5LIH0: Project

Quinten Kustermans
q.kustermans@student.tue.nl, 1439561

Stavros Spyridopoulos
s.spyridopoulos@student.tue.nl, 2234254

1 Introduction

This project report summarizes our findings on the project for the course Digital Integrated Circuit Design. The goal of this project was to get acquainted with using Cadence Virtuoso and Calibre to create, verify and simulate complex transistor design schematics and layouts. We will explain our process of realizing the schematics and layouts, and present our methodology in doing this, as well as in the verifying and testing of these designs. We will lastly discuss the results yielded from these tests, and reflect on our process.

2 Project Description

2.1 Assignment

During this project, we create and simulate 2 distinct 8-bit Brent-Kung adders in Cadence Virtuoso, the latter of which is to be an optimized revision of the first. We do so by first drawing the appropriate schematic using complementary PMOS and NMOS (CMOS) transistor logic. We then perform various tests and analyse both the functionality and performance of the schematic under a bitwise load of 40fF, and compare the computational functionality of the adders. After the schematics are finished, we create and verify corresponding layouts, again in Cadence Virtuoso, performing a DRC and LVS verification using Calibre, and performing parasitic extraction of the circuits. Finally, we perform the same analyses as for the schematics, and we compare the findings of the unoptimized and optimized adder. These steps will be explained in more detail below.

2.2 Schematic Entry

For the schematics, a 45nm technology set was used. Throughout all designs, the minimum width of an NMOS was chosen to be 90nm, PMOS widths were chosen to be twice this amount for balanced rise and fall times. None of the transistors used were body-biased. Using static CMOS structures, the following (standard) logic gates were created with which more complex cells were later constructed:

- NOT (inverter) (Fig. 1)
- NAND (Fig. 3)
- NOR (Fig. 2)
- OR (NOR followed by an inverter, created for ease of use and readability)
- AND (NAND followed by a NOT, created for ease of use and readability)

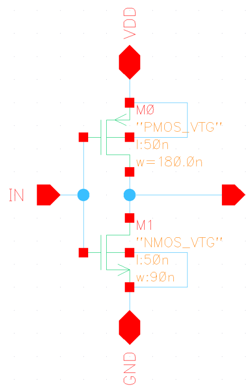


Figure 1: Inverter Schematic

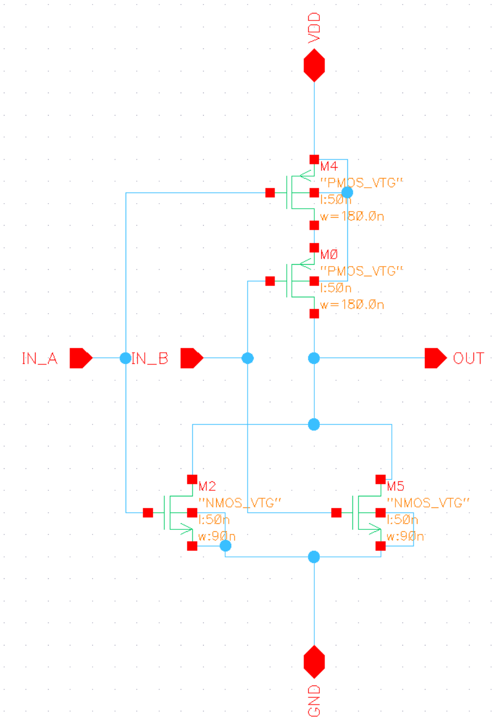


Figure 2: NOR Gate Schematic

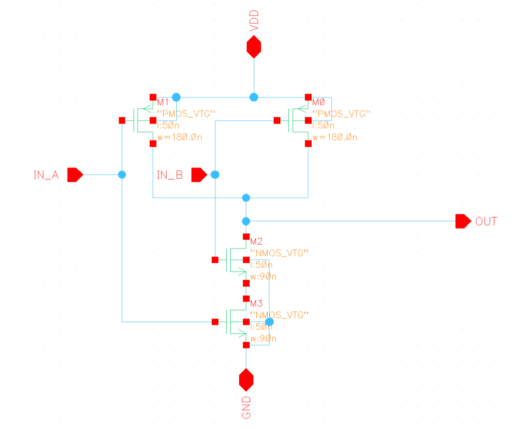


Figure 3: NAND Gate Schematic

2.2.1 Preprocessor

The preprocessor can be constructed straightforwardly using a combination of the already created AND gates, and an additional set of XOR gates, which are implemented as can be seen in (25)
For the schematic of the XOR gate, various equivalent possible logical expressions using the above gates were derived, and primarily judged on their total gate count, and secondarily on how many of these gates are inverting w.r.t the total.

Expression	Gates used	$\frac{\#\text{inverting}}{\#\text{total}}$
$(\overline{A}*B)+(\overline{B}*A)$	5 (2 * AND, 1 * OR, 2 * INV)	0.4
$(A+B)*(\overline{A}+\overline{B})$	5 (2 * OR, 1 * AND, 2 * INV)	0.4
$\overline{(\overline{A}*\overline{B})}*\overline{(A*B)}$	5 (2 * NAND, 1 * AND, 2 * INV)	0.8
$\overline{\overline{(A+B)}+(A*B)}$	3 (2 * NOR, 1 * AND)	0.67

Table 1: Various suggested implementations of an XOR gate with inputs A and B

There are more combinations possible for this if one continues the derivation, which is also how these formulas were realized. We were, however, satisfied with an implementation that uses 3 gates (2 of which are inverting).

2.2.2 Unoptimized Cell Design

To create the gray, black, and buffer cells that are used to structure the Brent Kung Adder, the simple schematics provided in the Project Manual were initially used. The result of this implementation was an unoptimized schematic for each cell using many inverters to simplify logic.

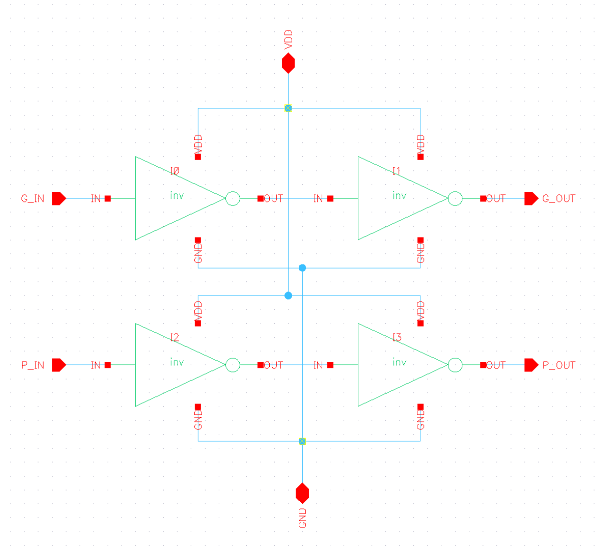


Figure 4: Schematic of Buffer Cell

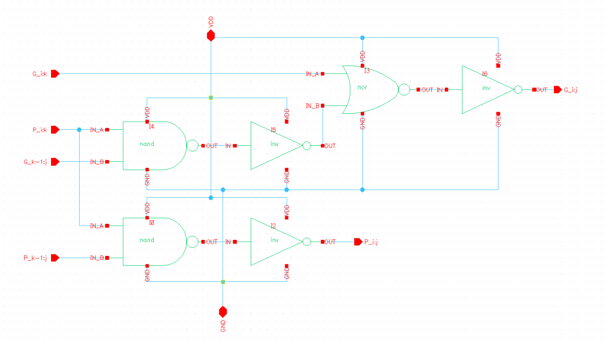


Figure 5: Schematic of Black Cell

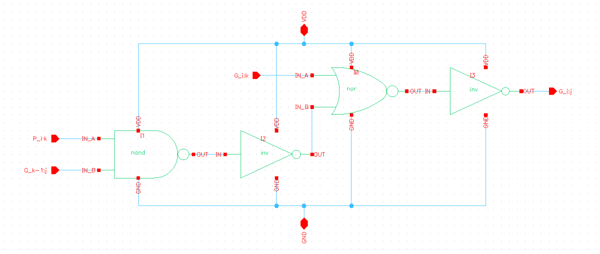


Figure 6: Schematic of Gray Cell

With NAND and NOR gates that both consist of 2 NMOS and 2 PMOS, the current transistor count of these cells is:

Cell	# of NAND Gates	# of NOR Gates	# of Inverters	# of Transistors
Buffer	0	0	4	8
Black	2	1	3	18
Gray	1	1	2	12

Table 2: Logic gate and transistor count per cell

2.2.3 Core adder, unoptimized

The Brent Kung Adder was constructed using the 8-bit top-level schematic provided in the Project Manual.

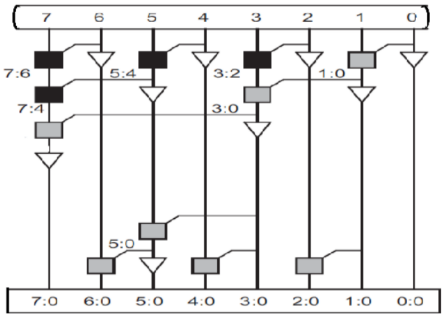


Figure 7: Top-Level Brent-Kung Schematic

Using the cells listed in 2.2.2), we constructed the schematic for the initial Brent-Kung Adder where every cell's initial letter is drawn on top of it:

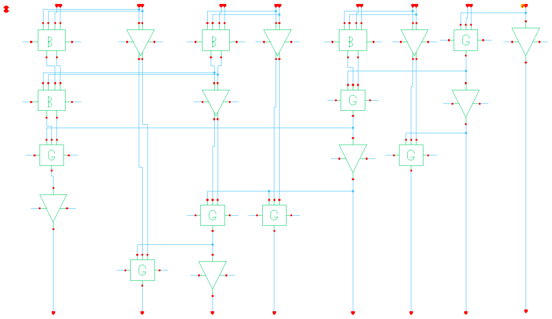


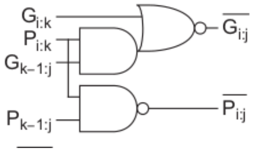
Figure 8: Unoptimized Adder Schematic

# of Gray Cells	# of Black Cells	# of Buffer Cells	# of Transistors
7	4	9	228

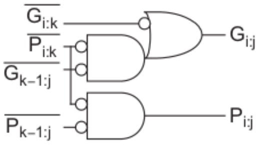
Table 3: Cell and Transistor count of Unoptimized Adder

2.2.4 Chosen optimizations

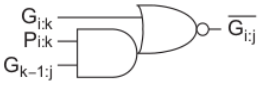
To achieve optimization of the schematic, inspiration was taken from [1] where an alternative cell structure was proposed to reduce transistor count and propagation delay. In this optimized structure an introduction of **odd** and **even** cells was proposed with their corresponding schematics found in the pictures bellow. This structure aligns with the *Compound Gates* technique used in VLSI. This way, there are no extra inversions from row to row so more that 2 inverters are eliminated on each row of the adder.



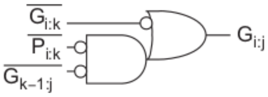
(a) Odd Row Black Cell



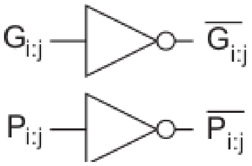
(b) Even Row Black Cell



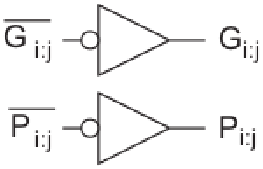
(a) Odd Row Gray Cell



(b) Even Row Gray Cell



(a) Odd Row Buffer Cell



(b) Even Row Buffer Cell

Cell	# of NAND Gates	# of NOR Gates	# of Inverters	# of Transistors
Odd Buffer	0	0	2	4
Even Buffer	0	0	2	4
Odd Black	2	1	1	14
Even Black	1	2	1	14
Odd Gray	1	1	1	10
Even Gray	1	1	1	10

Table 4: Logic gate and transistor count per cell

2.2.5 Core adder, optimized

Applying the optimized version of the cells into the Core Adder, the *Odd Cells* were used in rows 1,3,5,7 and *Even Cells* were used in rows 0,2,4,6. The new optimised schematic can be seen in picture (30b).

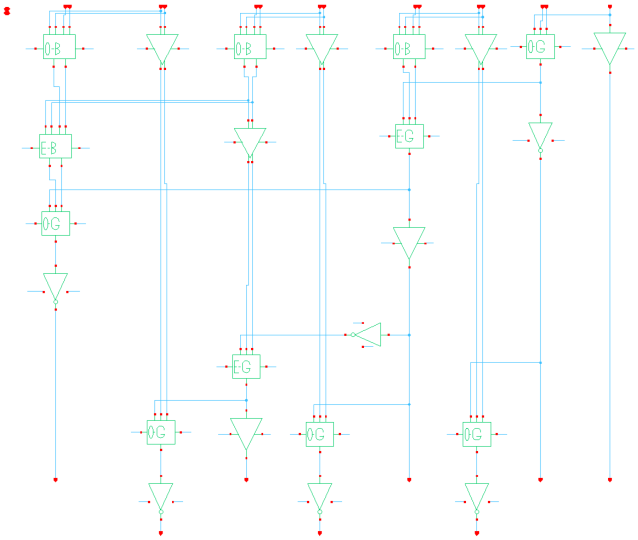


Figure 12: Optimized 8-bit Core Adder

	# of Even/Odd Gray Cells	# of Even/Odd Black Cells	# of Dual Buffer Cells	# of Single Buffer Cells
Cells	7	4	4	9
# of Transistors per Cell	10	14	4	2
Total # of Transistors	70	56	16	18

Table 5: Cell and Transistor count of Unoptimized Adder

All these result in **a total number of 160 transistors** in the whole Optimized Core Adder schematic.

2.2.6 Core Adder, Differences

All these optimizations result in a **decrement of 68 transistors**. This makes the design much faster and saves a lot of space in a later layout design.

2.2.7 Postprocessor

The post processor, shown in Fig. 26 under Sect. 5, consists of only the basic logic gates, and is a very straightforward design as far as the schematic goes. What is mentionworthy however, is the fact that each of the output bits is run through a 2-stage inverter chain, which we labeled as a ”boost buffer”. These buffers were introduced in a later iteration after initial schematic testing with the preprocessor, Brent-Kung core adder and initial postprocessor already in place, and their purpose is to reduce the rise and fall times on the output lines, as these were not within the specified range of at most 100ps. The values of the boost buffer were largely achieved by trial-and-error, where the following parameters were varied and increased in isolation (one at a time) with the aim of minimizing rise and fall times:

Parameter	Considered values
Stages	2, 4, 6, 8
W_{NMOS} ($\frac{W_{NMOS}}{W_{PMOS}} = 2$)	$[90, 665]$
Scaling factor	$\sqrt{2}, 1.5, 2, 3, 4$

Table 6: Varying parameters in optimizing boost buffer performance

Using a 2-stage inverter chain with W_{NMOS} of $665\mu m$ for the final stage and a scaling factor of 2, shown in Fig. 13 the results have been achieved (as compared to no buffer chain)

Configuration	Rise time	Fall time
No buffers	1115ps	743ps
With boost buffers	84ps	97ps

Table 7: Rise and fall times of S_1 for the same given bit transitions

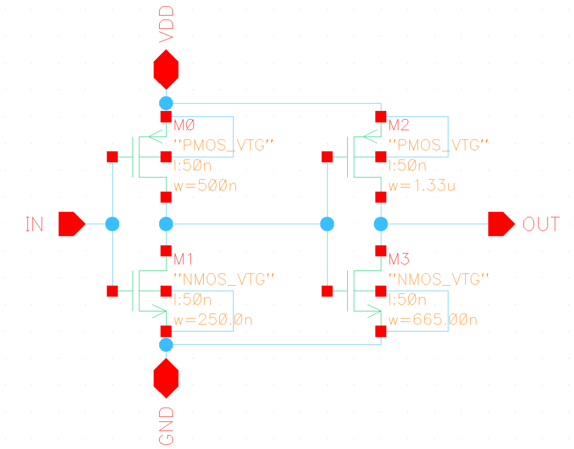


Figure 13: Schematic of a Boost Buffer

2.3 Layout Edit

2.3.1 General

In this section we will focus on how we tackled realizing the layout for our developed schematics. We will describe firstly how we structured our development, and how our eventual layout decisions followed from this. Furthermore, we explain particularly the concise methods used to minimize the total area of our adders taking into account the design rules [2].

2.3.2 Organization of structure and development

The development of the layout for the adder was done analogously to that of the schematic, starting with the transistor-level construction of the basic logic gates, and working further up the design hierarchy.

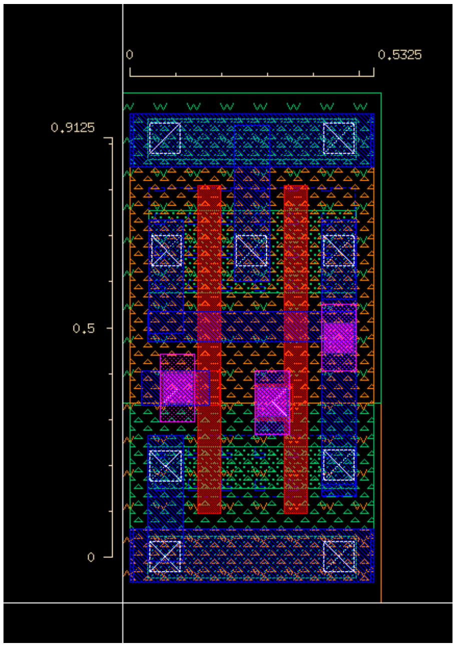


Figure 14: NAND Gate Layout

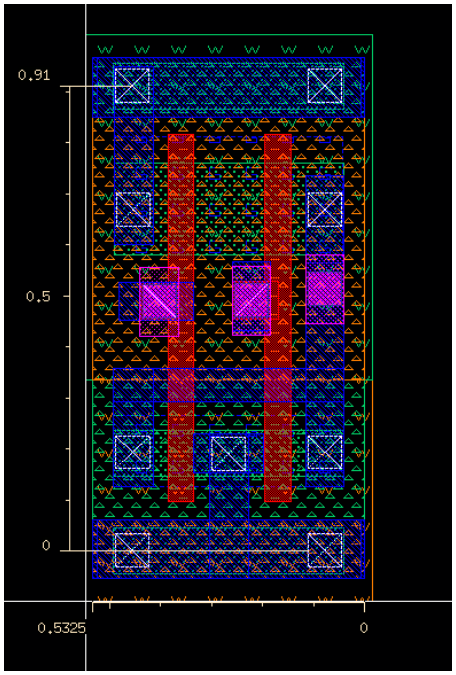


Figure 15: NOR Gate Layout

The pre- and post-processor were developed in isolation, and the core adder was a joint effort by both group members. To ensure smooth interactions between the individual efforts, the concept of ”lanes” was used, representing one bit (or its P-G pair) in the core adder and preprocessor, including its VDD and GND channels. The gates within the postprocessor have been separately organized in a similar fashion (with only one inverter width for a bitwise lane), but cannot be connected straightforwardly to the core adder due to the boost buffers. The reason for this is explained below.
To allow modular construction and simultaneous development, certain design decisions were fixed beforehand, such as lane width (determined by the height of black and gray cells) and the meeting point between lanes 4 and 5. After these parameters had been fixed, it was possible to work outwards from there, avoiding design collisions between the two separate designs.

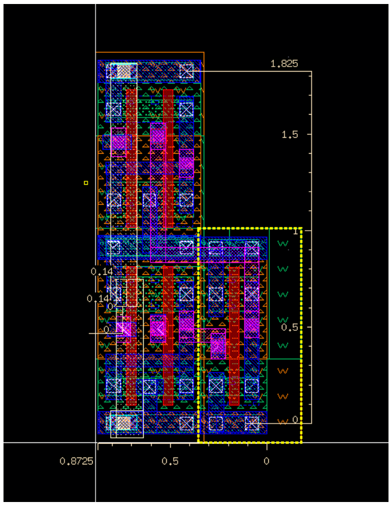


Figure 16: Even Gray Cell Layout

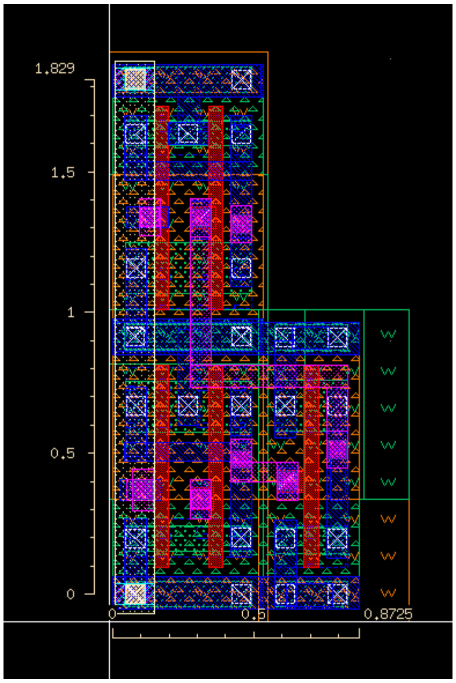


Figure 17: Odd Gray Cell Layout

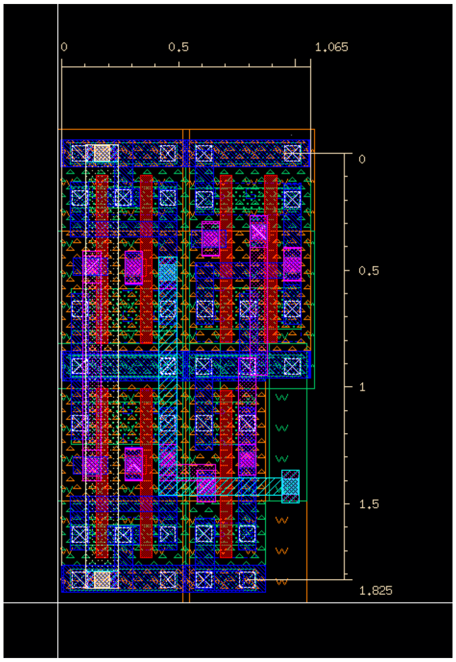


Figure 18: Even Black Cell Layout

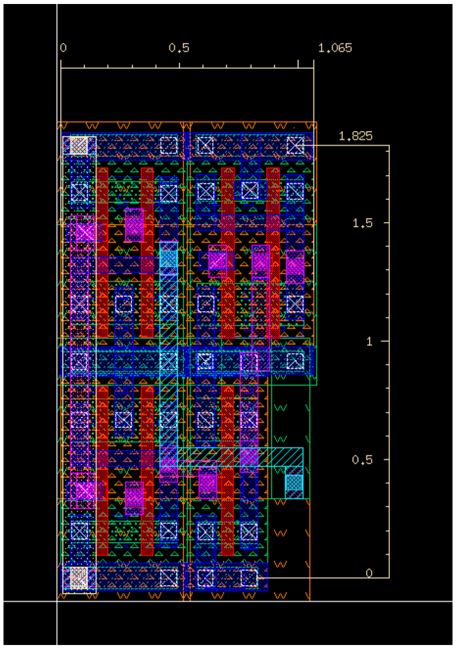


Figure 19: Odd Black Cell Layout

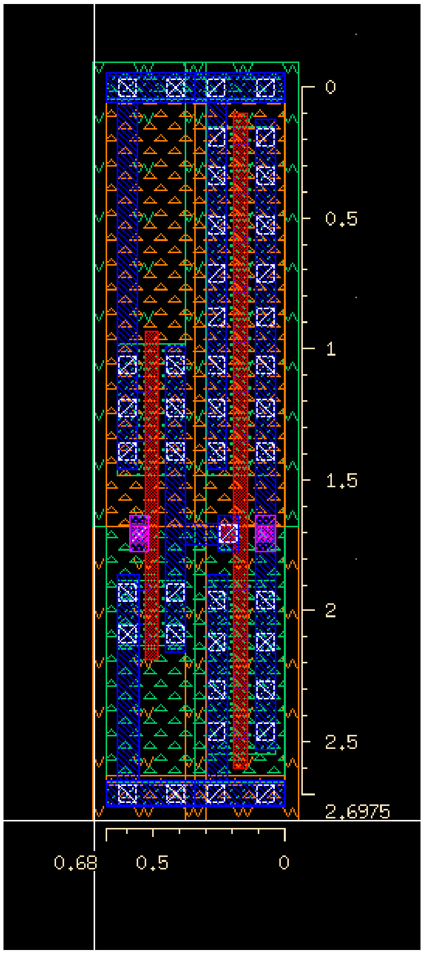


Figure 20: Layout of a Boost Buffer

2.3.3 Footprint optimization

As the area is one of the key metrics for the adder, it is only natural that we have tried to minimize the total footprint of our layout. We tried to minimize the space between components at every layer of the design hierarchy, and have employed a few methods to achieve this.
The first is trivial: minimizing space between lanes and individual cells within the core adder and pre- and postprocessor. We achieved this by taking appropriate margins in account when designing the cells to allow exact adjacent placement (sharing p/n wells). Furthermore, we repeatedly mirrored lanes along the horizontal axis as we moved upwards in our layout, such that the VDD channel at the top of lane n can serve as the VDD channel of an adjacently placed, mirrored cell in lane $n+1$ like on pictures above (19).
A second method we employed, is a structured use of different metal layers. Though not a strictly followed rule, we mostly applied the convention of allocating metal layers to given layers in the design hierarchy (also providing items in layer n with connection endpoints for layer $n+1$) as shown in Table 8. We only went up in metal layer within one layer if absolutely necessary.

Layer(s)	Metal used
Within individual logic gates	metal 1
Within P/G cells, buffers	metal 1 (sometimes metal 2)
Within lanes	metal 2 (sometimes metal 3)
Across lanes	metal 3
VDD, GND net	metal 4
Bypassing core adder (G-lanes)	metal 5

Table 8: Layers of design hierarchy and their corresponding metal layers

The last method we used, though not strictly a design strategy and more good engineering practice, is an effective use of resources yielded by forced sacrifices elsewhere in the design. In our case, this strategy evolved in the encountered scenario where some lanes of the core adder are simply longer than others. This yields leftover space in the shorter lanes as can be seen in picture (30b).
The formerly unused space was later however used to serve as a footprint area for the boost buffers (highlighted section of (30c)), as opposed to placing these in accordance with the lane-strategy, which would have expanded the design beyond its current measurements. Combined, the above yielded the final footprints (Fig. 21, Fig. 22) for the adders' layouts, which are shown in Table 9.

Measurement	Unoptimized adder	Optimized adder
$MaxHeight(\mu m)$	14.410	14.595
$MaxWidth(\mu m)$	9.085	7.478

Table 9: Measured values for different adders

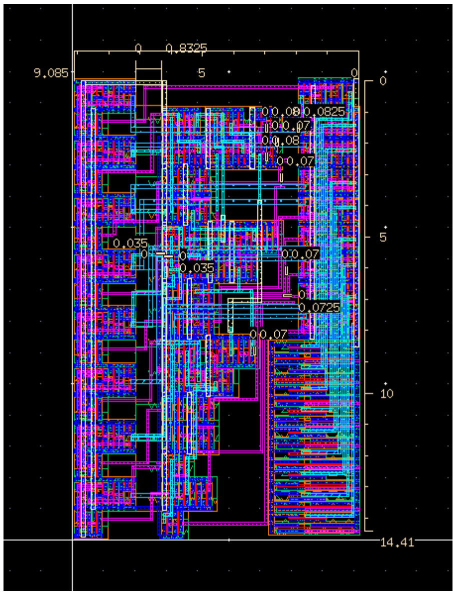


Figure 21: Unimproved adder layout with measurements

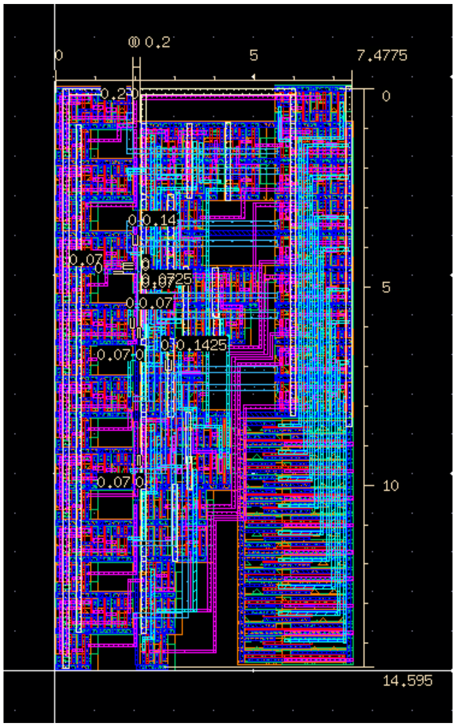


Figure 22: Improved adder layout with measurements

2.4 Layout verification

The designed layouts were verified by the provided DRC from Calibre both during design as well as in the end. Running DRC in intervals during the design process especially proved useful when assembling the core adder and connecting sections highest in the layout hierarchy. This is where the designs get progressively more dense, and a mistake can be made more easily.

Using Calibre's LVS in a similar manner proved slightly problematic for us, as an LVS check on incomplete design features many discrepancies that are irrelevant at that moment (due to missing components that are yet to be added). Hence, this was only done after a design had been completed.

Parasitics used for the eventual performance analysis of the layout were automatically extracted using PEX, also from Calibre.

2.5 Simulating schematic and layout

2.5.1 Simulation parameters

The used test setup for analyzing the functionality and performance of the gates can be found in Fig. 27. Here, 2 sets of seeded 8 random bit generators with a period of 2ns were used for bytes A and B, and one extra generator for the carry in. These generators were respectively fitted with seeds of incrementing values, starting at 0 and ending at 16. The entire schematic is supplied by a single VDD source of 1.0V, and each output bit is driving a 40fF capacitor. Simulations were run at 90.0°C over the course of 90ns simulation time, and using the provided ADE libraries.

2.5.2 Result analysis methods used

Result analysis was done using the ADE L tools provided by Virtuoso, where the transient reponse was the primary subject of interest. An example of an output trace is shown in Fig. 23. Preconfigured scripts were provided to test the adders' logical functionality (by comparing the expected output w.r.t the input bits to the output byte S and C_{out}) for a sequence of inputs, as well as the rise- and fall times. These scripts were naturally used to do these analyses.

The propagation delay was measured manually for our critical path, S_6 for the worst-case scenario, Fig. 23 displays this process. The worst-case scenario is when the input consists of exactly 1 high value and one low value for each location plus a high-value carry in, which was realised by fixing all $A_{0:7}$ to 0 and all $B_{0:7}$ to 1, and letting the C_{in} flip randomly.

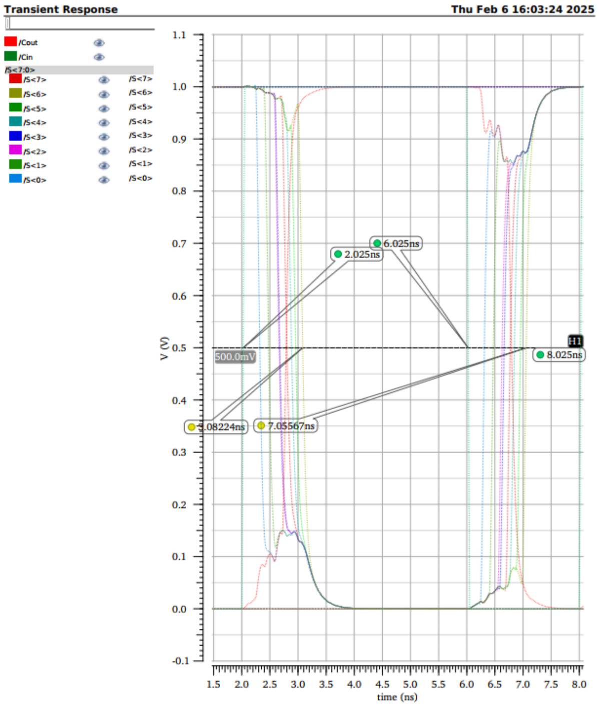


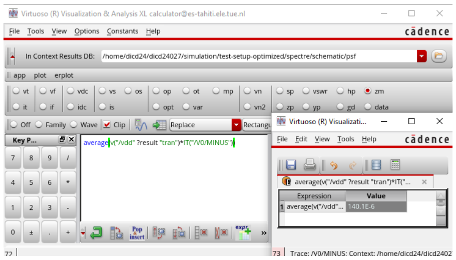
Figure 23: Transient analysis to obtain propagation delay for the unoptimized adder in the worst case scenario

2.6 Results

Using the test setup described in Sect. 2.5, the results in Table 11 were obtained through use of Virtuoso's calculator, an example of this process is shown in Fig. 24. The rise and fall time are for the worst-case scenario, the average power consumption has been taken over the randomly generated inputs.

Measurement	Unoptimized adder	Optimized adder
$delayHL$	1.129ns	1.057ns
$delayLH$	1.0ns	1.031ns
t_{period}	2.0ns	2.0ns
$U * I_{avg}$	$136.4\mu W$	$140.1\mu W$

Table 10: Measured values for different adders


Virtuoso (R) Visualization & Analysis XL calculator@es-tahiti.ele.tue.nl

In Context Results DB: /home/dicd24/dicd24027/simulation/test-setup-optimized/spectre/schematic/psf

```
average(v("/vdd" ?result "tran")*IT("/V0/MINUS"))
```

Virtuoso (R) Visualizati...

	Expression	Value
1	average(v("/vdd"...	140.1E-6

Trace: /V0/MINUS: Context: /home/dicd24/dicd2402

File Tools View Options Constants Help
cādence
app plot erplot
○ vt ○ vf ○ vdc ○ vs ○ os ○ op ○ ot ○ mp ○ vn ○ sp ○ vswr ○ hp ● zm
○ it ○ if ○ idc ○ is ○ opt ○ var ○ vn2 ○ zp ○ yp ○ gd ○ data
○ Off ○ Family ○ Wave ☑ Clip
Replace
Rectangu
Key P...
7 8 9 /
4 5 6 *
1 2 3 -
0 ± . +
Pop insert
expr
72
File Edit View Tools Help
cādence
73

Figure 24: Virtuoso's calculator being used to compute average power for the optimized adder

2.7 Performance

We will now assess the performance of the individual adders. The provided metric by which the designs are judged is based on the product of energy, area and delay, and is calculated as follows:

$$Performance = \frac{1}{Energy * Area * Delay}$$

The following calculations on the measured values found in Sect. 2.6 are applied (default SI units will be used):

$$Energy = U * I_{avg} * t_{period}$$

$$Area = max\_width * max\_height$$

$$Delay = (delayHL + delayLH)/2$$

Measurement	Unoptimized adder	Optimized adder
$Energy(J)$	$2.728 * 10^{-13}$	$2.802 * 10^{-13}$
$Area(m^2)$	$1.309 * 10^{-10}$	$1.0913 * 10^{-10}$
$Delay(s)$	$1.065 * 10^{-9}$	$1.044 * 10^{-9}$

Table 11: Measured values for different adders

Applying the formula described above, the following results are obtained. We present these as both their unitless value $Performance_{design}$, as well as $PerformanceIncrease$ given by the ratio of unimproved and improved design performance:

$$Performance_{unoptimized} = 1/(3.803 * 10^{-32}) = 0.263 * 10^{32}$$

$$Performance_{optimized} = 1/(3.192 * 10^{-32}) = 0.313 * 10^{32}$$

$$PerformanceIncrease = \frac{Performance_{optimized}}{Performance_{unoptimized}} = \frac{0.313}{0.263} = 1.191$$

Hence, we claim to have obtained a 1.191 times performance increase with our optimized design.

3 Future work

- The largest consumers of area in the preprocessor are the XOR gates. We considered later in the project to revisit this gate and to design a custom CMOS implementation for this (e.g. developing a custom AND-OR invert gate), but abandoned this idea as it was outside the scope of our intended improvements. This is a simple but likely effective possible improvement for future iterations of this design to reduce area and/or transistor count of this gate, and the preprocessor as a whole.

- In the optimized adder (and using the specified design strategies), the bottleneck for minimizing the adder footprint is the boost buffers. If these can be made less wide, the overall width of the design can be decreased. This can be feasibly achieved by getting the optimal stage count and size for the boost buffers by calculation instead of approaching acceptable results by trial-and-error.

- The spacing between pre-processor and core adder can be significantly reduced for unoptimized version. This is already shown by the fact that the optimized version already reduces this distance greatly, whilst using the same connection endpoints. We also believe the same to be true for optimized design (eg. by rerouting the left VDD strip, which is currently drawn in metal 1), though slightly severe.

- The design strategy of using lanes ”forced” us to make sacrifices in terms of area for the longer lanes, and though this area was used as well as we could manage, it's still suboptimal considered to disregarding the design decision for the gates at the ends of the longer lanes, and also placing these in that ”dead space”.

4 Reflection

4.1 Work division

In the schematic design stage of the project, Quinten created the basic logic gates, as well as the pre- and post processors, including the XOR gate and the later introduced boost buffer (and all experimentation that went into this). Stavros designed the gray and black cells, and built the unoptimized core adder. The test setup was built by Quinten, whilst Stavros was designing the optimized adder and its cells. The testing procedure for the schematics was carried out by Quinten, who also showed the work at the demo. Unfortunately, at this time, the optimized core adder was not functioning according to specification, debugging this was a (long) joint effort. Once it was working focus was shifted to the layout design. Here, the decision to split the design of cells and gates so that each other's work never overlapped was a very effective technique. Also, the decision to start building the core adder from opposite ends (bit 0-3 and bit 4-7) while using the same principles went as planned when bits 3 and 4 were connected in the final designs. During this phase, construction of the basic logic gates was a joint effort, from there, Quinten built the black odd and even cells, and the gray odd cell. Stavros built the gray even cell. The layout design stage plans were drafted by Quinten, and lanes 0-3 and 4-7 were assigned and built as described in the report. Quinten also built the pre- and post processor layouts, and assembled everything together, debugging DRC and LVS until a passing layout was formed. While Quinten was doing PEX on this layout, Stavros then built the unoptimized core adder schematic, which was debugged jointly and PEX was carried out by Quinten. All these resulted in a 40% - 60% work division for Stavros Spyridopoulos and Quinten Kustermans respectively.

4.2 Pitfalls and tricks

A major breakthrough for us in debugging DRC was the discovery that Calibre's tool is capable of highlighting DRC discrepancies. Similarly, the LVS tool's cross-referencing functionality saw increasingly more use throughout the project, and it significantly improved our efficiency in debugging mistakes for these checks. Also, the ability to edit lower-level instances while on a higher-level layout editor turned out to be very useful in such a large design.

4.3 Project feedback

Overall, this project was a learning experience for us both. A few points of feedback from our part are given below:

- The inverter lab was a very good way to get used to using virtuoso, and should be considered a crucial part of this course's practical assignments.

- The metric for performance should be defined more precisely, it is currently completely unitless, meaning that the chosen order of magnitude (if not SI) and method of calculation is solely what gives weight to the metrics. For instance, due to the fact that delay is in the order of pico/nano seconds, it can hardly ever outweigh area, which is in the order of micrometers.

- It was mentioned in the project manual that we would be provided with unknown test signals, which was not entirely the case, as we were free to design our own test setups. Perhaps providing students with a standardized testing circuit can prevent this. It can also greatly help prevent issues with the provided test scripts due to student's own naming conventions, and it (along with a standardized metric) even allows equal performance assessment across the entire class (and thus across the entire assignment).

REFERENCES

[1] D. M. H. Neil H. E. Weste. *CMOS VLSI Design A Circuits and Systems Perspective*. Addison-Wesley, 2010.

[2] N. S. University. FreePDK45 — NC State EDA.

5 Appendix A: Large schematics

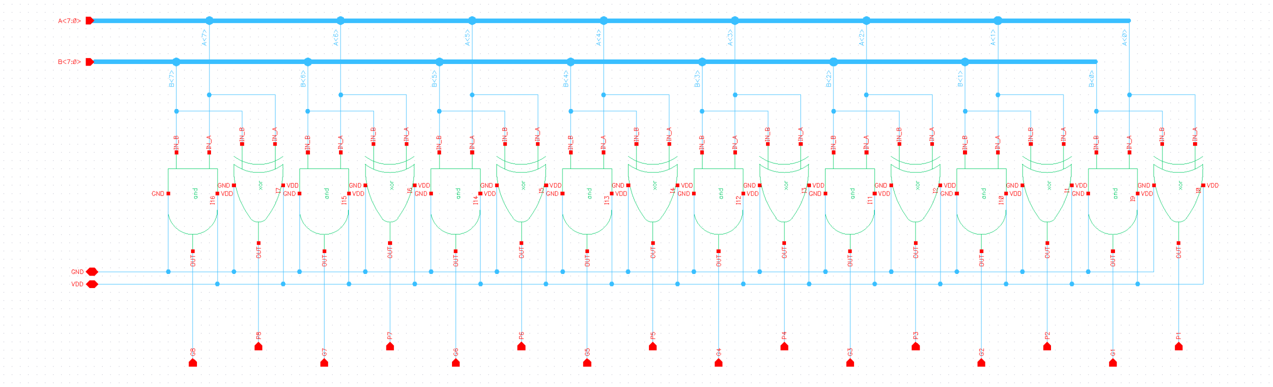


Figure 25: Preprocessor schematic

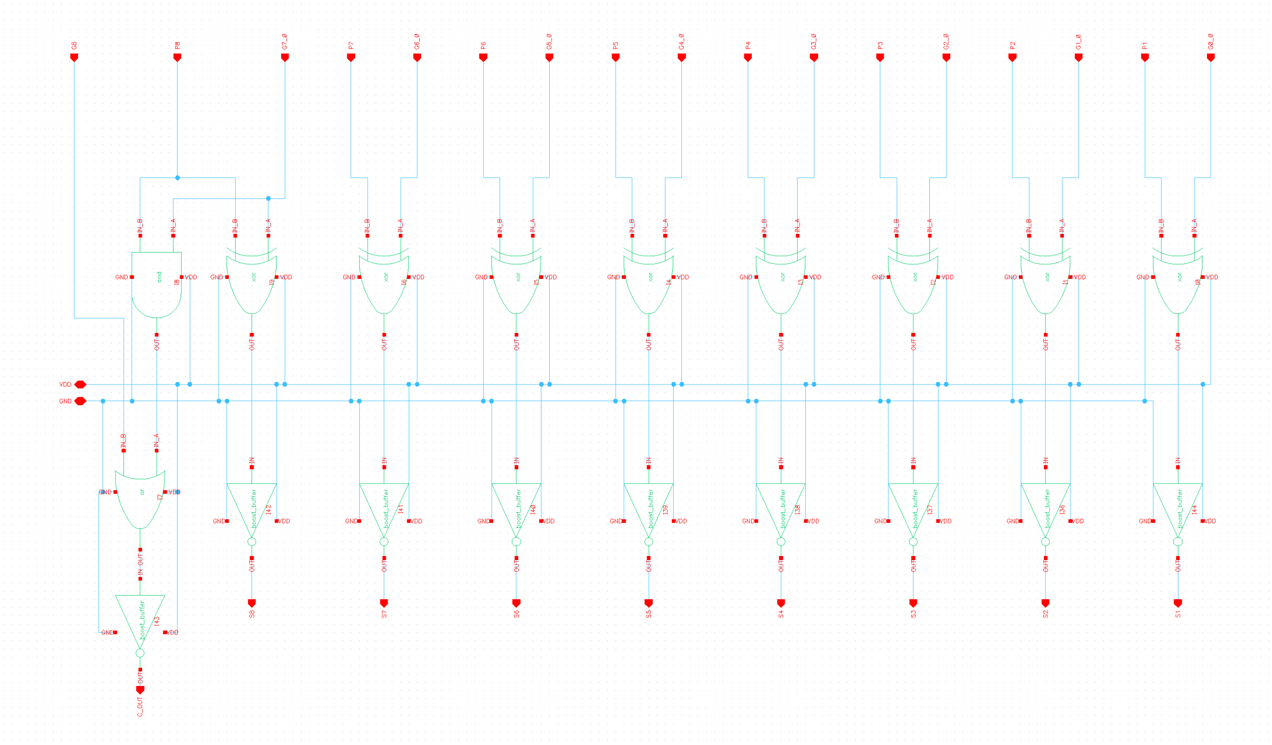


Figure 26: Postprocessor schematic

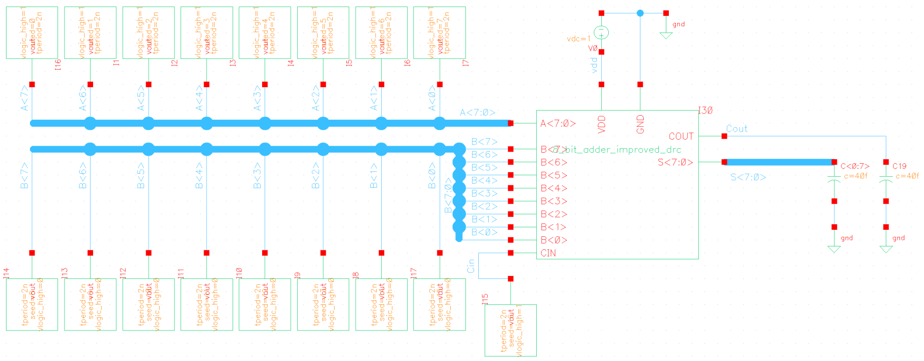


Figure 27: Test setup schematic for the optimized adder

6 Appendix B: Large Layouts

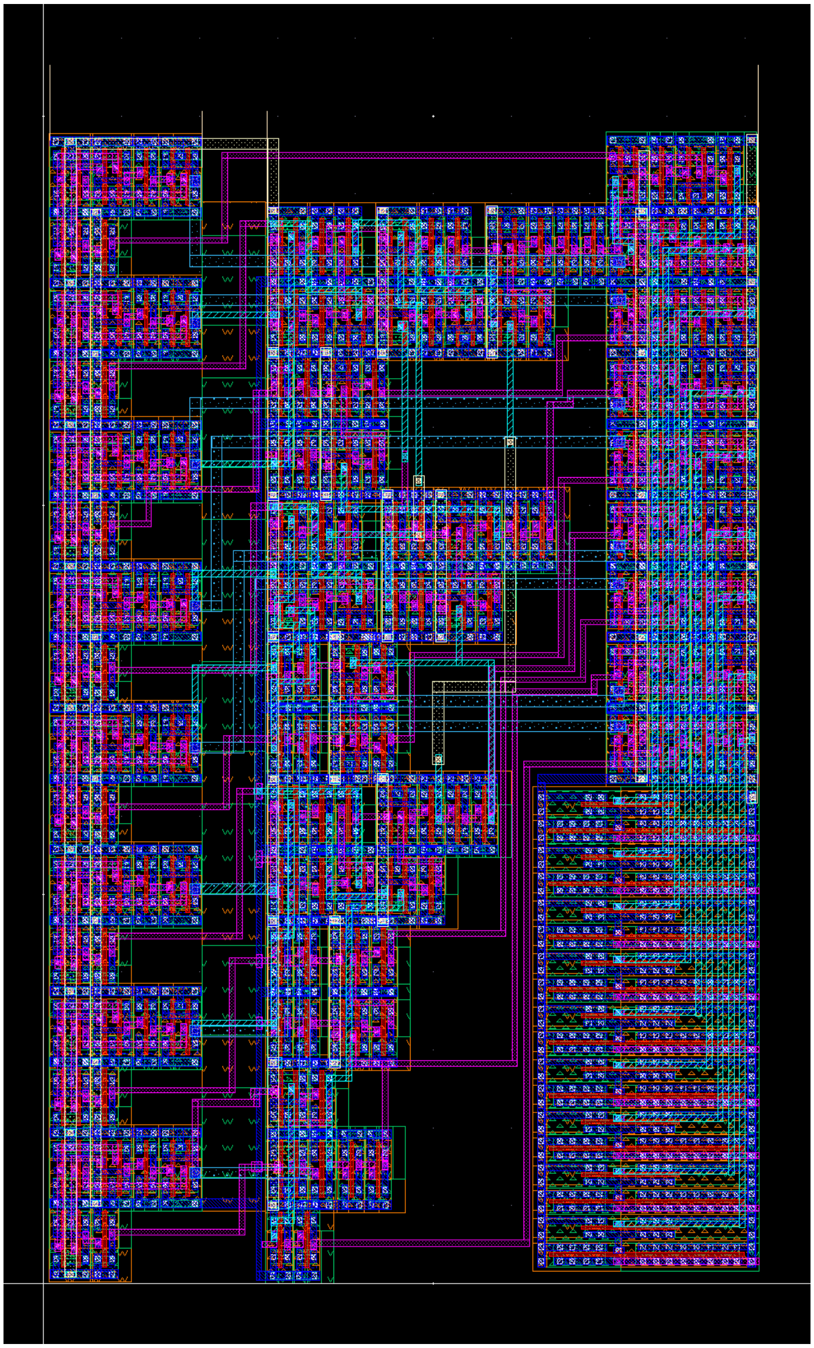


Figure 28: Unoptimized 8-bit Adder

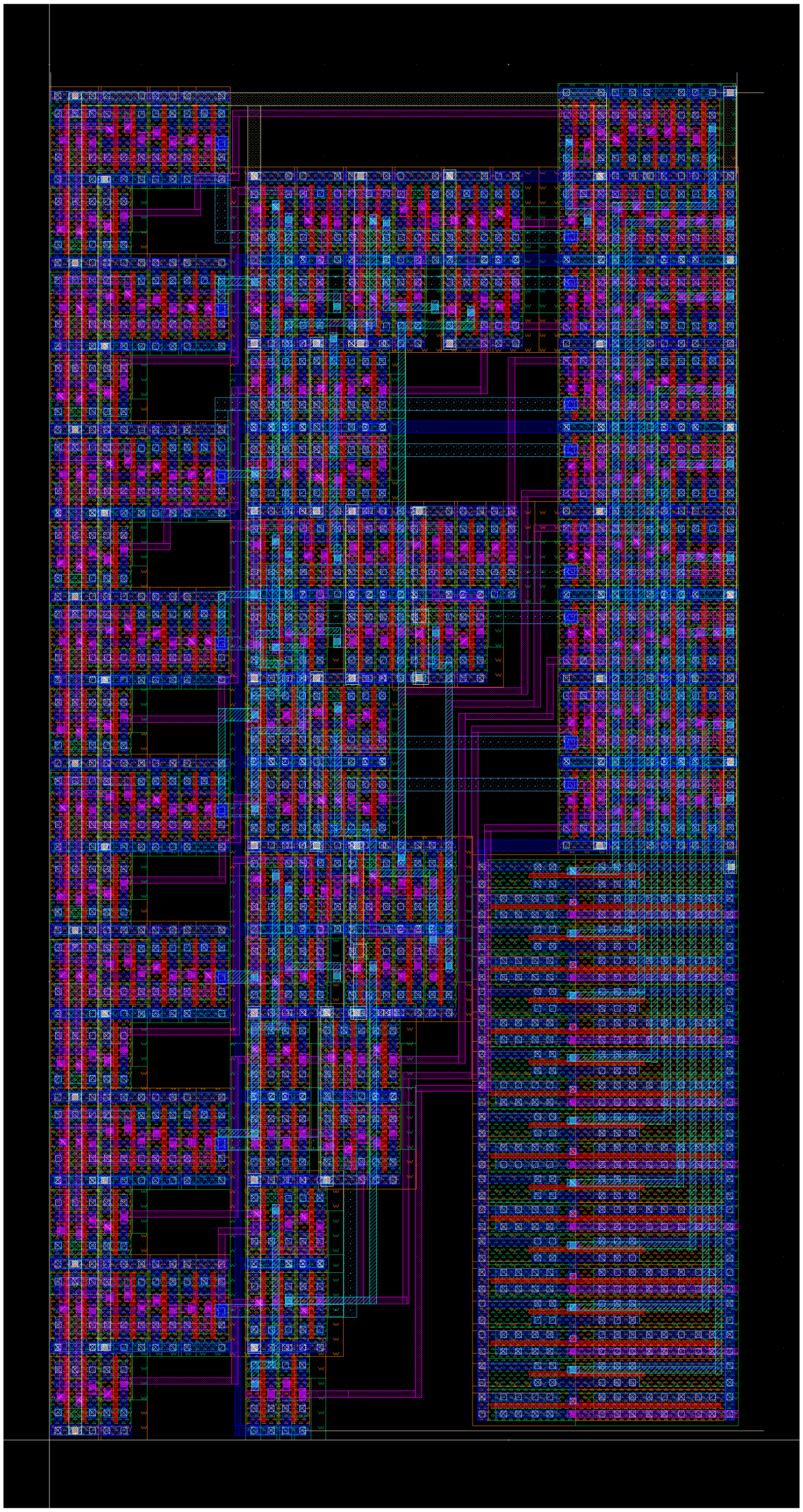
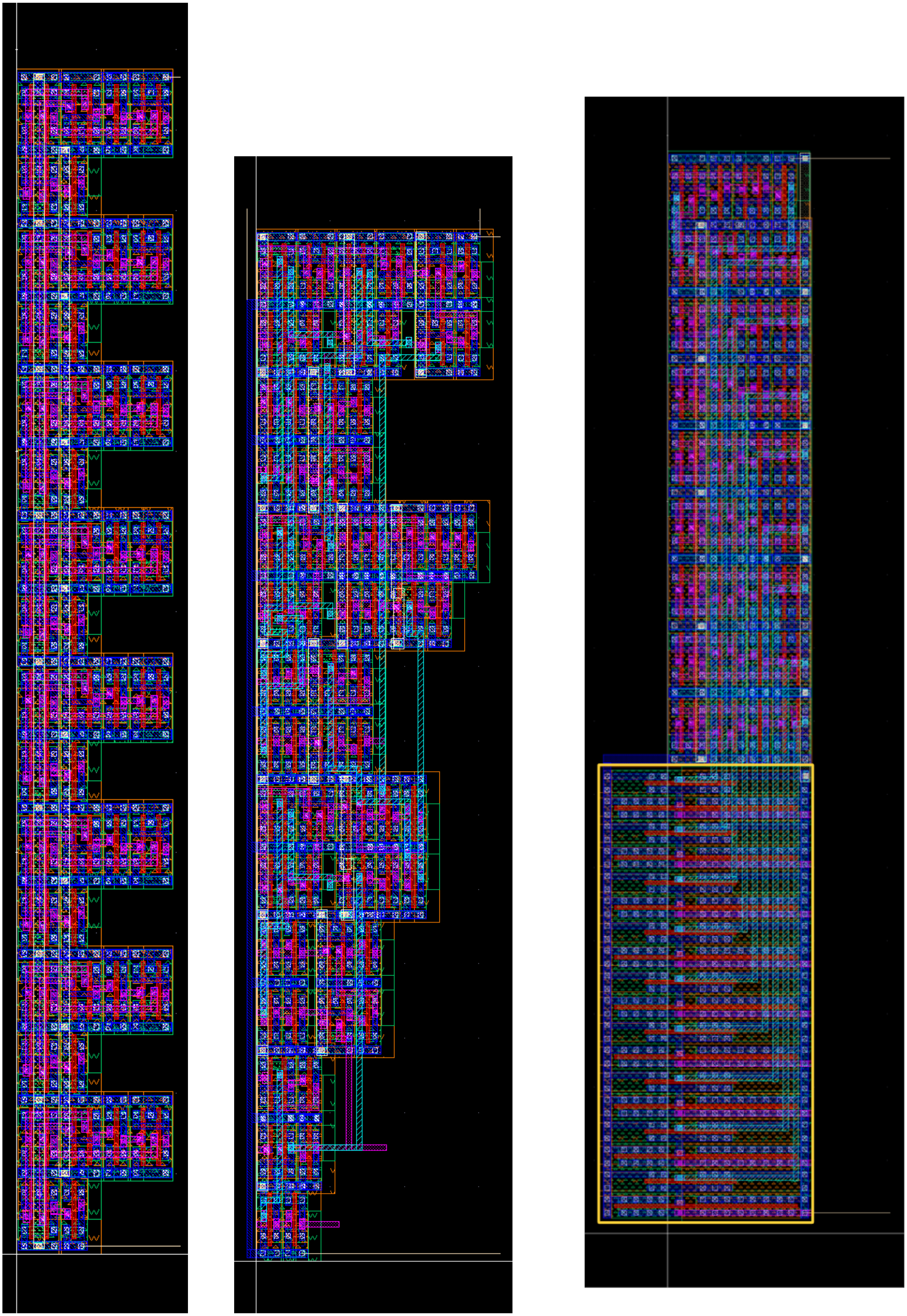


Figure 29: Optimized 8-bit Adder



(a) Pre-processor Layout

(b) Optimized Core Adder Layout

(c) Post-processor Layout - Highlighted Boost Buffers

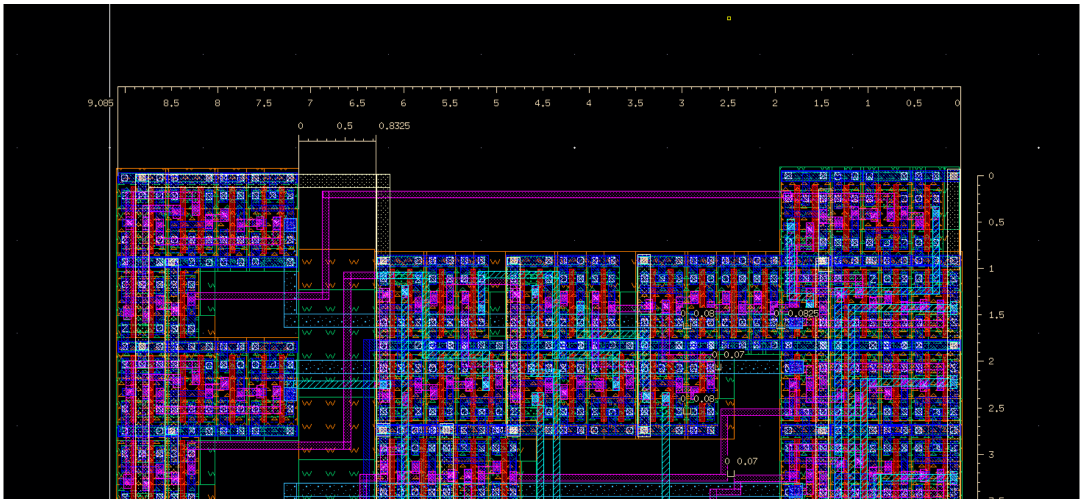


Figure 31: Spacing between preprocessor and adder in unimproved version

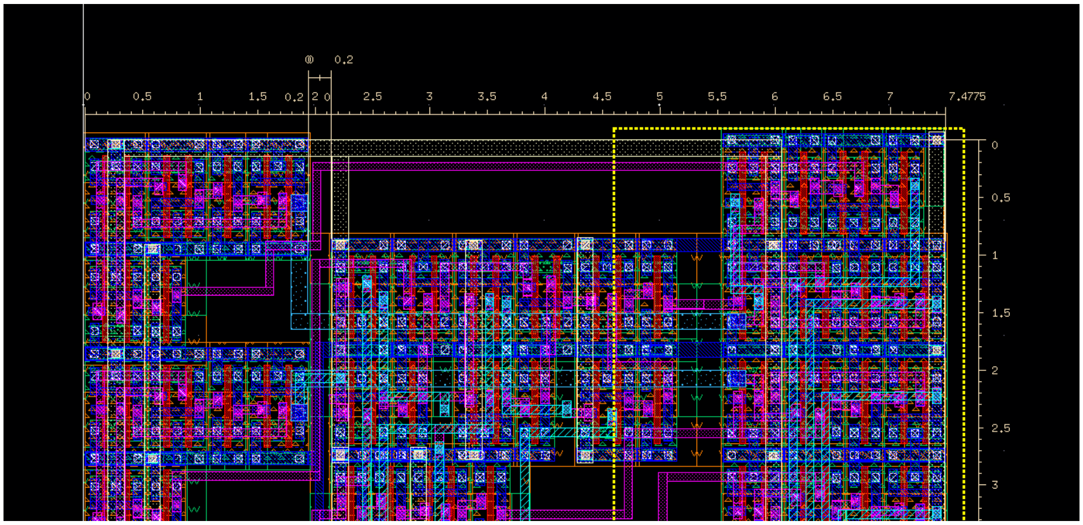


Figure 32: Spacing between preprocessor and adder in improved version