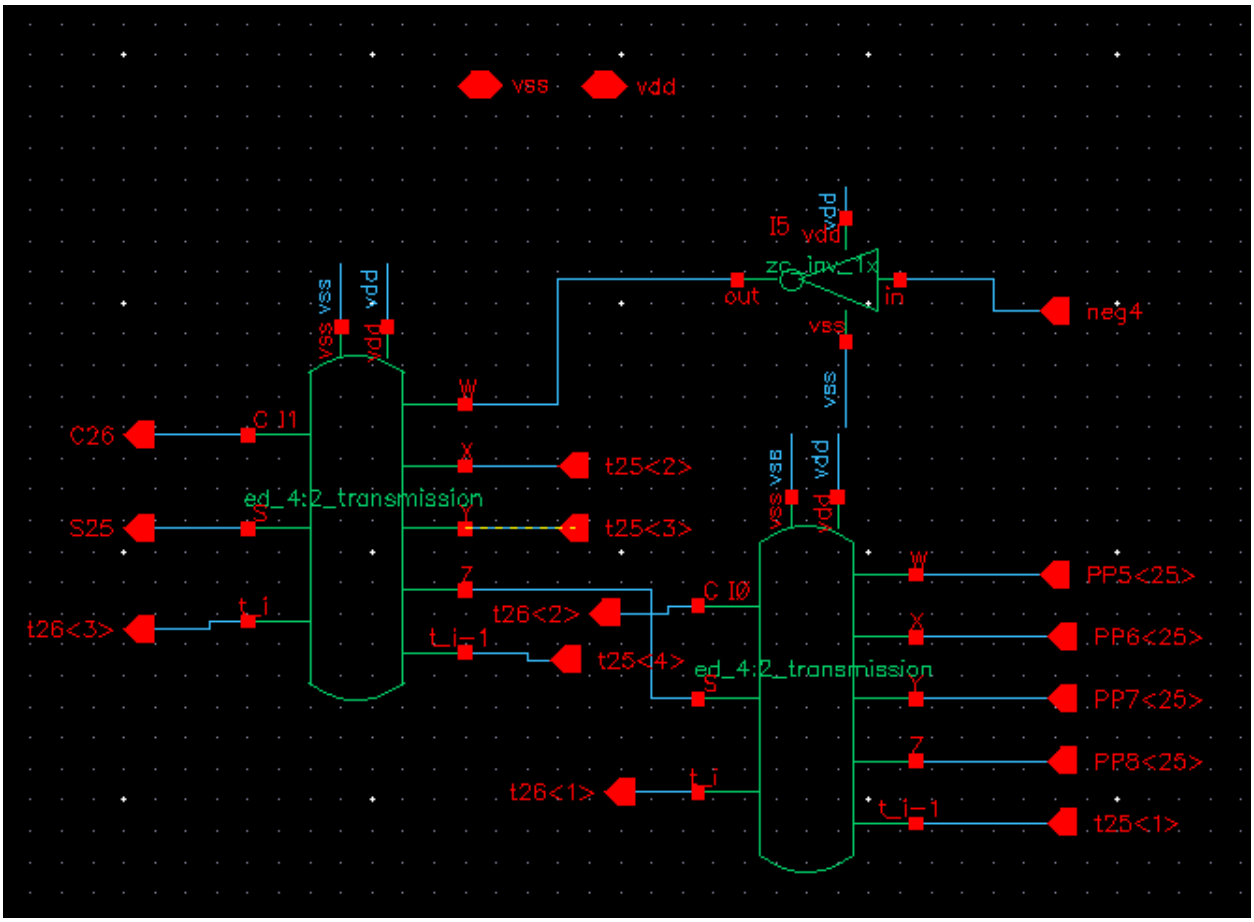


Figure 49: Column 25

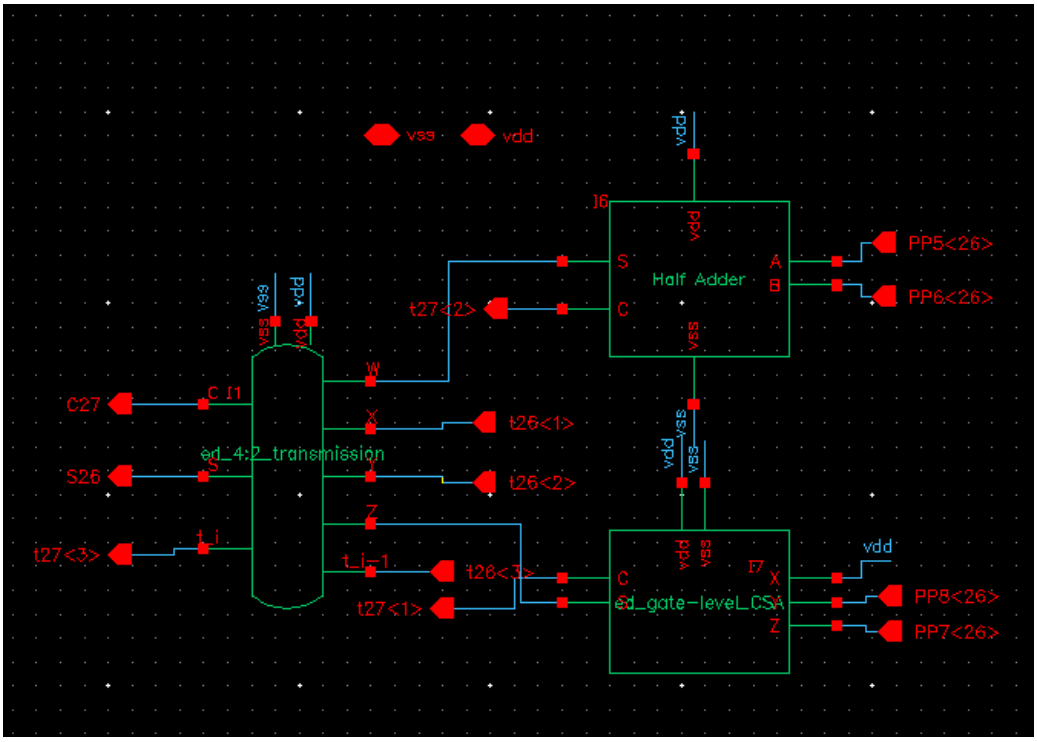


Figure 50: Column 26

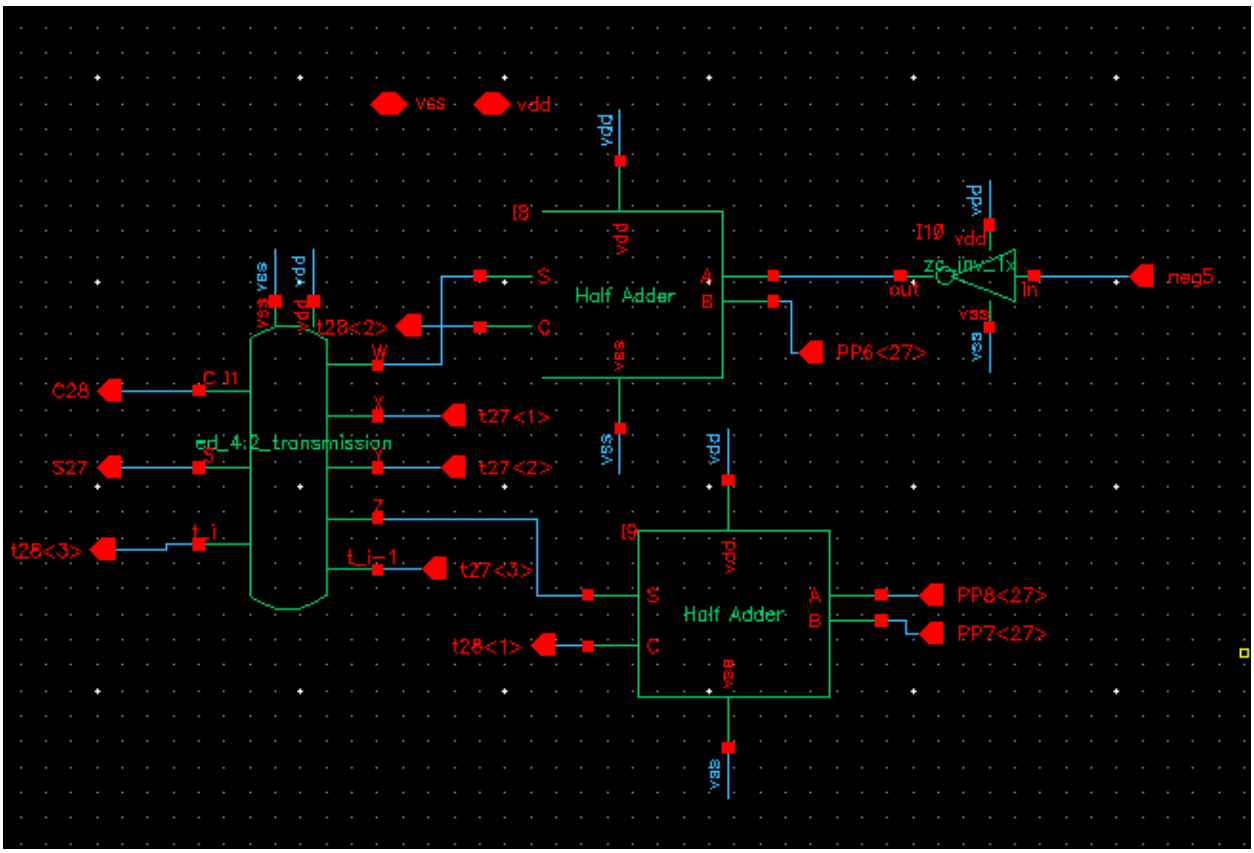


Figure 51: Column 27

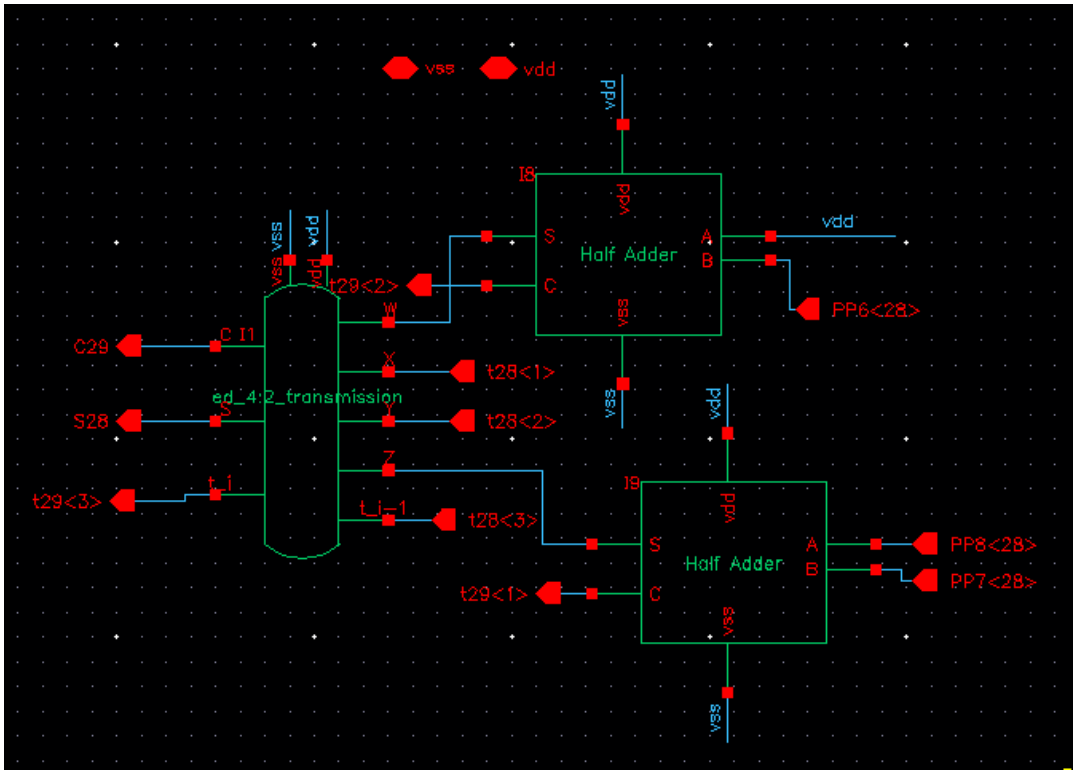


Figure 52: Column 28

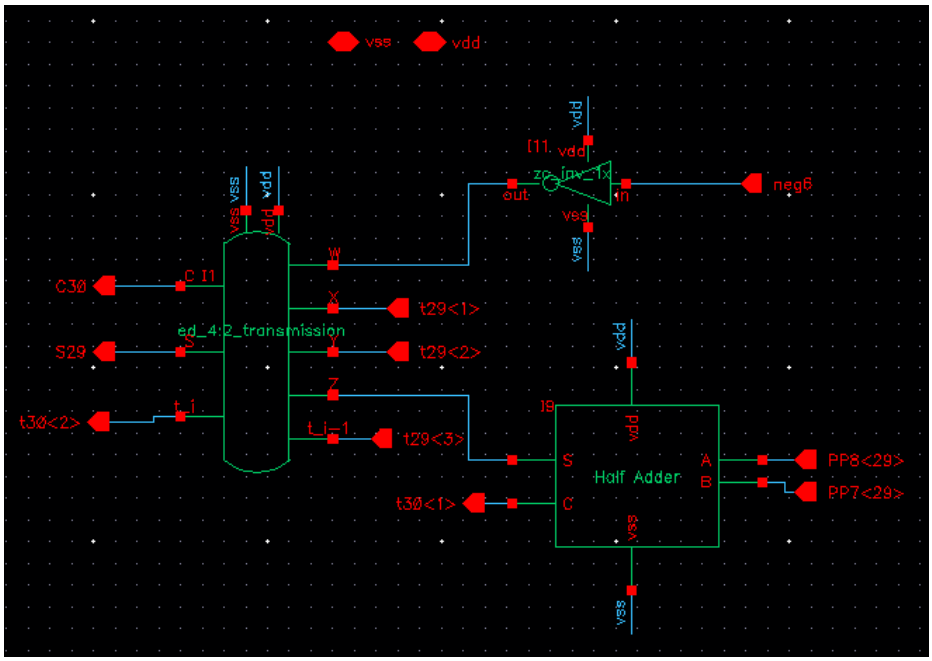


Figure 53: Column 29

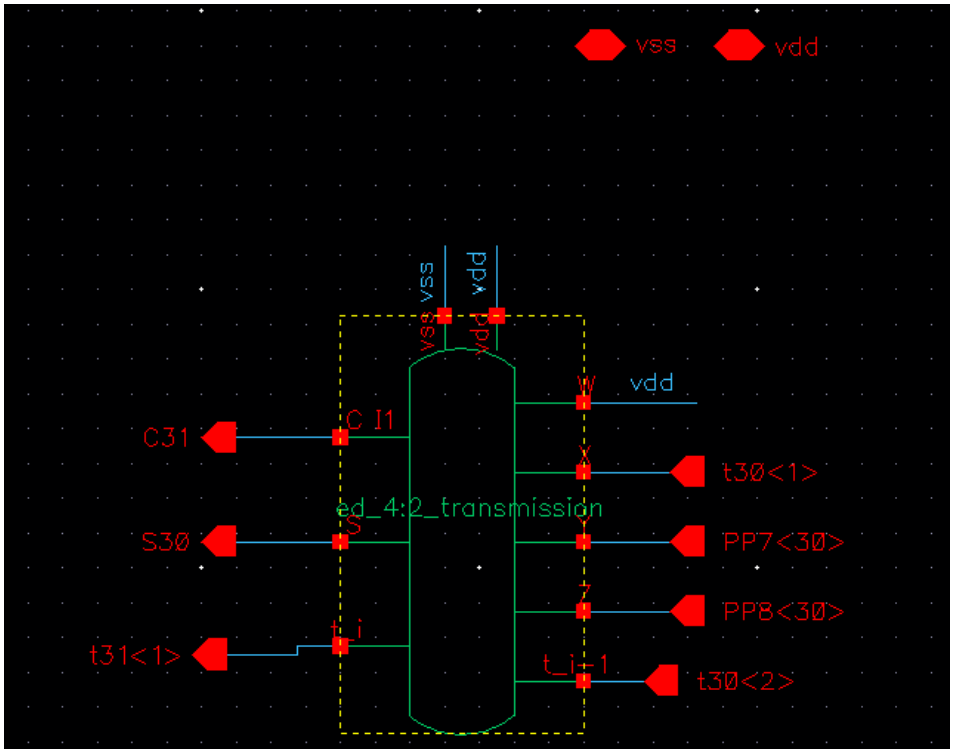


Figure 54: Column 30

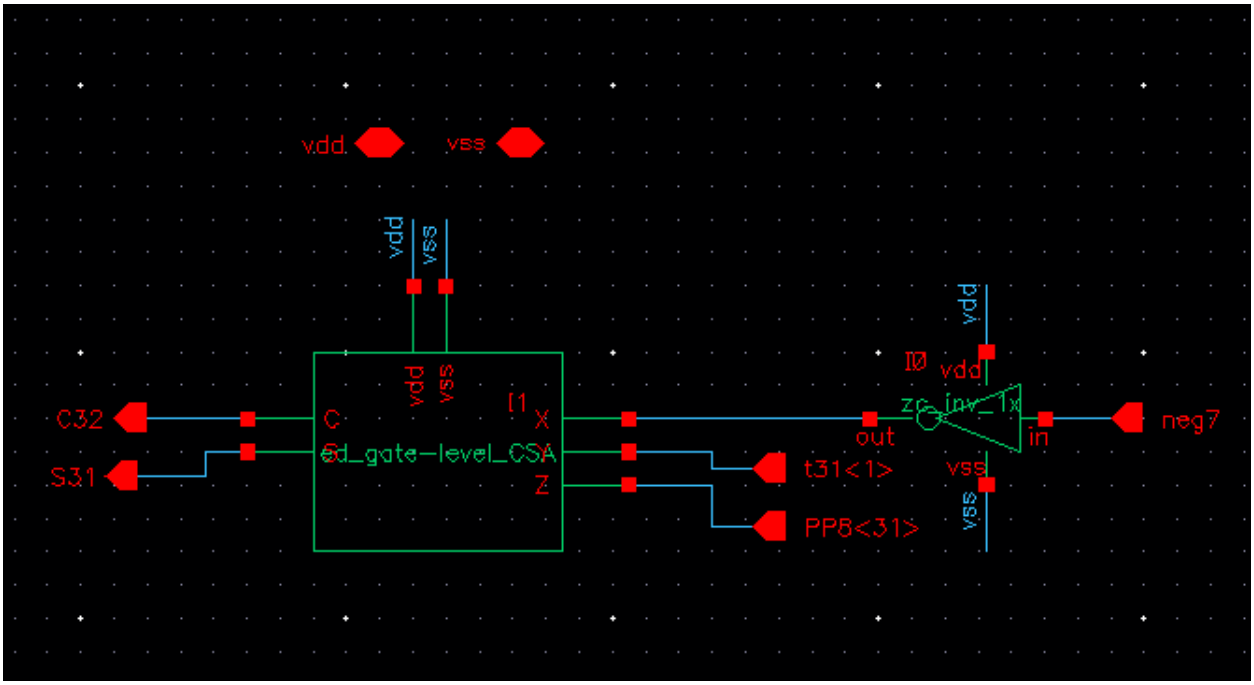


Figure 55: Column 31

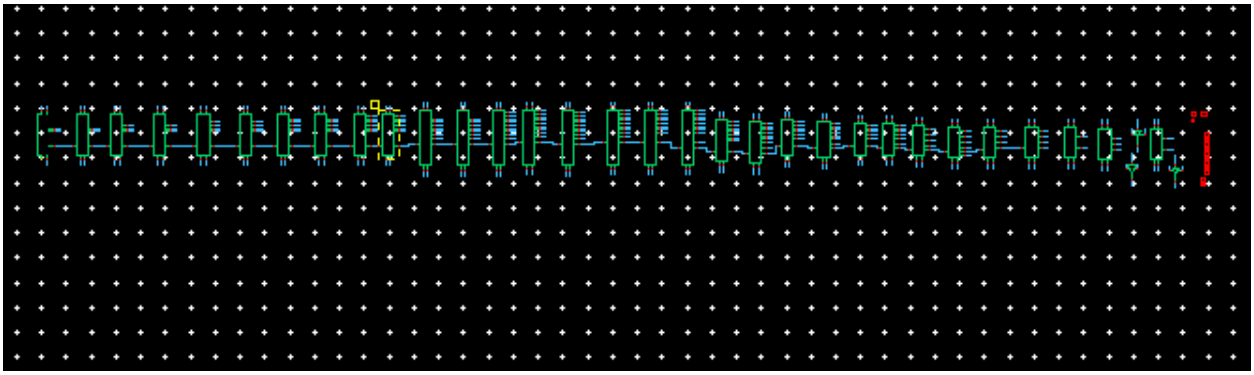


Figure 56: Column Addition

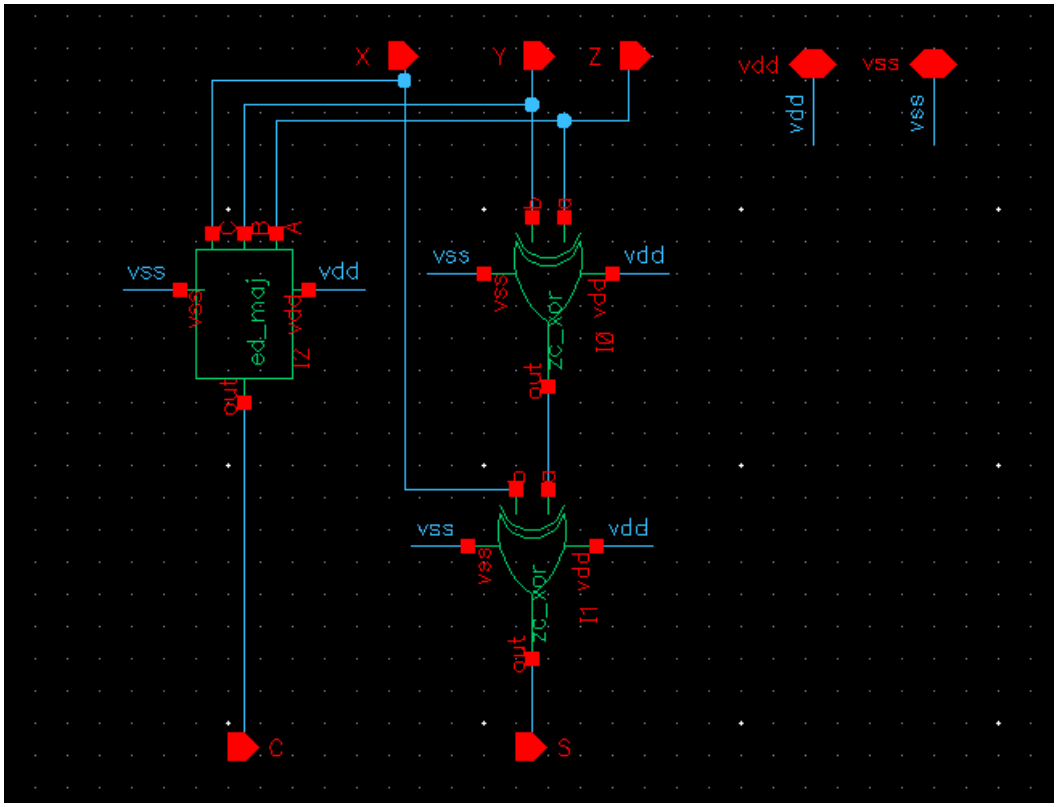


Figure 57: Carry-Save Adder

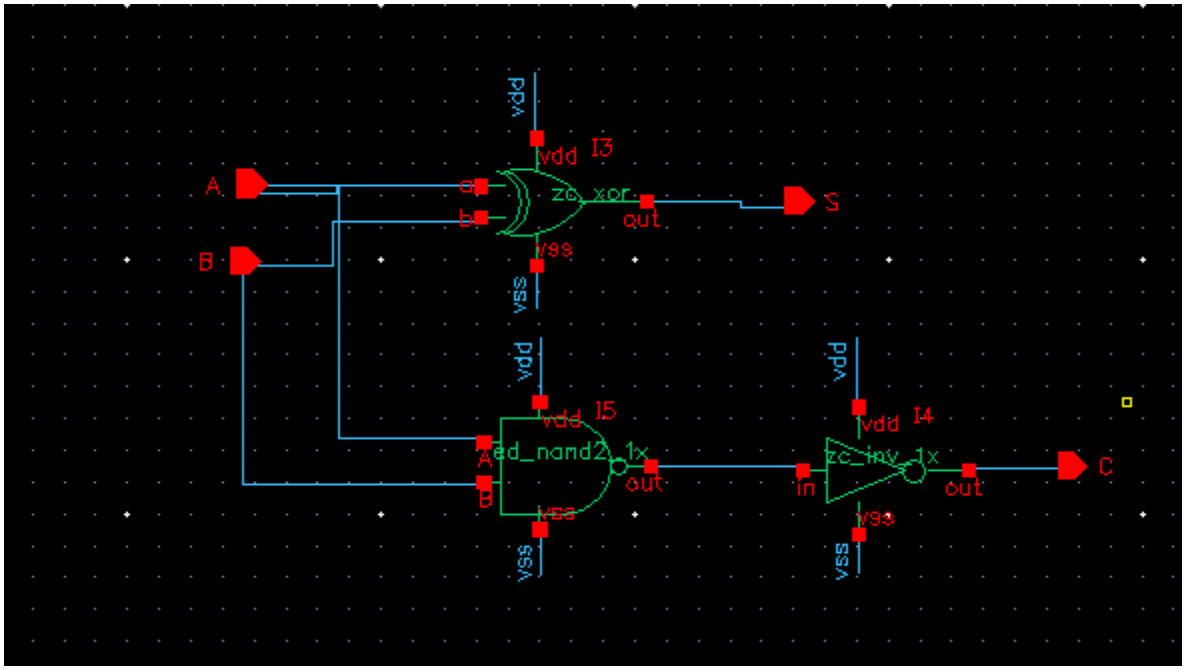


Figure 58: Half Adder

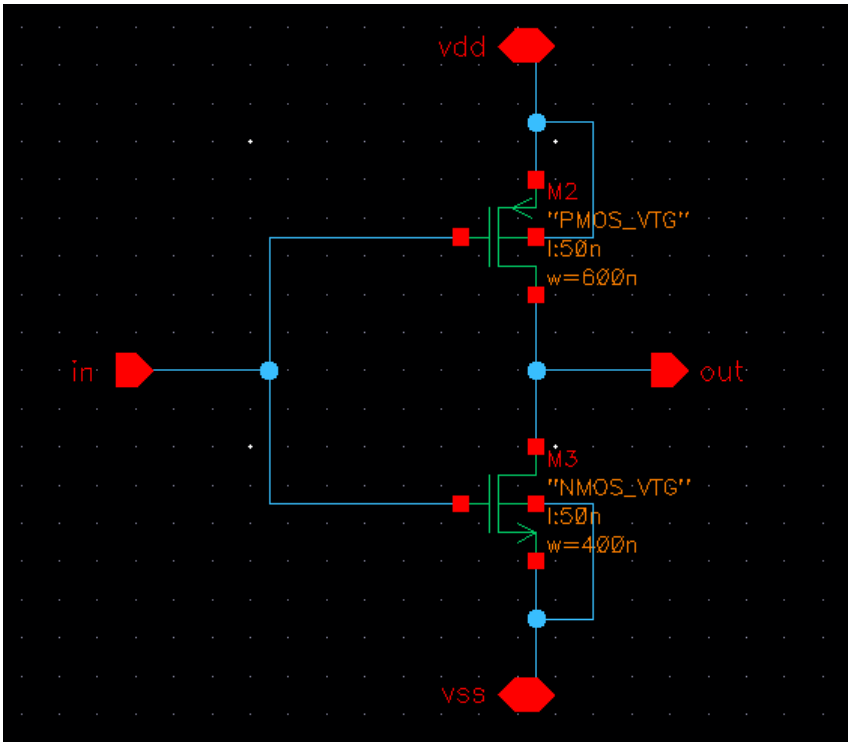


Figure 59: inverter

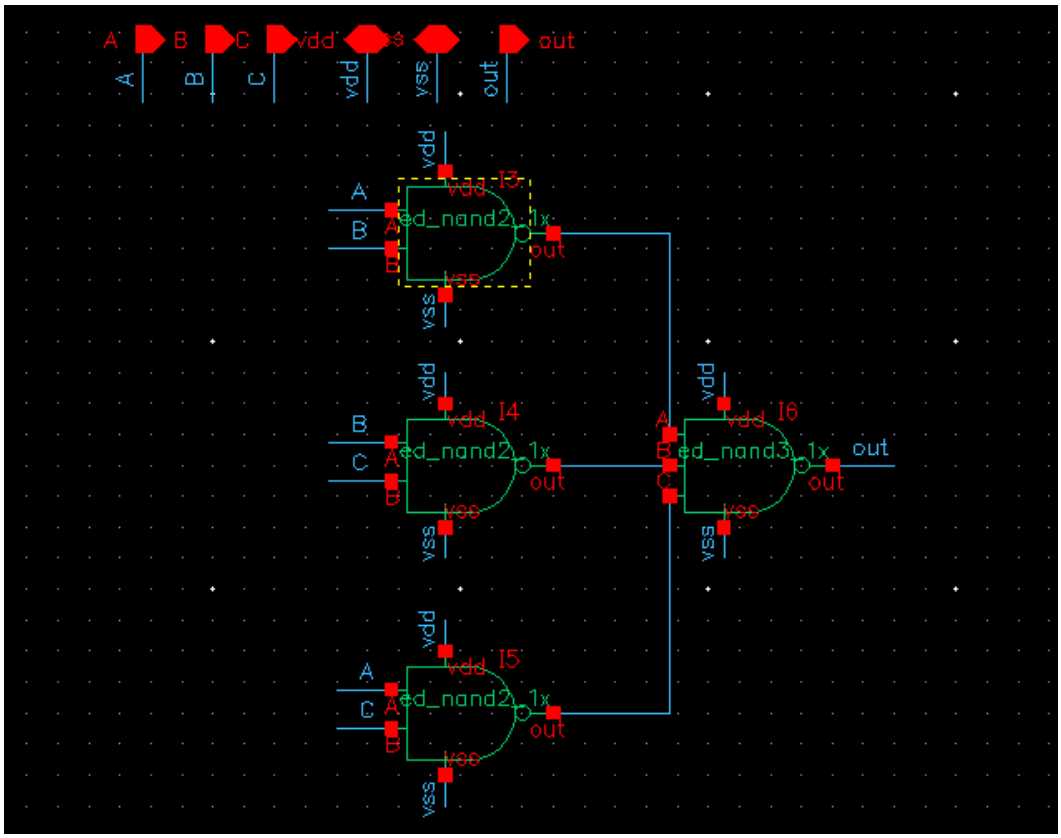


Figure 60: Majority

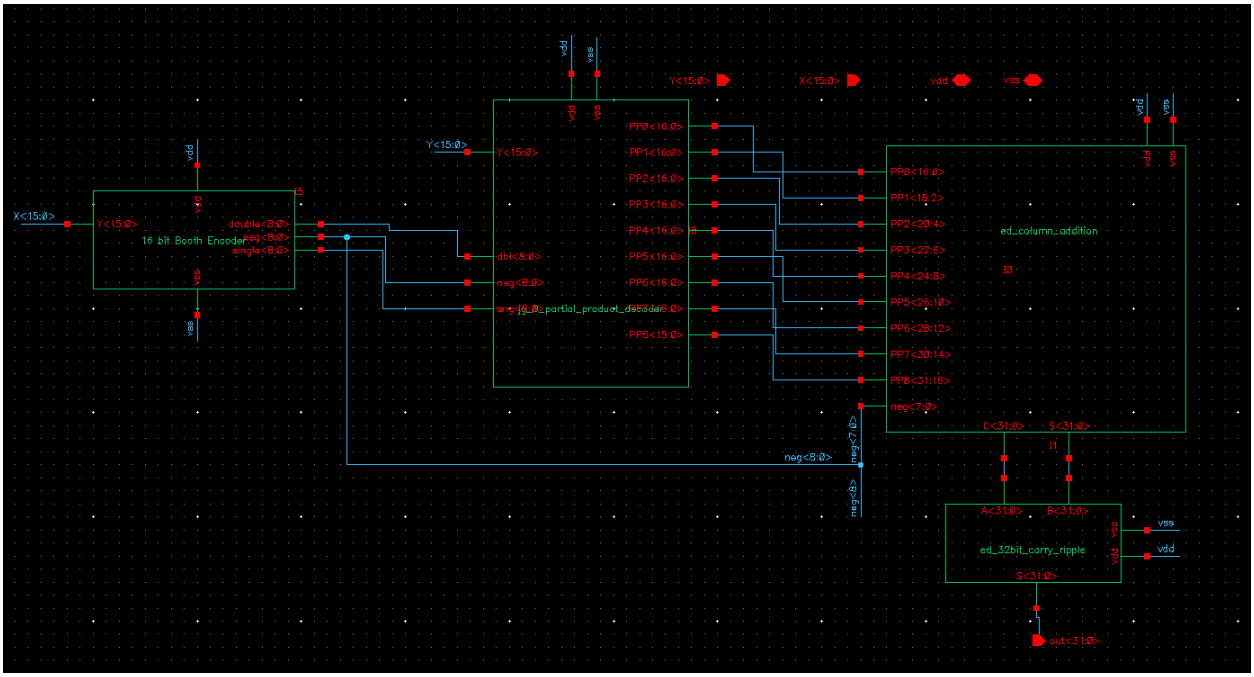


Figure 61: Multiplier

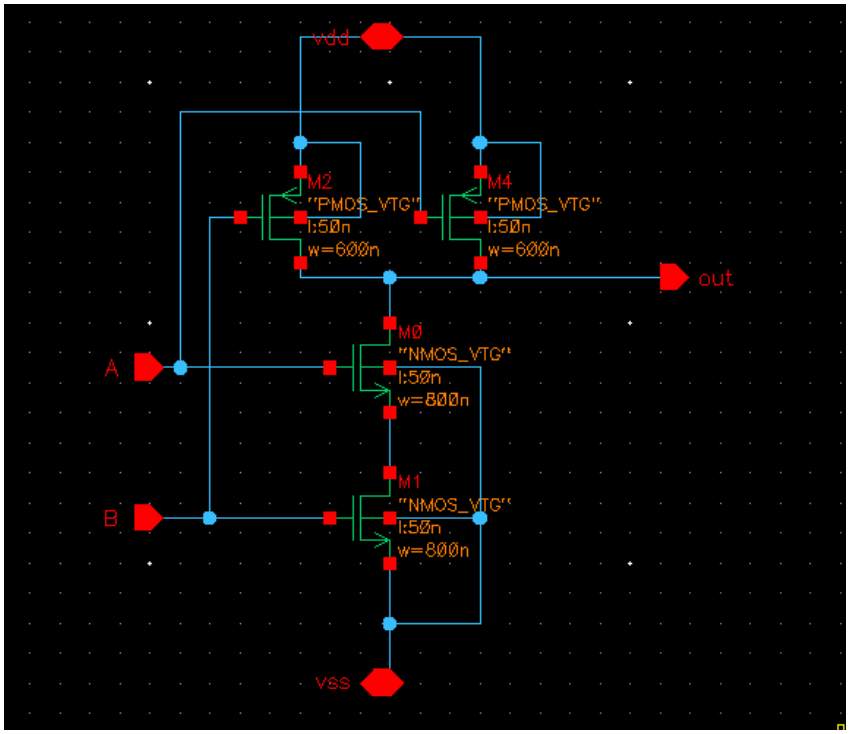


Figure 62: NAND2

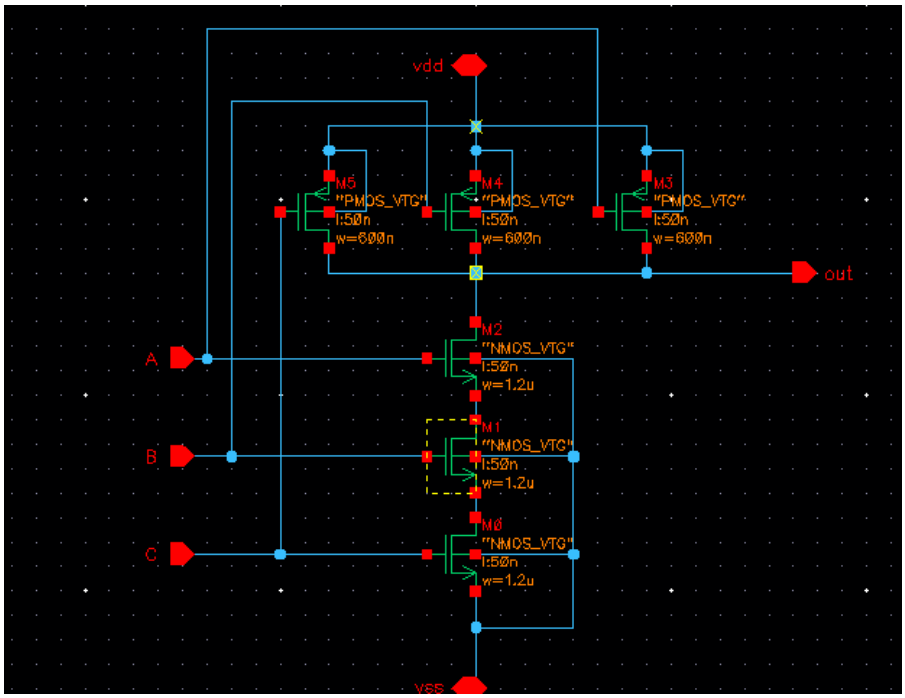


Figure 63: NAND3

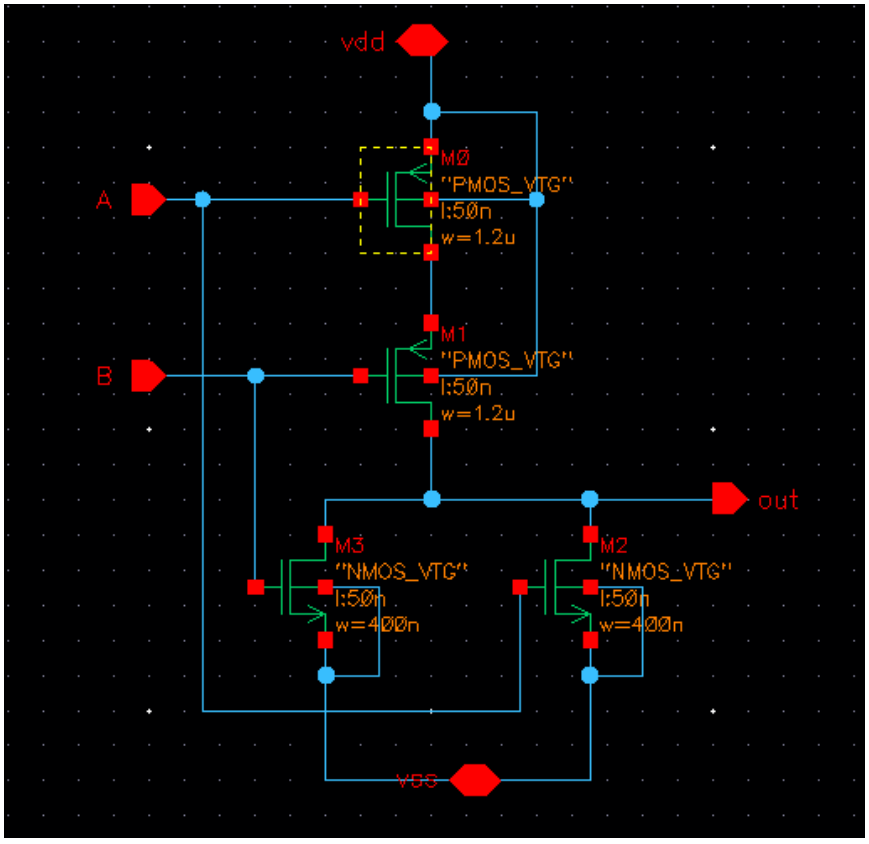


Figure 64: NOR

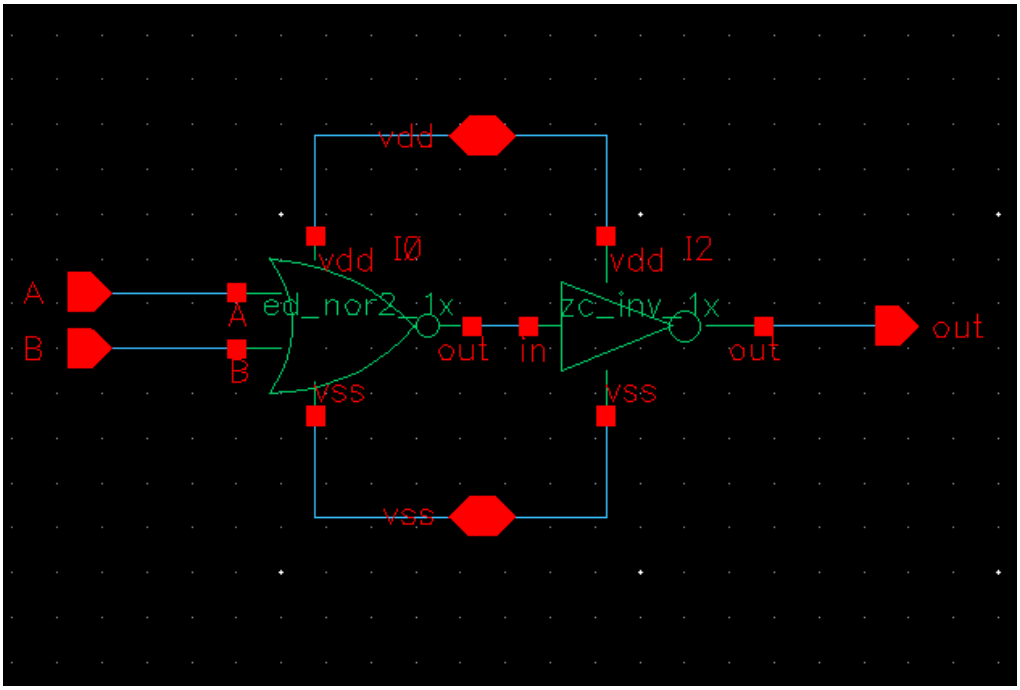


Figure 65: OR

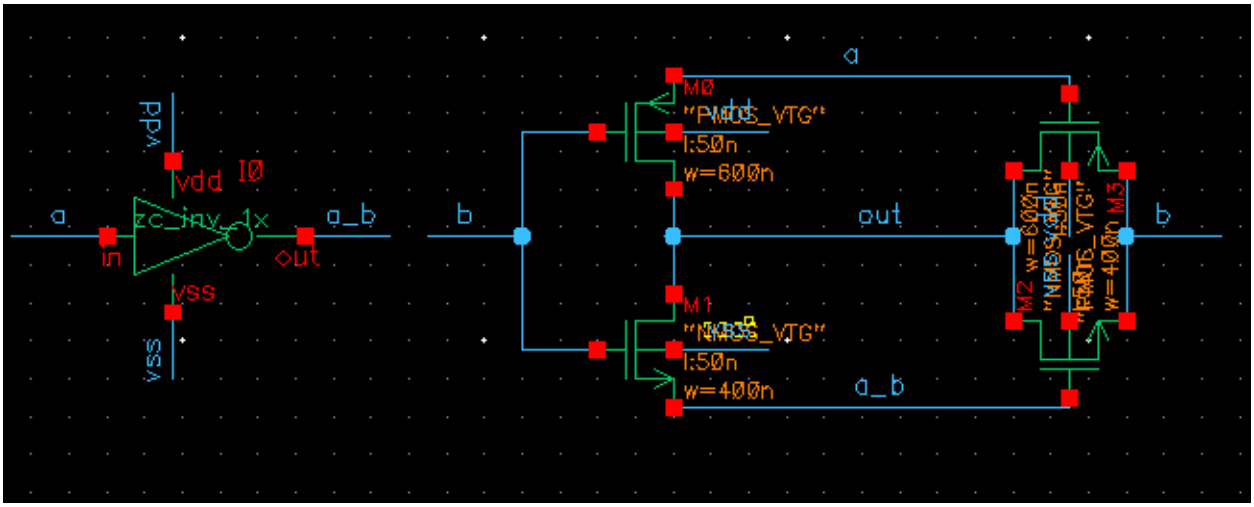


Figure 66:XOR

Appendix B of layouts

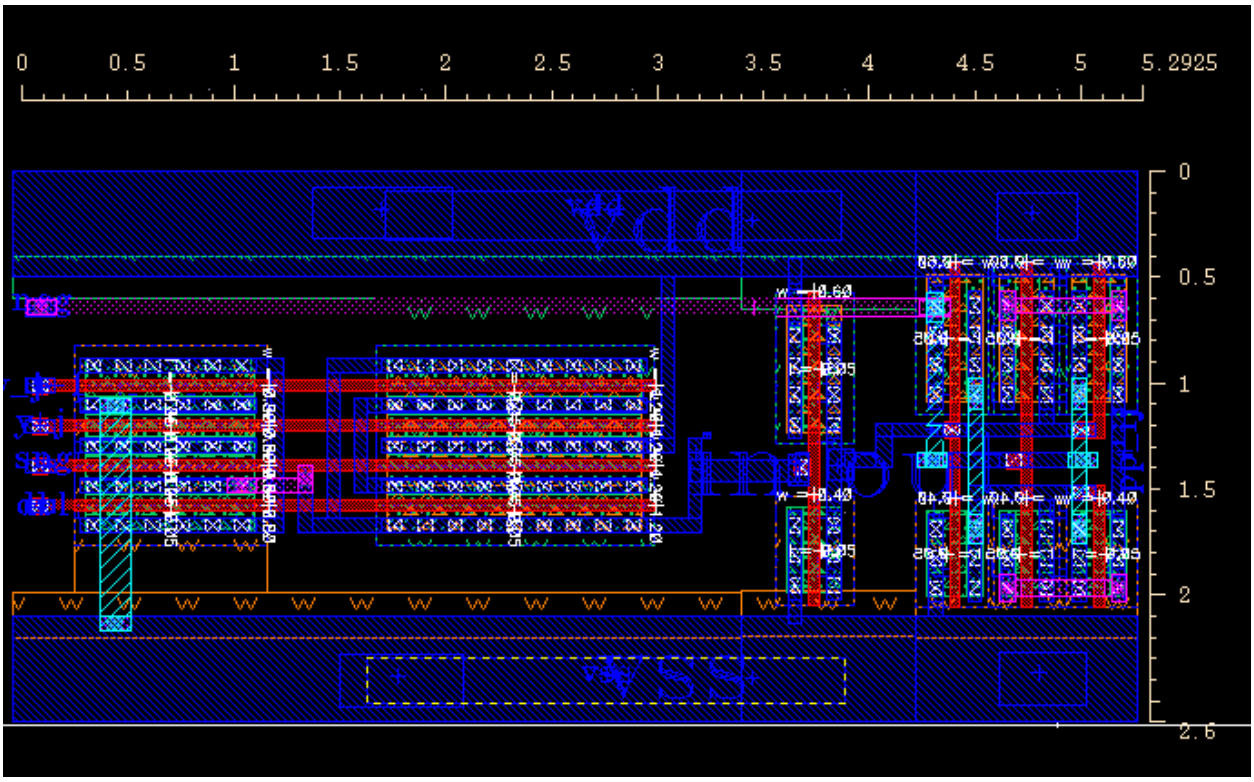


Figure 67: 1 Bit Decoder

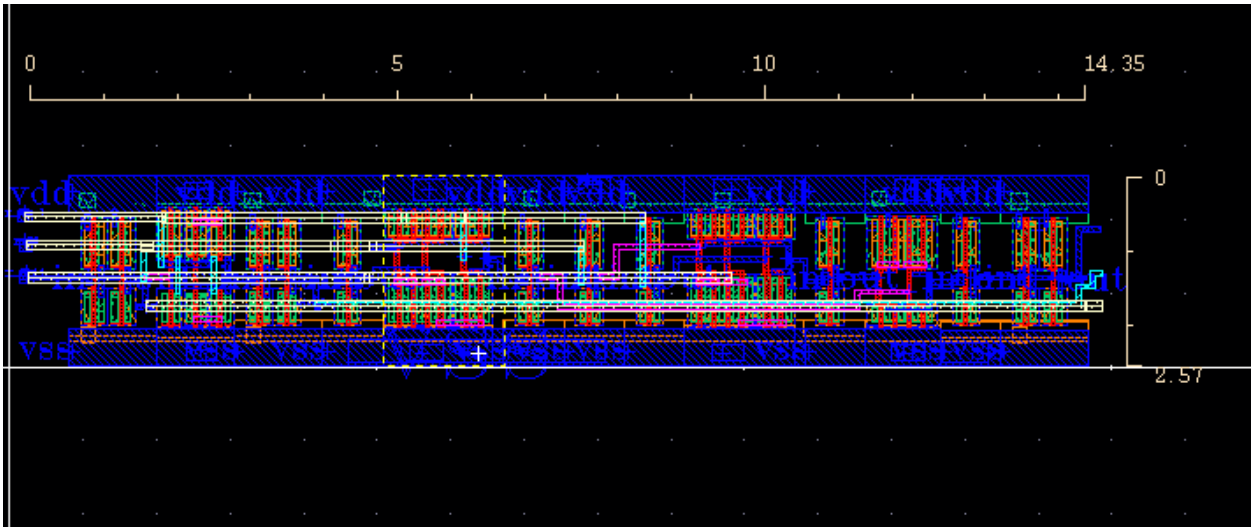


Figure 68: 3 Bit Encoder

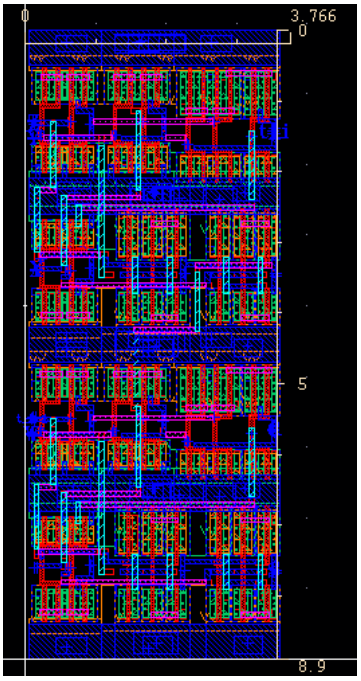


Figure 69: 4:2 Compressor

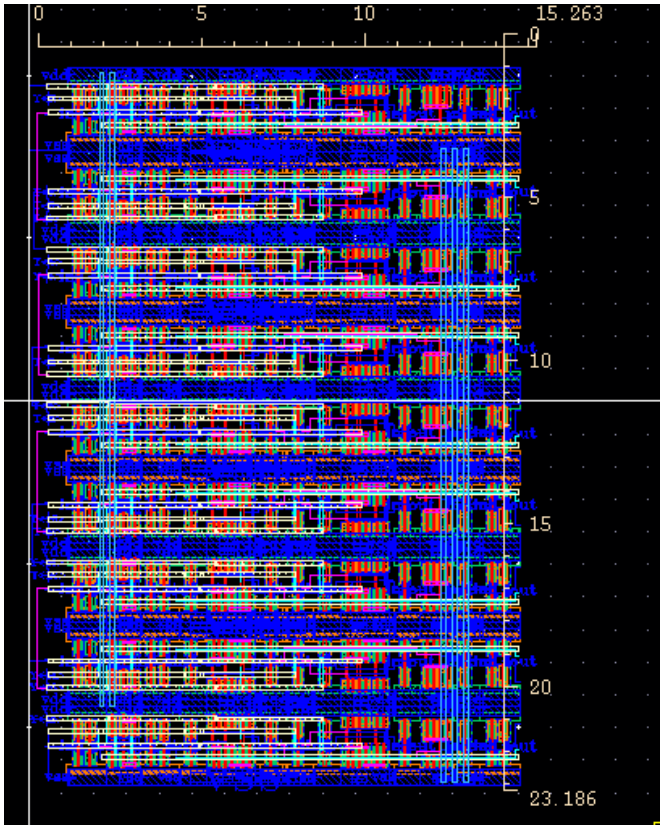


Figure 70: 9 Product Encoder

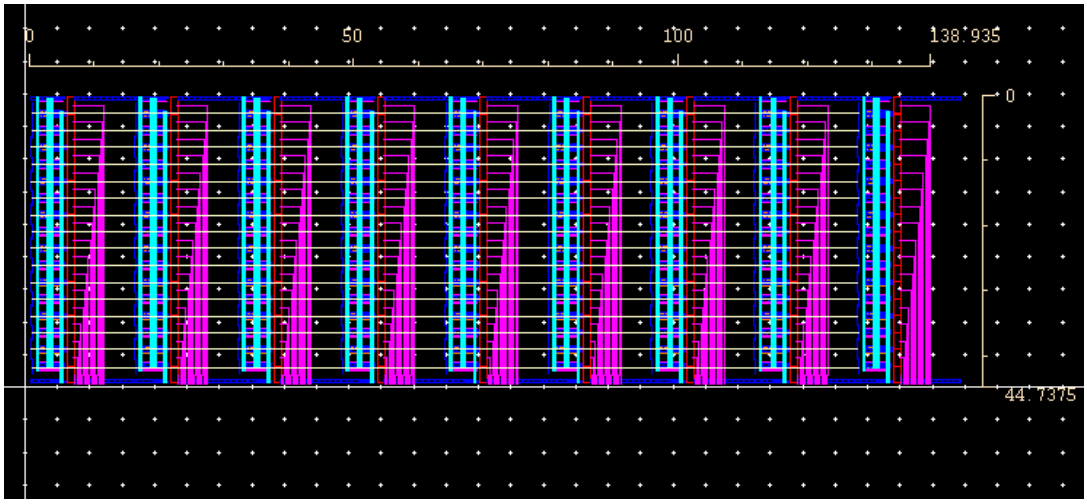


Figure 71: 9 Product Decoder

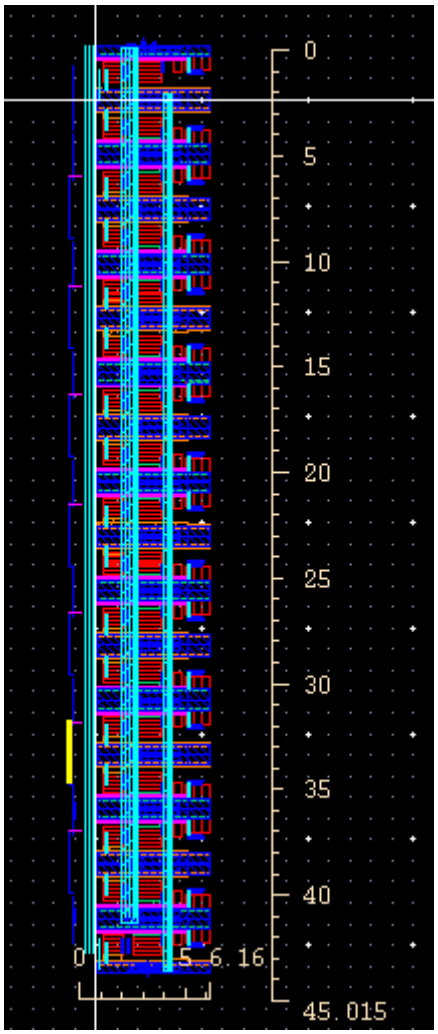


Figure 72: 17 Bit Decoder

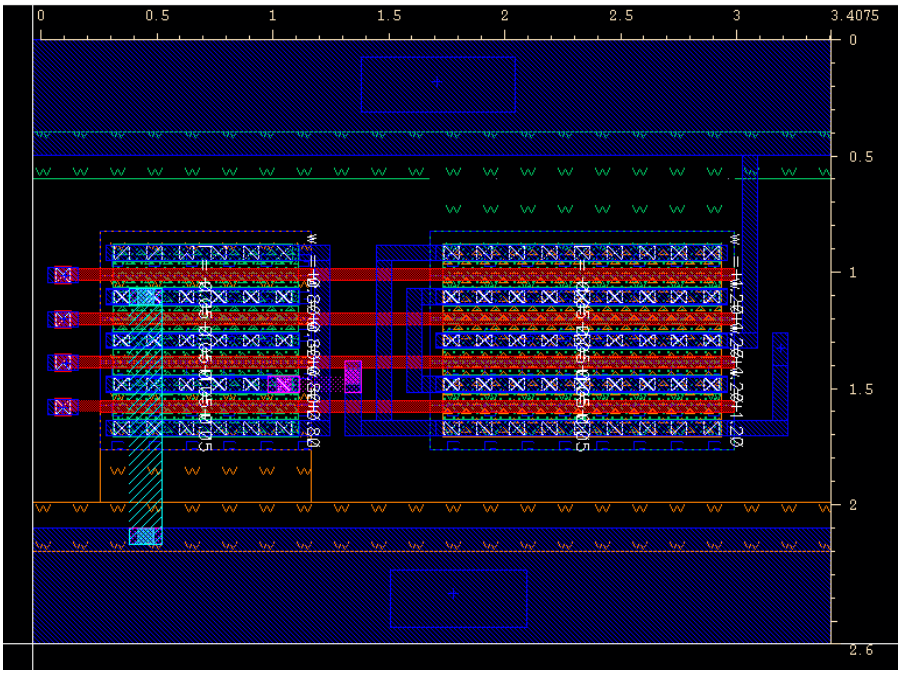


Figure 73: AOI22

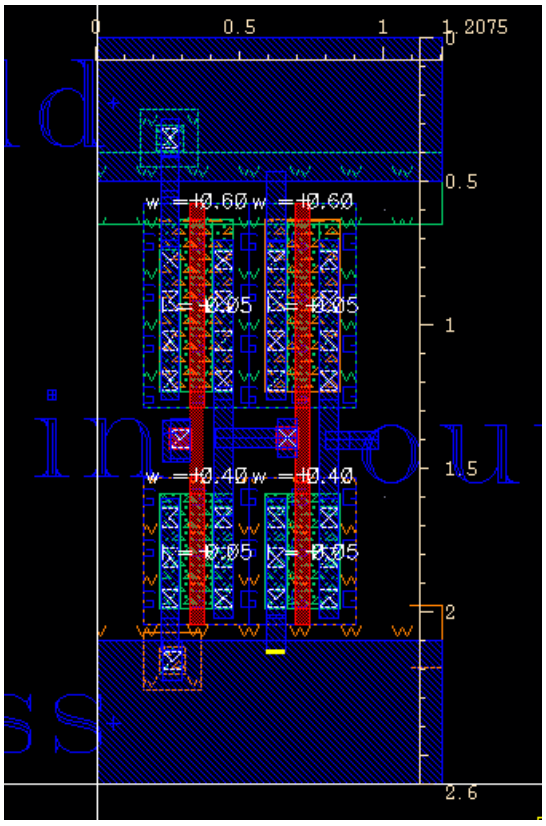


Figure 74: Buffer

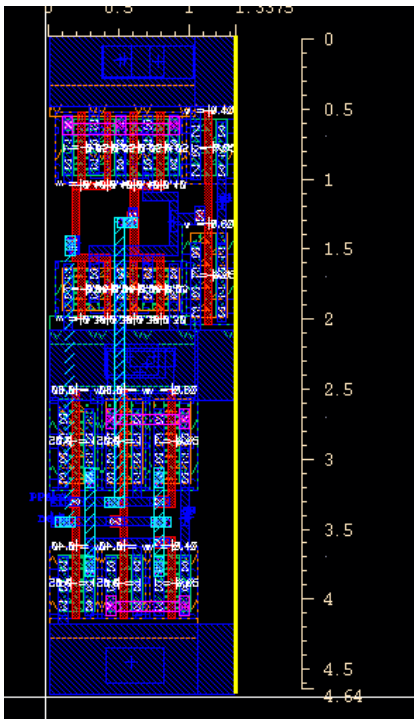


Figure 75: Column 0

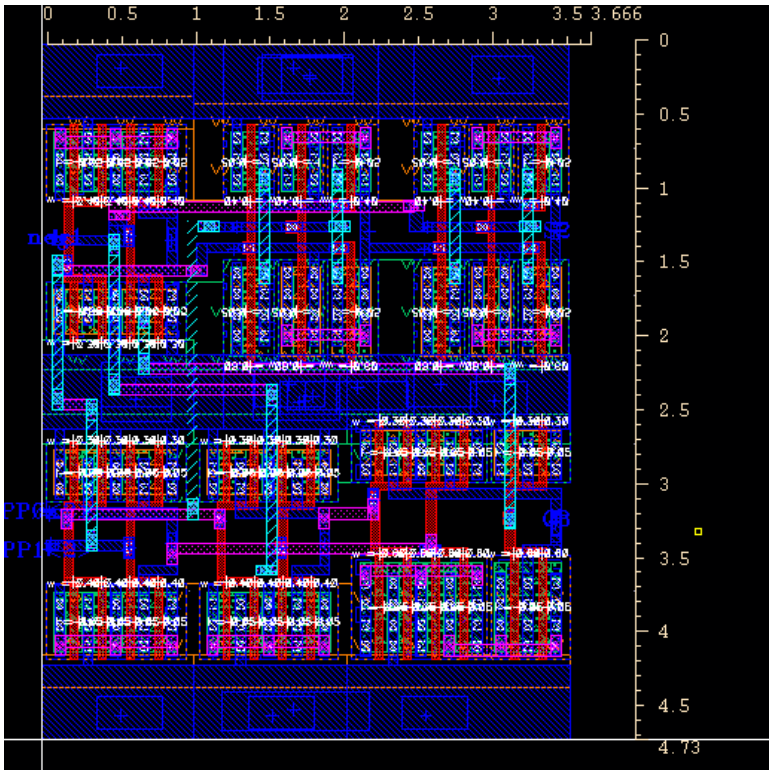


Figure 76: Column 2

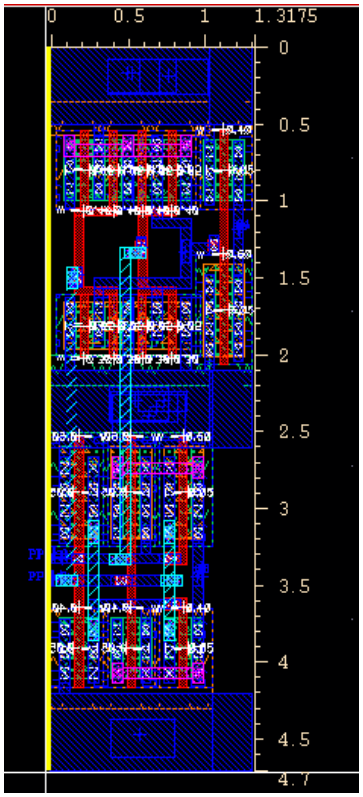


Figure 77: Column 3

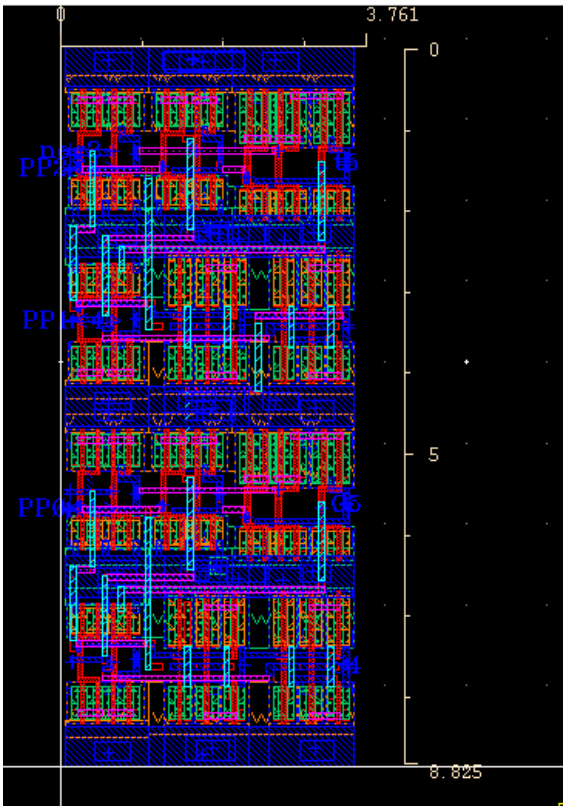


Figure 78: Column 4

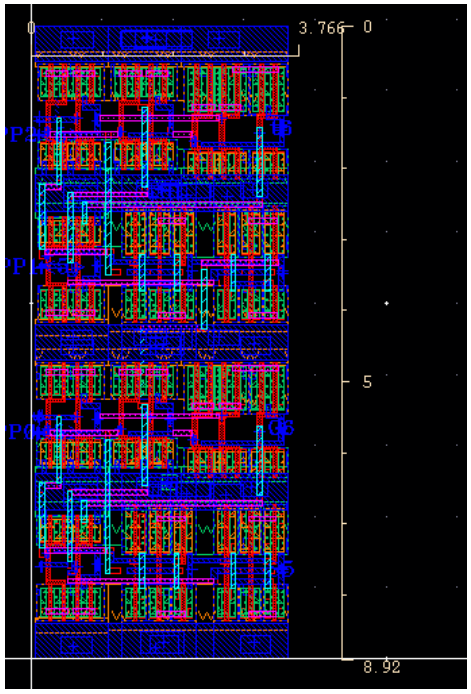


Figure 79: Column 5

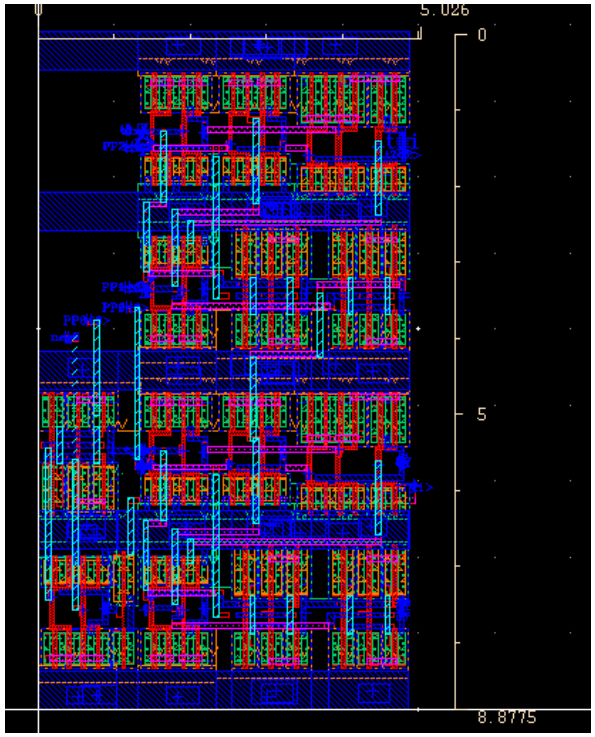


Figure 80: Column 6

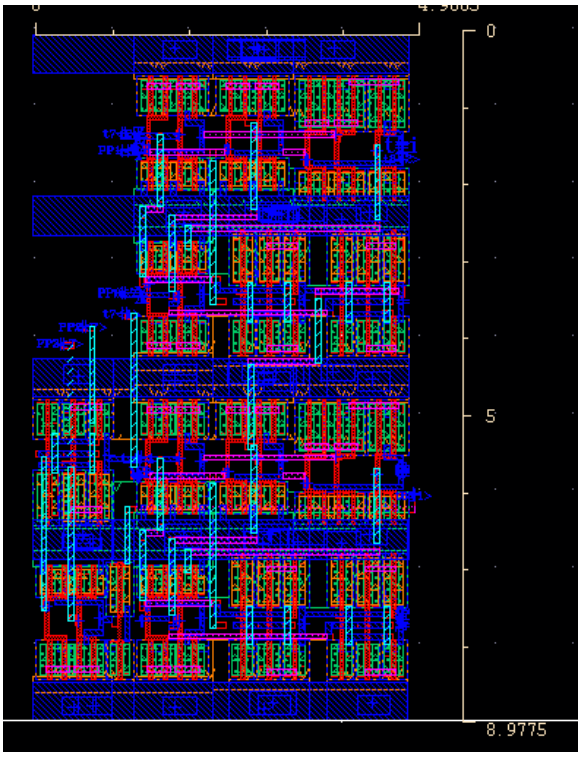


Figure 81: Column 7

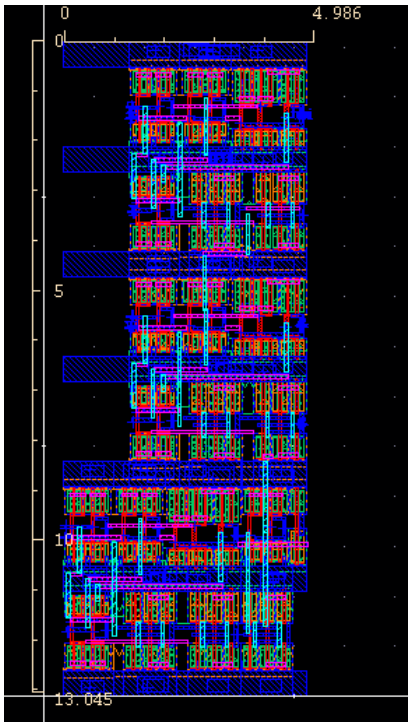


Figure 82: Column 8

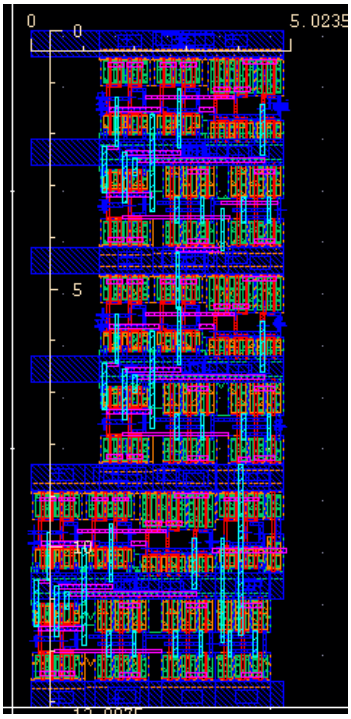


Figure 83: Column 9

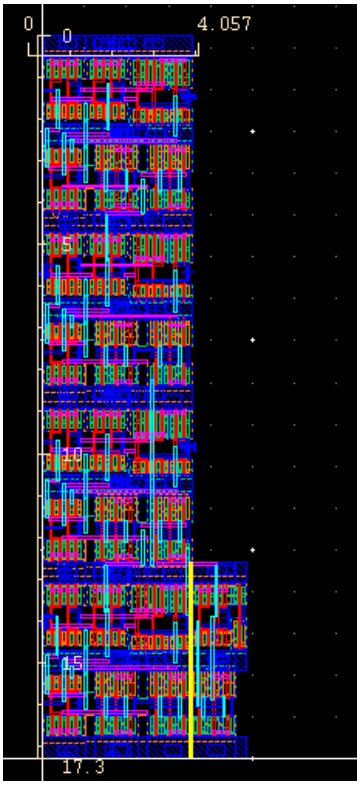


Figure 84: Column 10

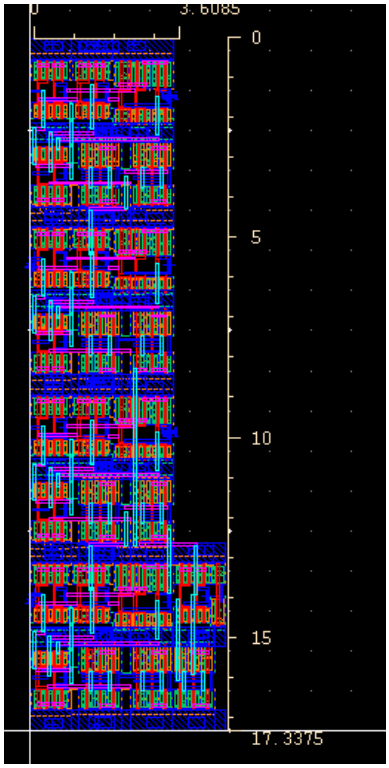


Figure 85: Column 11

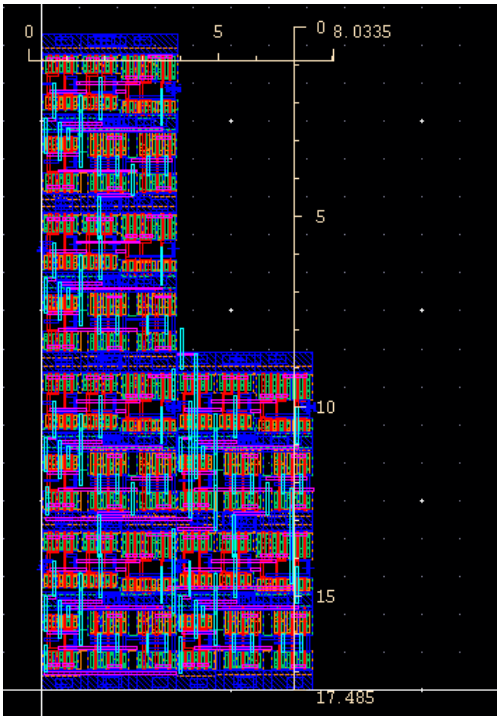


Figure 86: Column 12

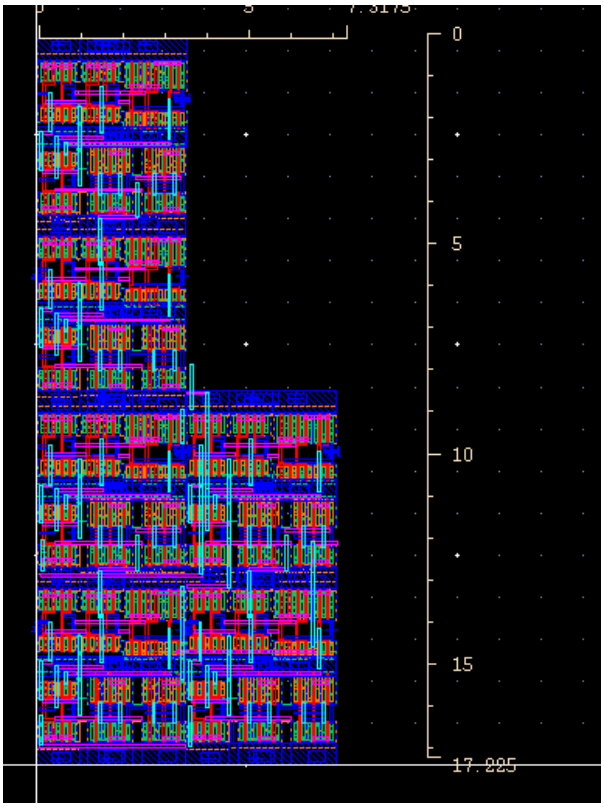


Figure 87: Column 13

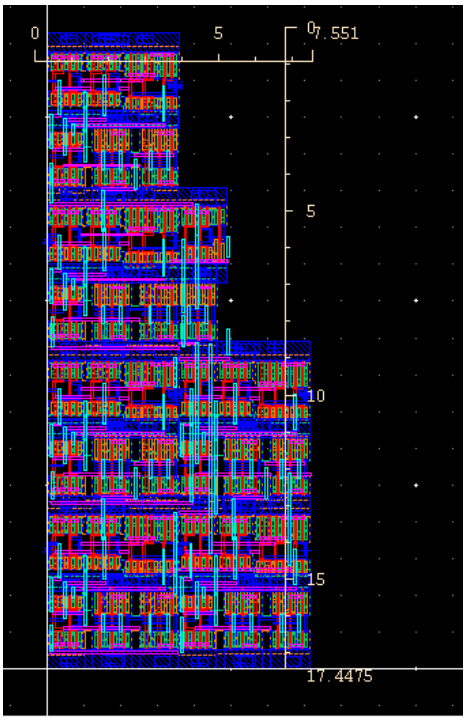


Figure 88: Column 14

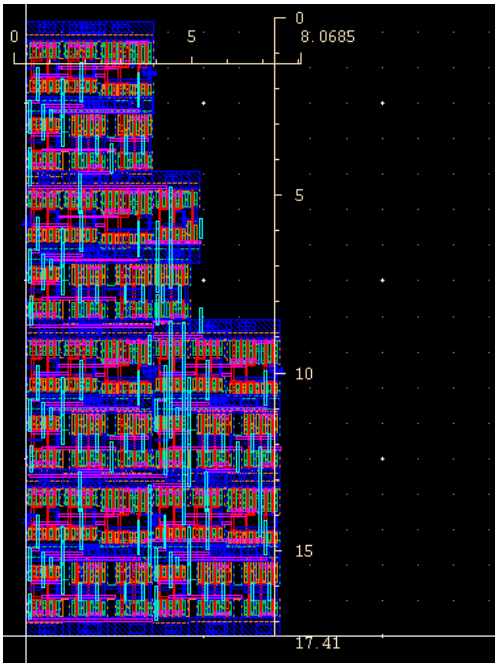


Figure 89: Column 15

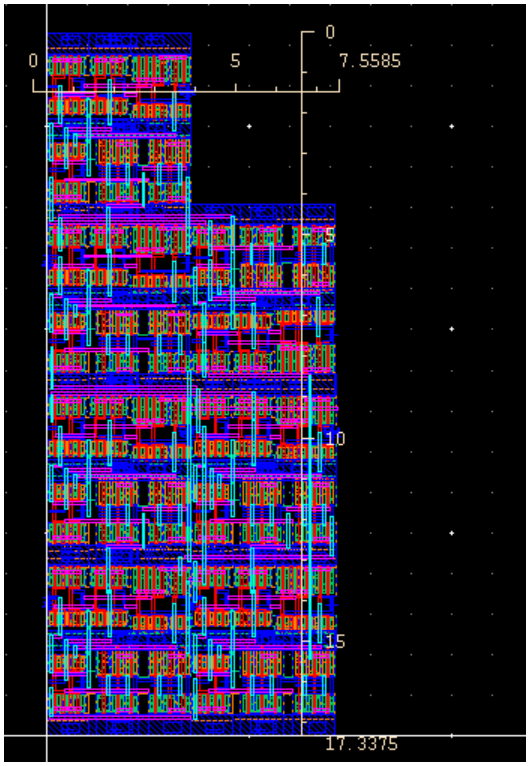


Figure 90: Column 16

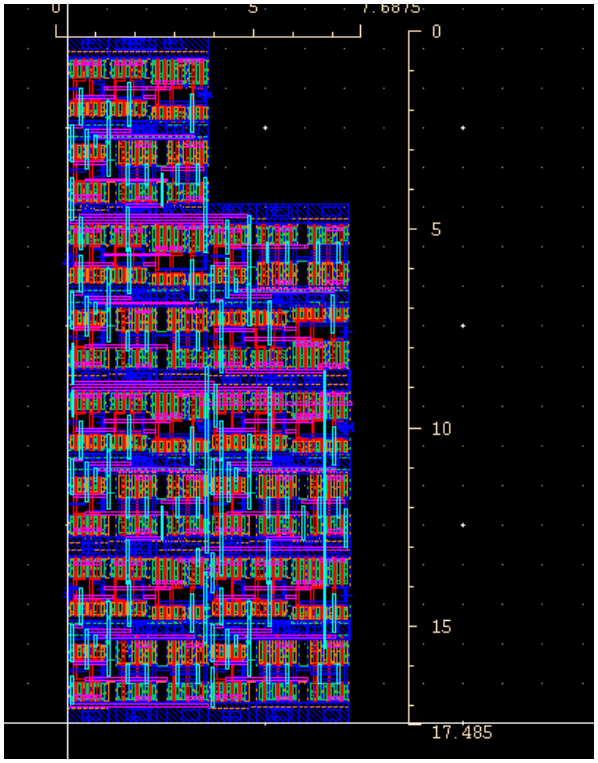


Figure 91: Column 17

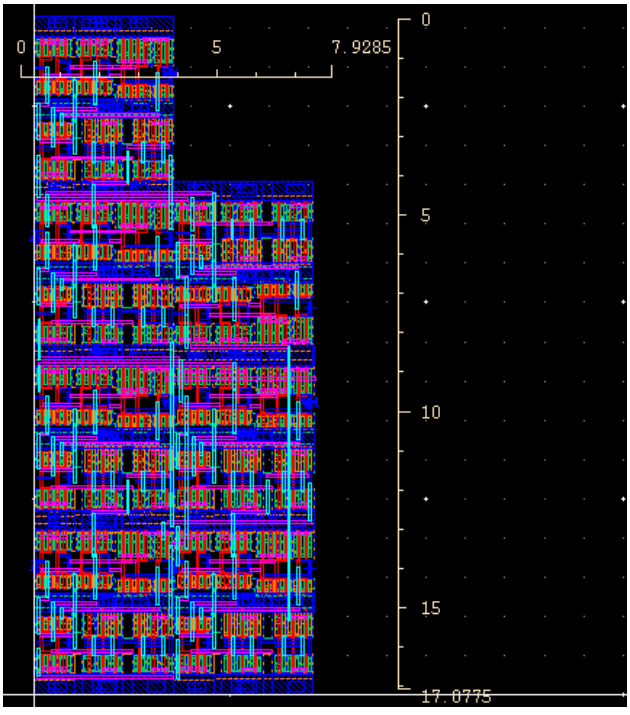


Figure 92: Column 18

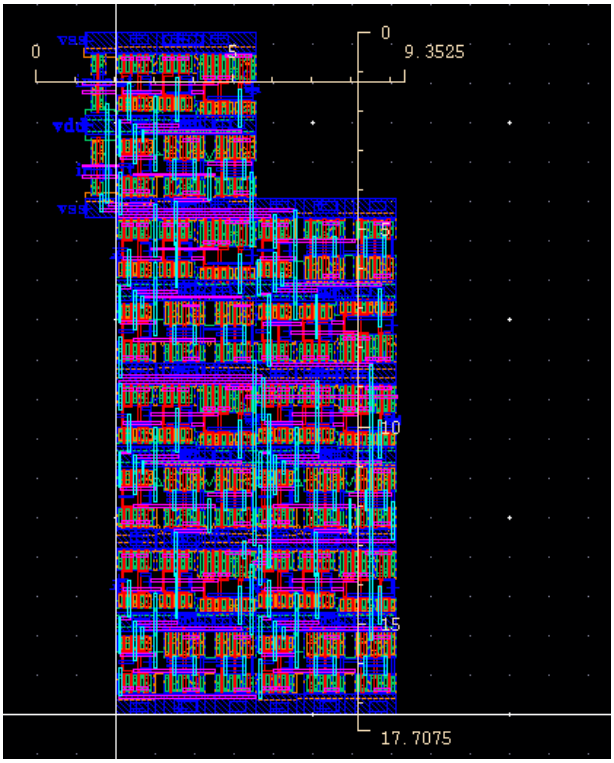


Figure 93: Column 19

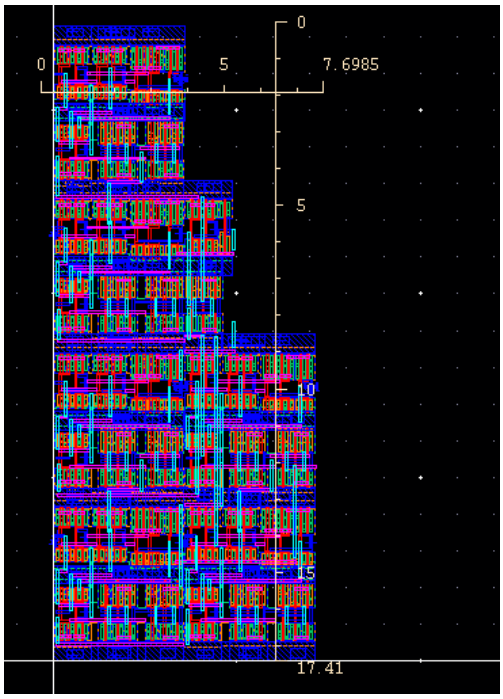


Figure 94: Column 20

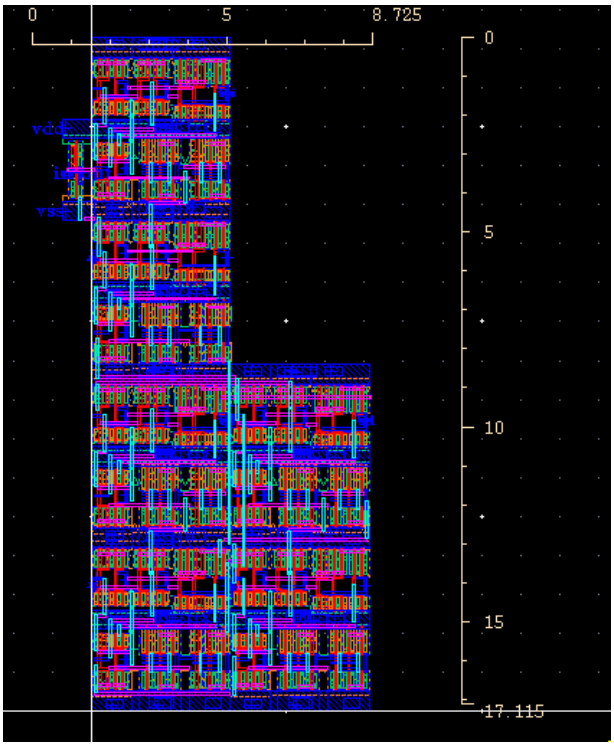


Figure 95: Column 21

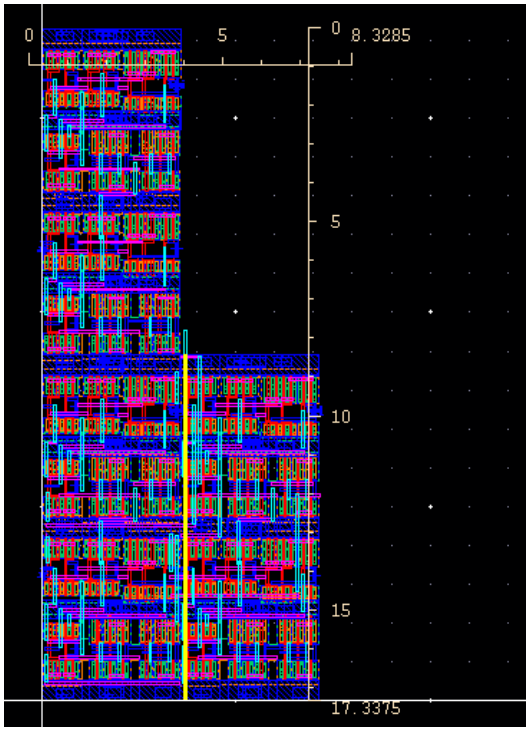


Figure 96: Column 22

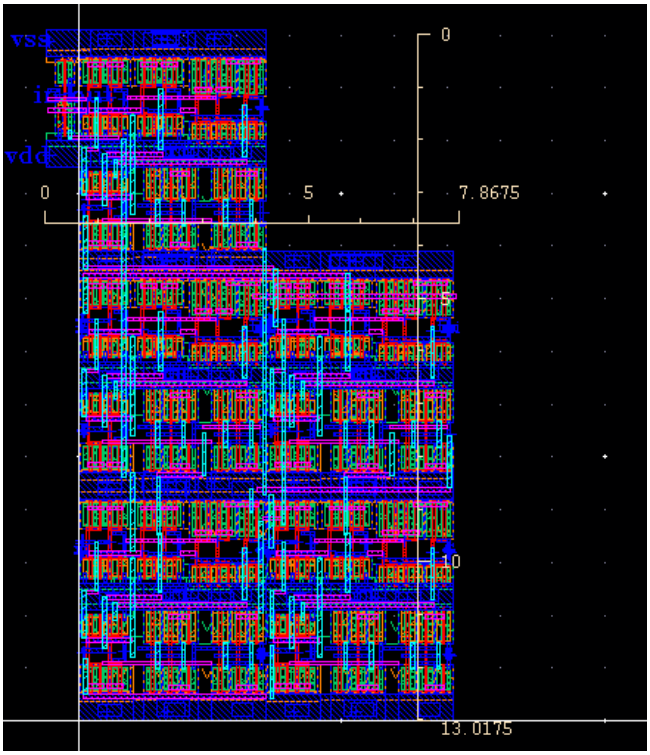


Figure 97: Column 23

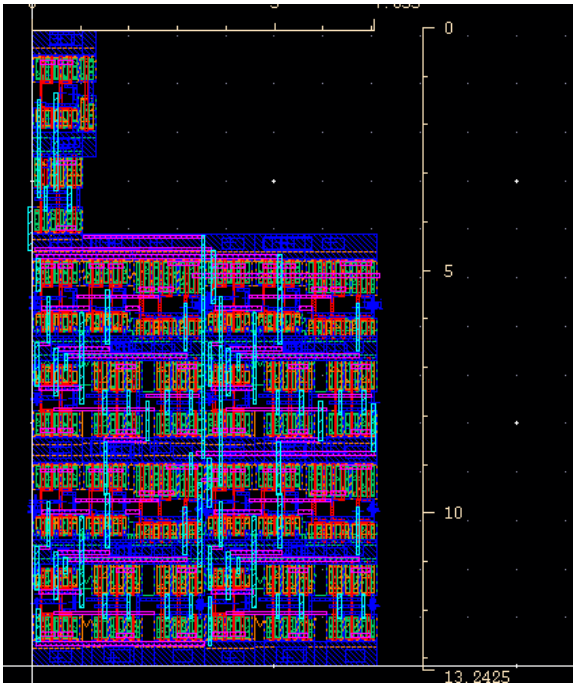


Figure 98: Column 24

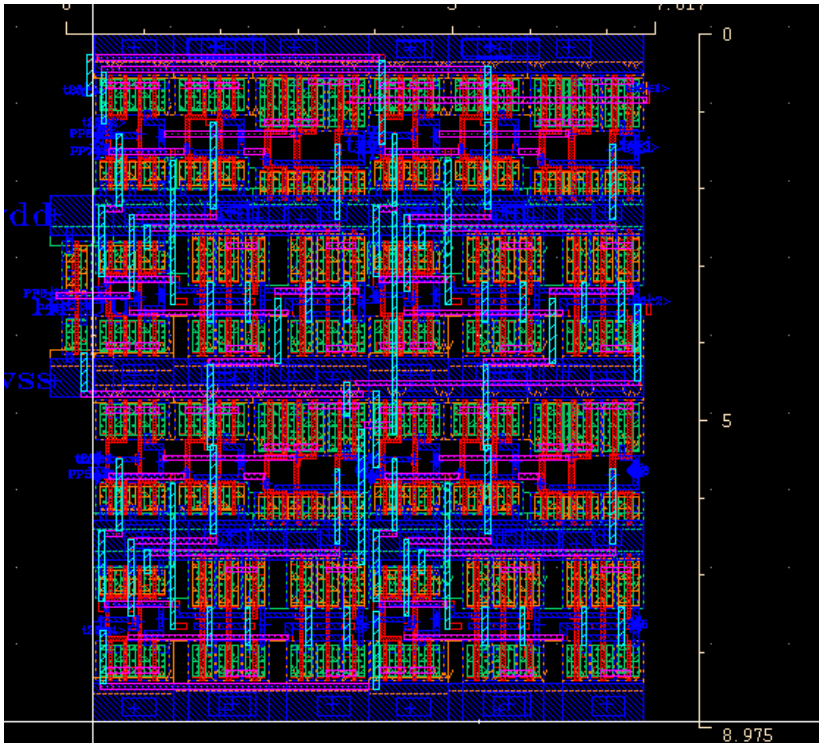


Figure 99: Column 25

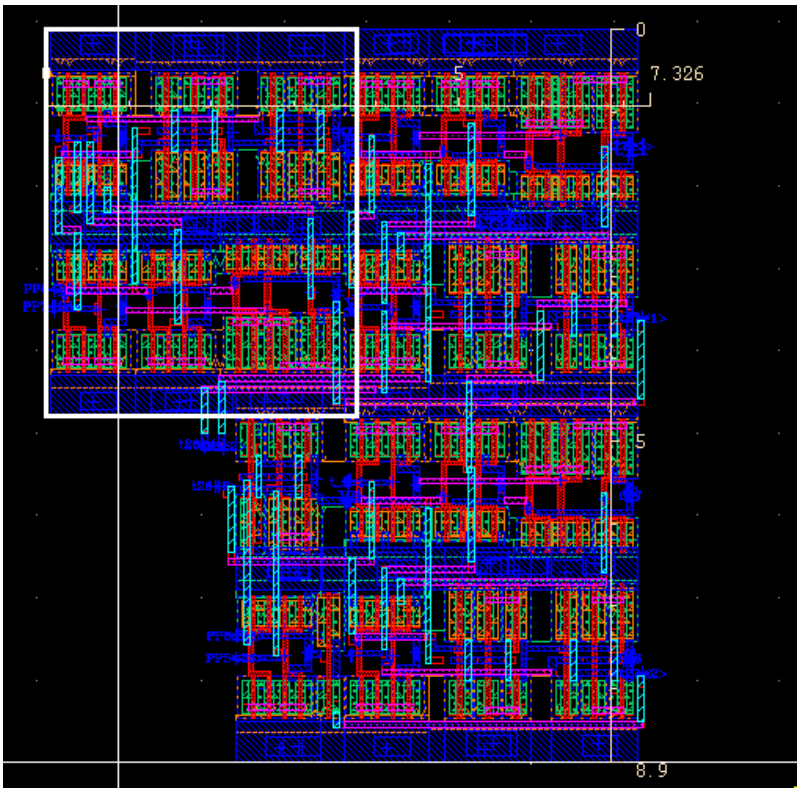


Figure 100: Column 26

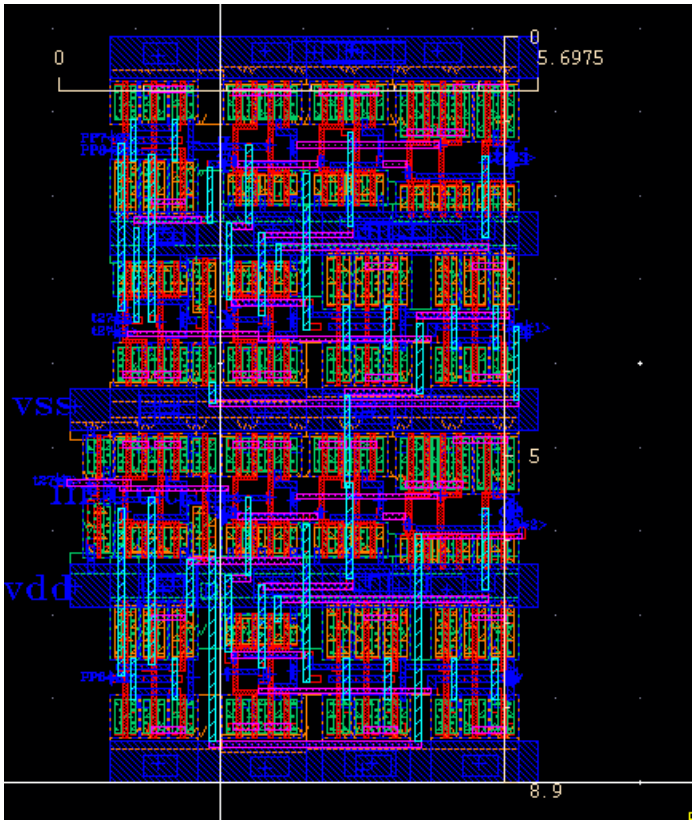


Figure 101: Column 27

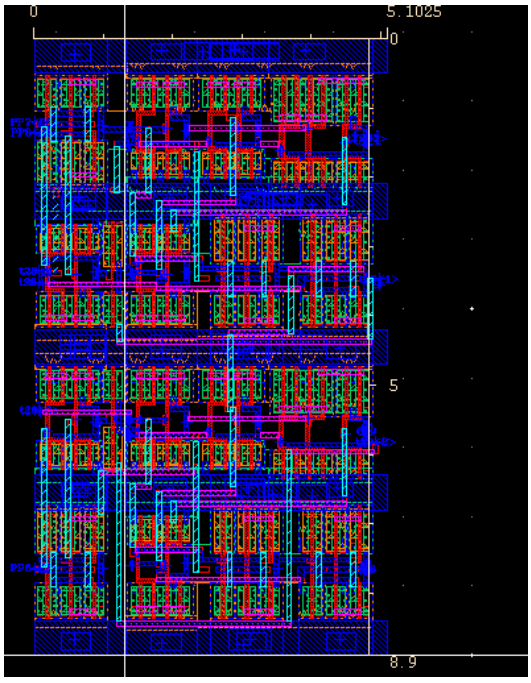


Figure 102: Column 28

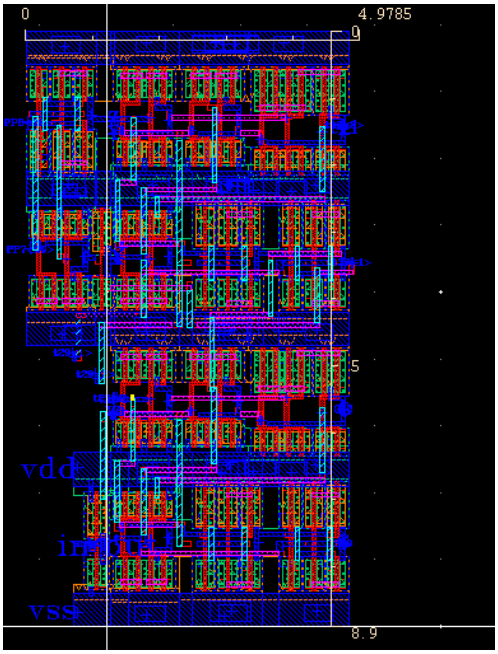


Figure 103: Column 29

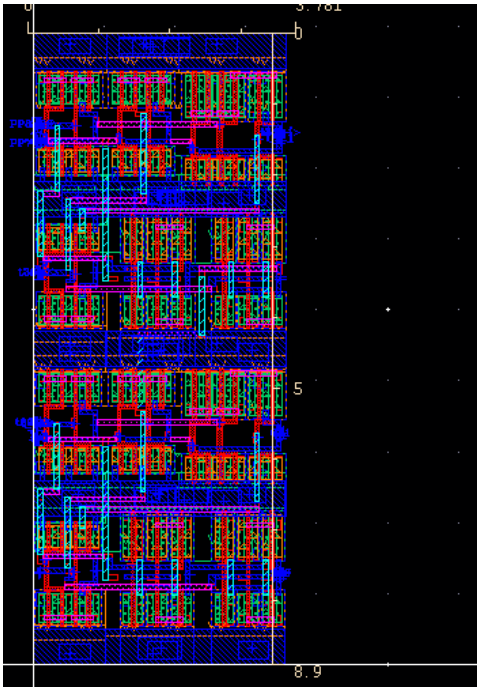


Figure 104: Column 30

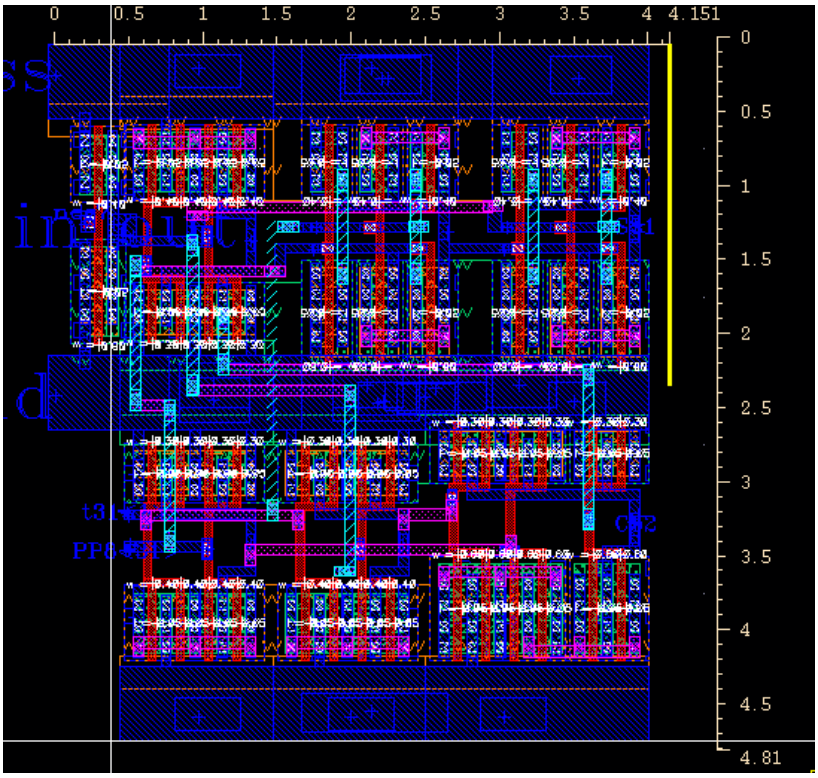


Figure 105: Column 31

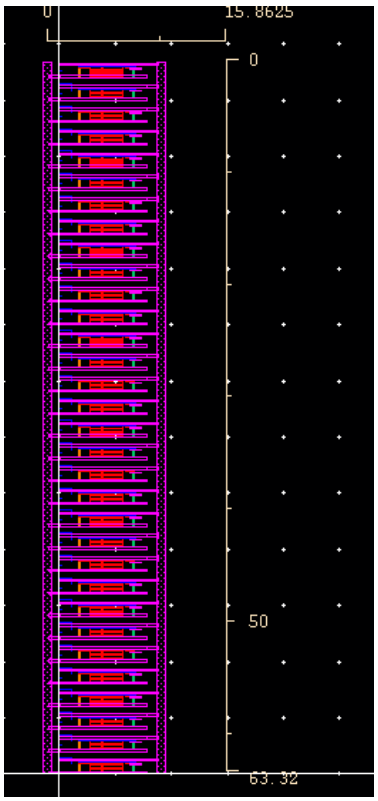


Figure 106: 32 Bit Carry Ripple Adder

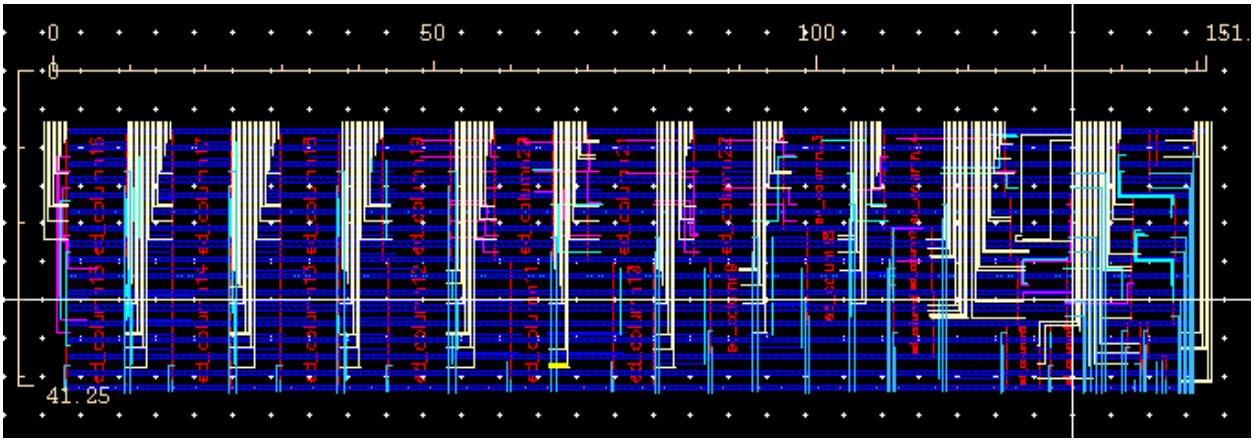


Figure 107: Column Addition

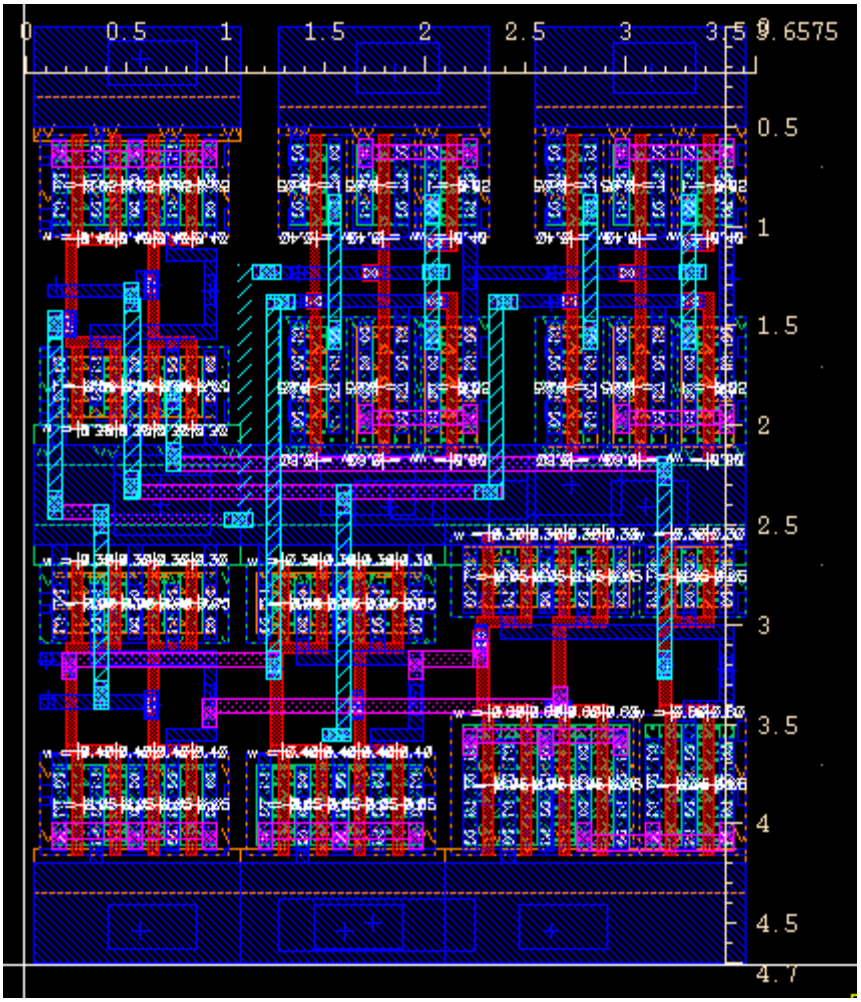


Figure 108: Carry Save Adder

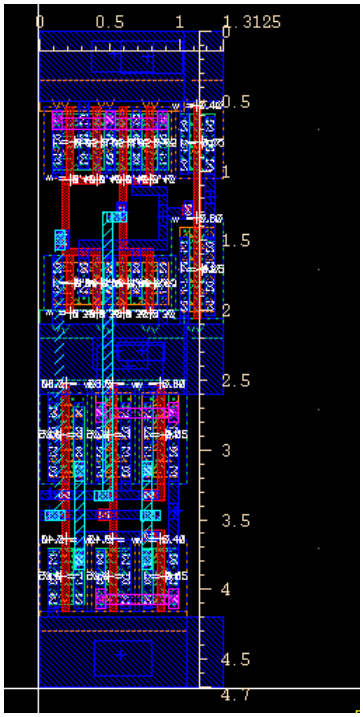


Figure 109: Half Adder

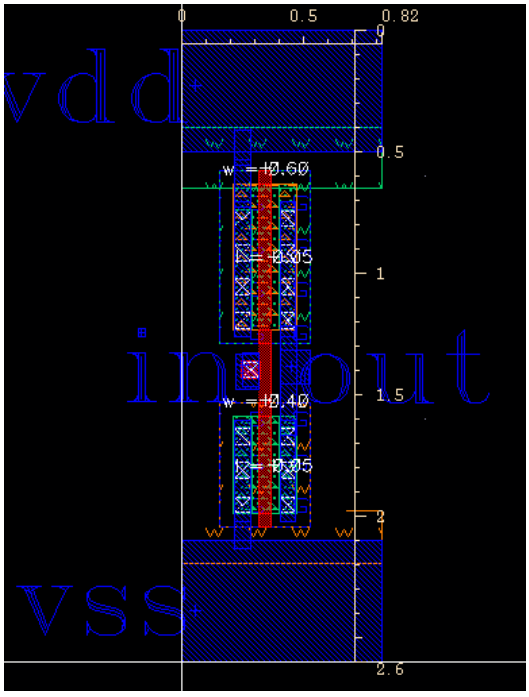


Figure 110: Inverter

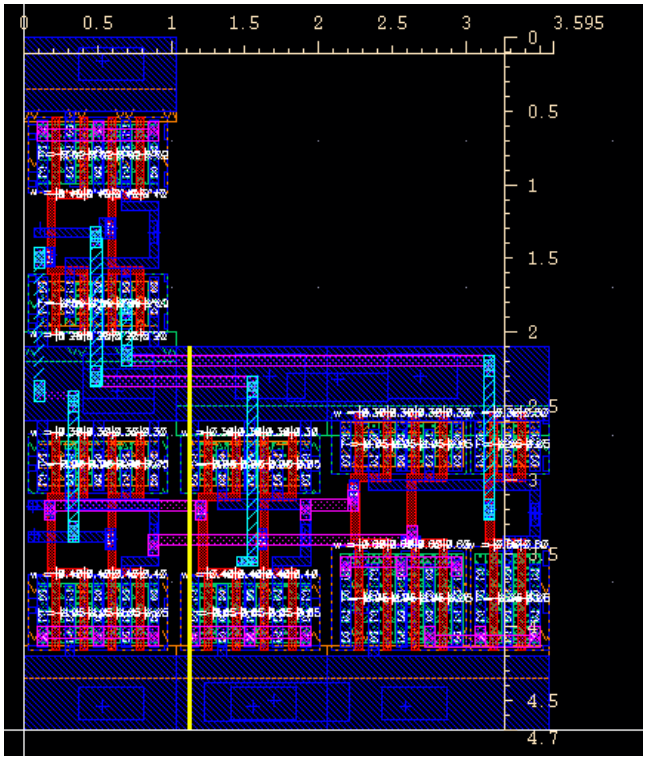


Figure 111: Majority

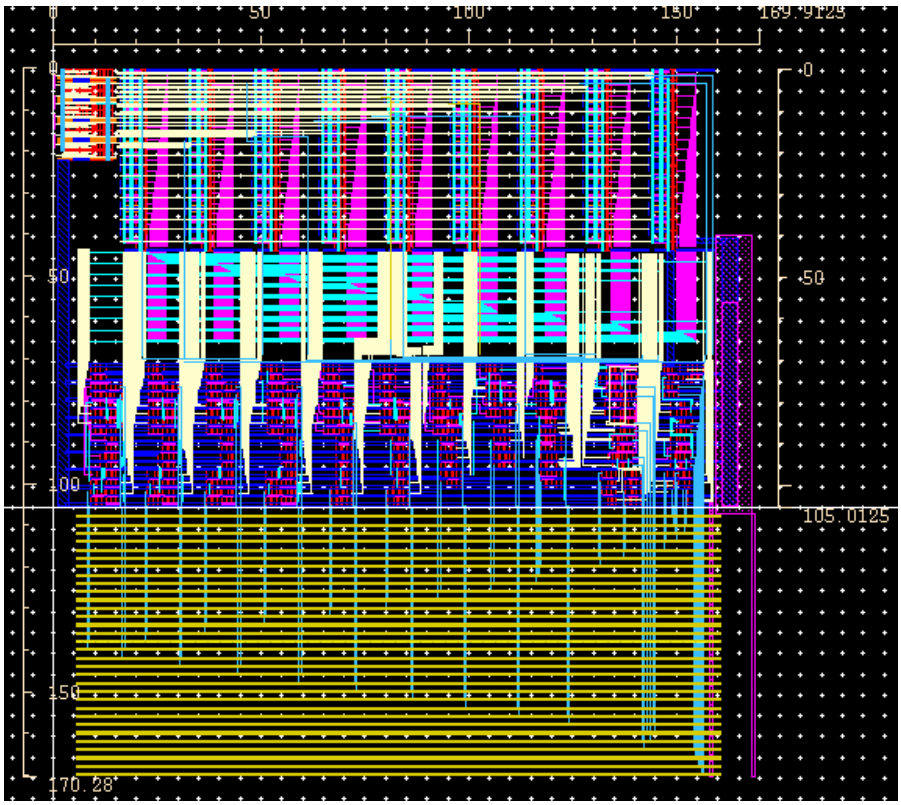


Figure 112: Top Level Multiplier

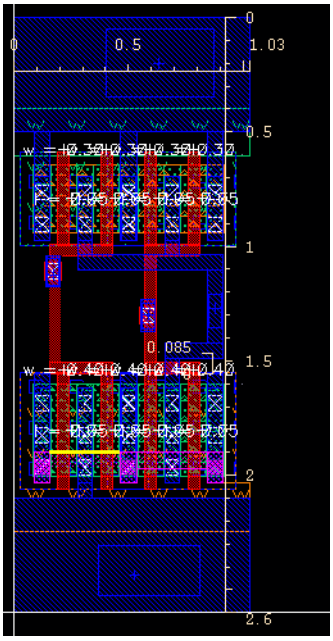


Figure 113: NAND2

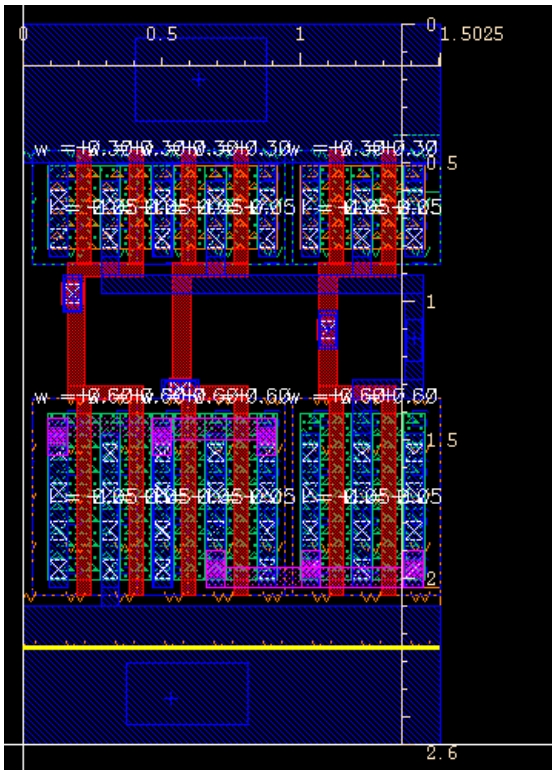


Figure 114: NAND3

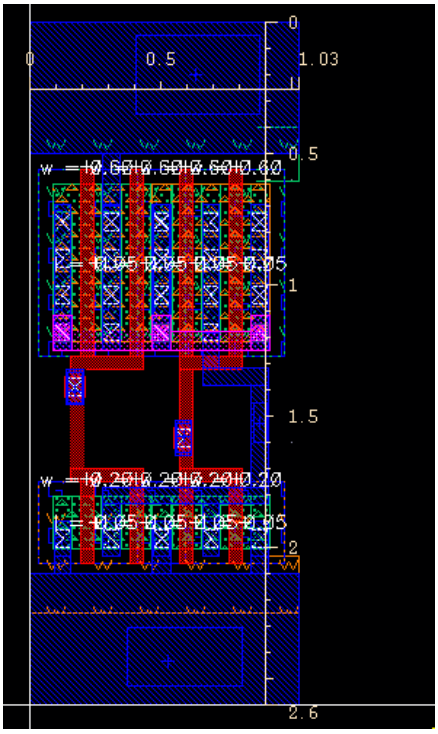


Figure 115: NOR

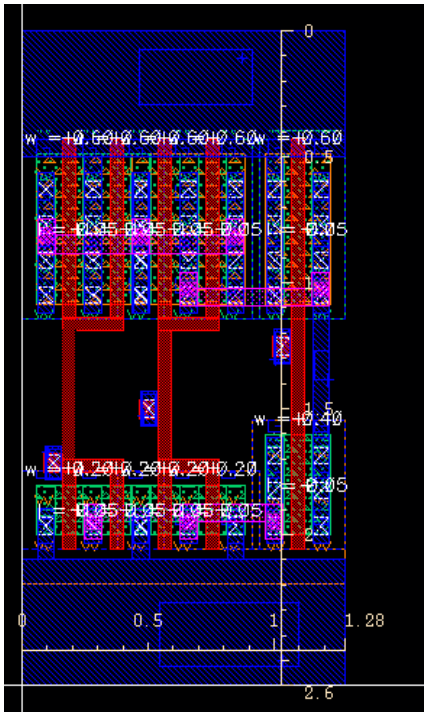


Figure 116: OAI

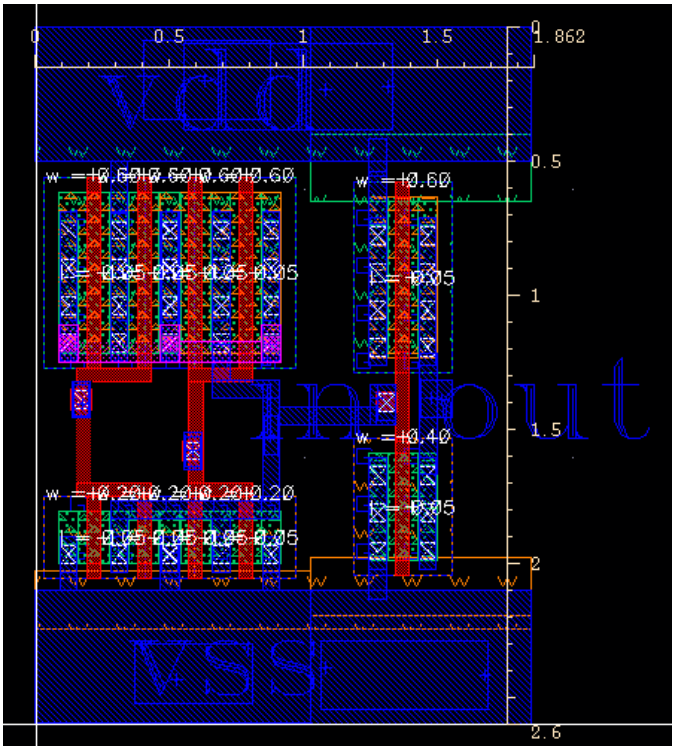


Figure 117: OR

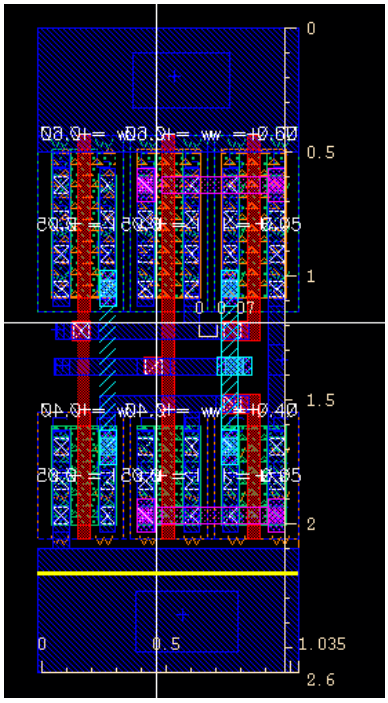
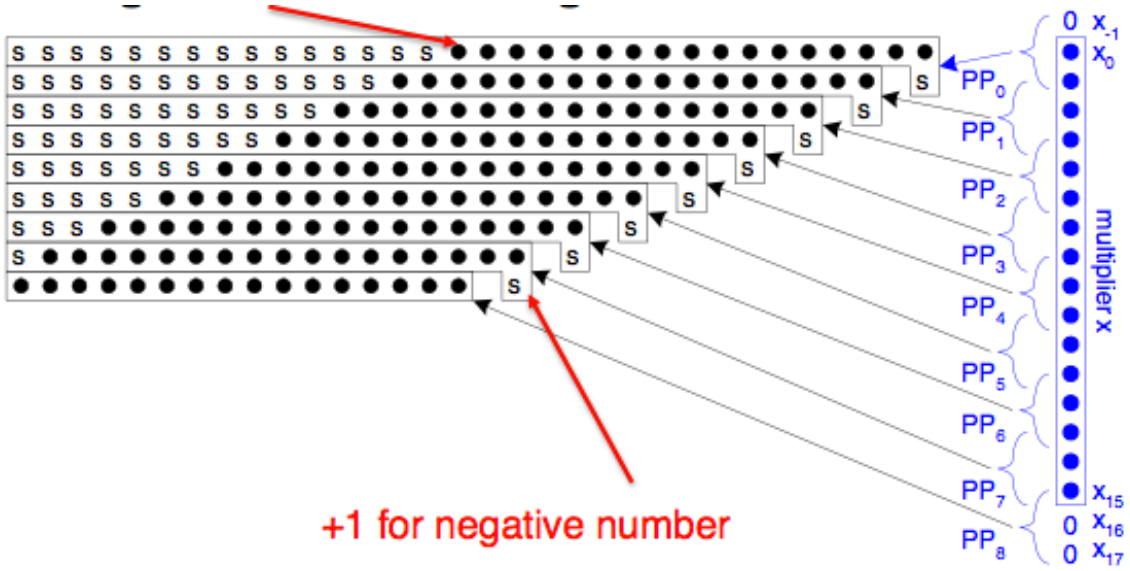


Figure 118: XOR

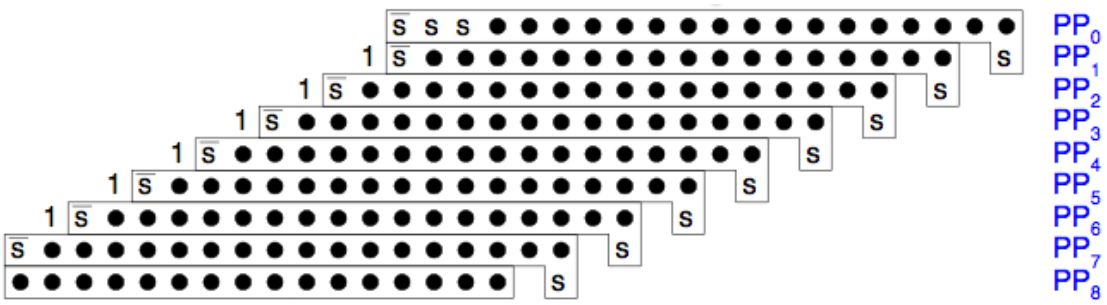
Appendix C of sign extension

A Booth Multiplier has signed partial products shown below with sign extension to the remaining columns.



The fanout of each sign bit is massive and would greatly add to parasitic capacitance in the layout version of the multiplier. Knowing that each S for a given partial product is either a 0 or a 1, this format can be used by making every S a 1 and have a single S carry bit that is 0 if the partial product is negative, and 1 if it is positive.

However, the sign extension can be further reduced to minimize routing of vdd into all of the column additions as shown below.



This is the final form used in the multiplier, and while the fanout of each sign bit is 2, except for partial product #0 with a fanout of 4, the balance between minimal routing and fanout is achieved. In addition, it reduces the number of XOR levels within the 4:2 compressors and CSAs for the column additions.