ECE 165 Final Project: 8-bit Kogge Stone Adder

Luke Wilson | A16745607

Siona Ahmed | A17372404

Mustahsin Zarif | A17499159

Adders are essential arithmetic blocks for any microchip. Having an adder that fits your requirements is necessary if you want a good microprocessor. There are various strategies to construct adders – all varying in size, speed, complexity, and power consumption. The challenge we are tasked with in this project is to optimize speed, while keeping our power at a reasonable level. In this paper, we will look at a custom 8-bit adder with a Kogge-Stone architecture. This architecture is attractive because it allows us to have the benefits of carry-look-ahead (CLA) architecture (speed) without the high fan-in that comes with CLA in large bit adders. Kogge-Stone achieves this by splitting the logic into layers. This is the case for any tree adder. What makes Kogge-Stone different from other tree adders is that the branches encompass every single input index – resembling a binary tree.

In a previous approach, we implemented a static logic Ripple-Carry Adder (RCA) architecture to employ an 8-bit adder. However, it was not optimal for speed – topping out at 2.3 GHz. All the group generates are calculated sequentially; therefore, the carry out for each stage cannot be calculated until the carry in, which is the carry out of the previous stage, is calculated. Hence, in the worst-case propagation delay scenario, we would need the carry to be propagated through each stage. This would severely slow down our logic.

While an RCA has a time complexity of O(N), a Kogge-stone has a time complexity of O(log2N). If we consider that we are using two 8-bit numbers in this project, that means the Kogge-stone should be much faster than the Ripple-carry. That reasoning is precisely why we decided to choose the Kogge-stone architecture. Initially we attempted dynamic logic, but we ended up switching to static logic in the end. More will be discussed about that unsuccessful attempt later.

Now we will describe how our Kogge-Stone adder functions. The first step is to translate the input bit pairs to P's and G's. We then group index neighboring PG values to create group-PG values, and then group neighboring group-PG values to create even larger group-PG values. The optimal number of layers for an 8-bit architecture is 3. This allows us to create all Gk:0 in a very time-efficient manner. For the Kogge-Stone, the critical path is A=00000000, B=011111111. This is because the 8th sum bit requires one XOR gate less logic than the 7th output bit. We also tried to used the smallest sizing possible to reduce

capacitance and as a result delay. With this architecture we were able to achieve 3.1 GHz. Another important thing to mention is that we designed all of our cells at the transistor level except for the XOR; standard cells were used.

As we stated earlier, we initially tried dynamic logic but decided to switch to static. Dynamic logic's most crippling problem was weak drive strengths. We tried to compensate for this by inserting buffers, but this was very severely ramping up our power usage. Without the buffers, the outputs from the earlier layers were unsynchronized, and as a result our XOR's would output incorrect values. Another issue we faced was a preemptive clock signal for the XOR's and some of the third layer group-PG blocks. The rise edge of the clock would arrive at these blocks before their inputs would. After many hours of discussion and troubleshooting, we disregarded the dynamic approach and opted for static logic instead.

For a future design, we would still use the Kogge-Stone architecture with dynamic logic, but we wouldn't design each gate by hand. If we had used standard cells, we suspect the drive strength would not be as much of a problem. We would most likely still need to buffer the clock for deeper layers, but that problem is much more manageable than the unsynchronized inputs.

With our static adder, we have measured a maximum clock frequency of 3.1GHz with a 1.1V power supply, hence outperforming the RCA by 800MHz. This was validated using transient waveform plots showing desired timing behavior for over 10 input transitions. We also verified that the Kogge-stone based adder required 451.8 µW of power to operate, slightly more than the RCA at 318.42 µW.

Luke Wilson was the team lead and made both adder designs in Cadence. Siona simulated the circuit and helped design many of the blocks with Luke, along with the Kogge-stone adder. Mustahsin helped both Luke and Siona with troubleshooting the adders and writing the report.

Acknowledgments:

We would like to express our gratitude to Professor Mercier, the Teaching Assistants Theresa Pia and Matthew , and our classmates for their continuous support.

References:

[1] CMOS VLSI Design A Circuits and Systems Perspective, Neil H. E. Weste, David Money Harris

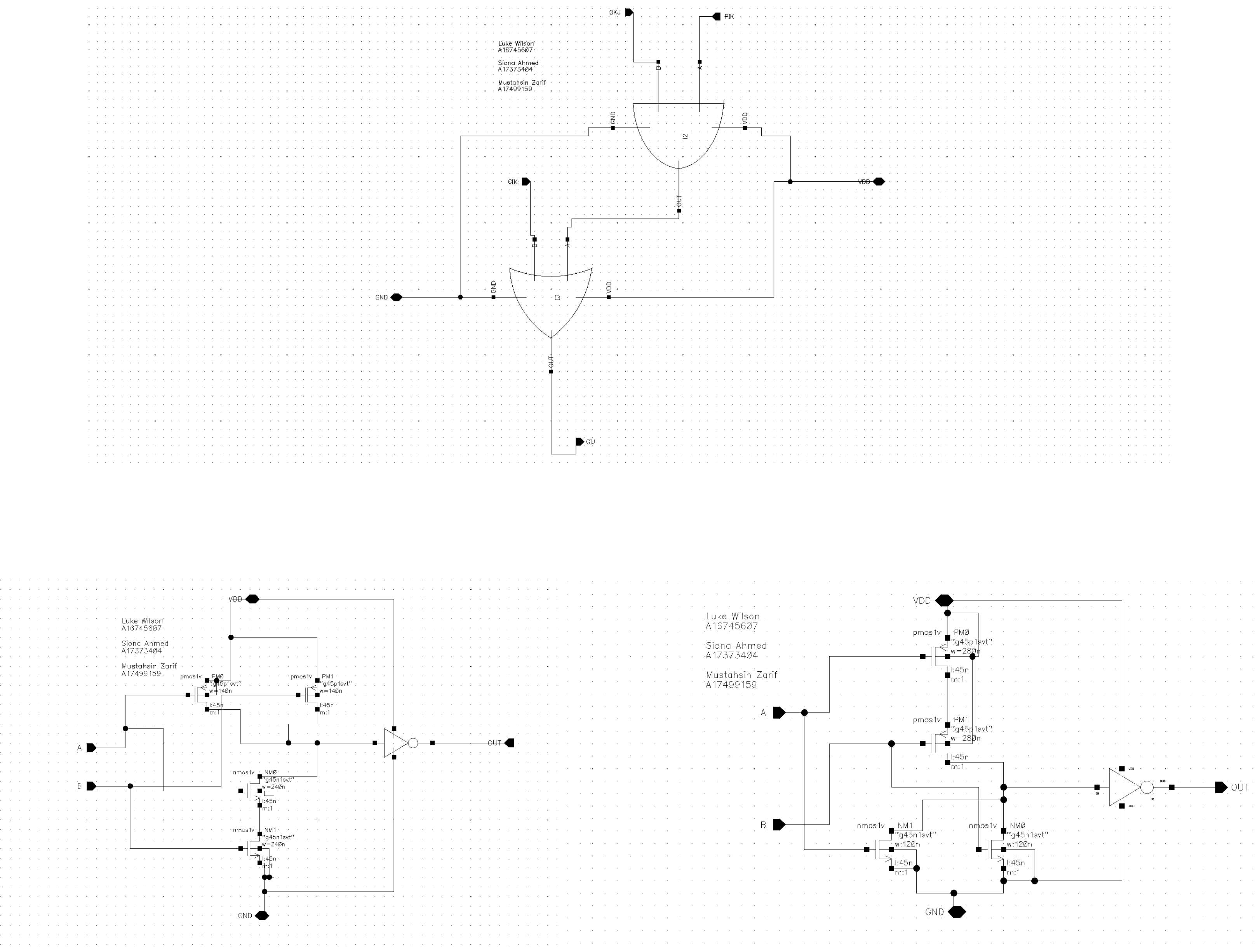

Luke Wilson
A16745607
Siona Ahmed
A17373404
Mustahsin Zarif
A17499159
GKJ
PIK
GIK
GIJ
GND
VDD
I2
I3
A
B
OUT
pmos1v PM0 "g45p1svt" w=140n l:45n m:1
pmos1v PM1 "g45p1svt" w=140n l:45n m:1
nmos1v NM0 "g45n1svt" w=240n l:45n m:1
nmos1v NM1 "g45n1svt" w=240n l:45n m:1
pmos1v PM0 "g45p1svt" w=280n l:45n m:1
pmos1v PM1 "g45p1svt" w=280n l:45n m:1
nmos1v NM1 "g45n1svt" w:120n l:45n m:1
nmos1v NM0 "g45n1svt" w:120n l:45n m:1

Figure 1 | Top: Group-G Block, Bottom-left: AND, Bottom-right, OR

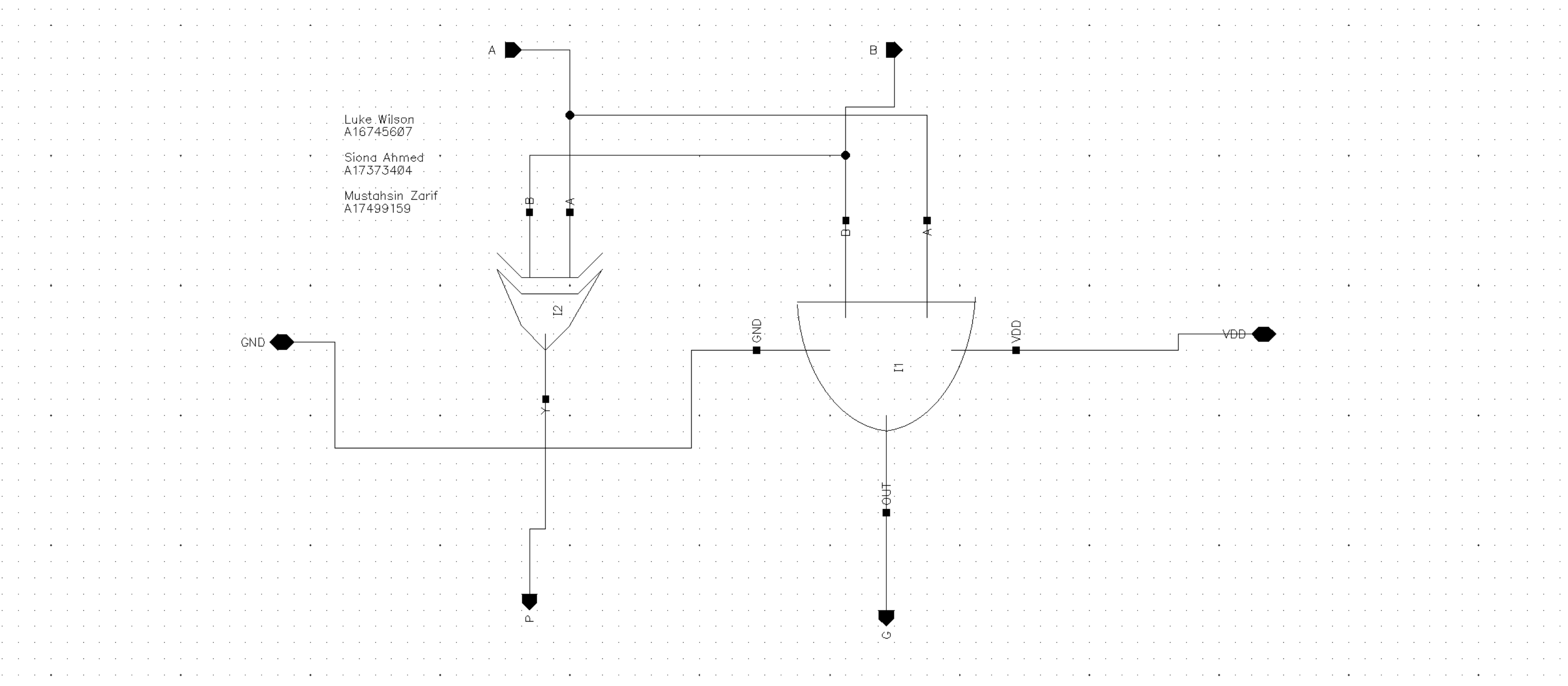


Figure 2 | PG Block

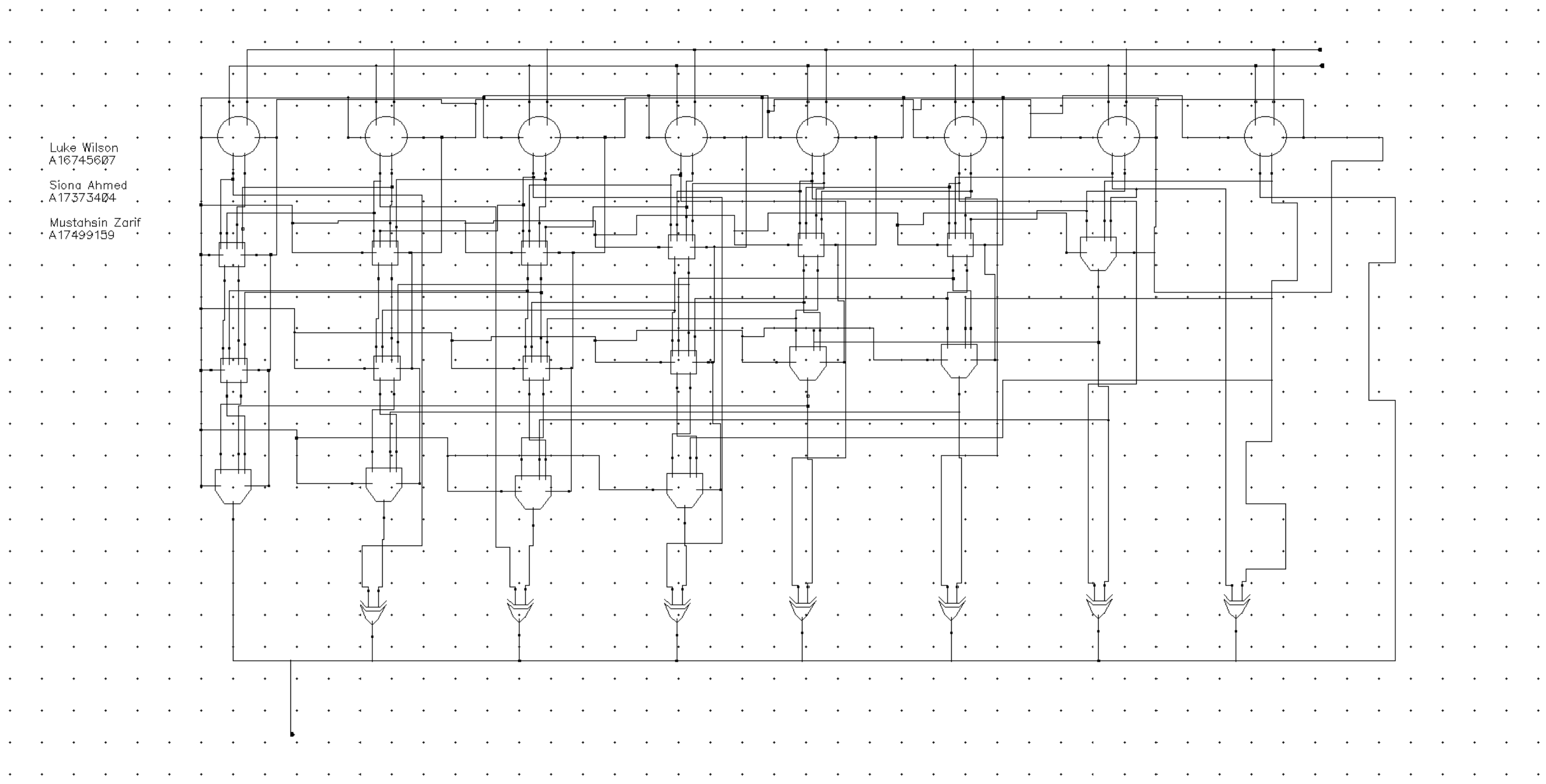


Figure 3 | Static 8-bit Kogge-Stone Adder

Figure 4 | n/a no layout done

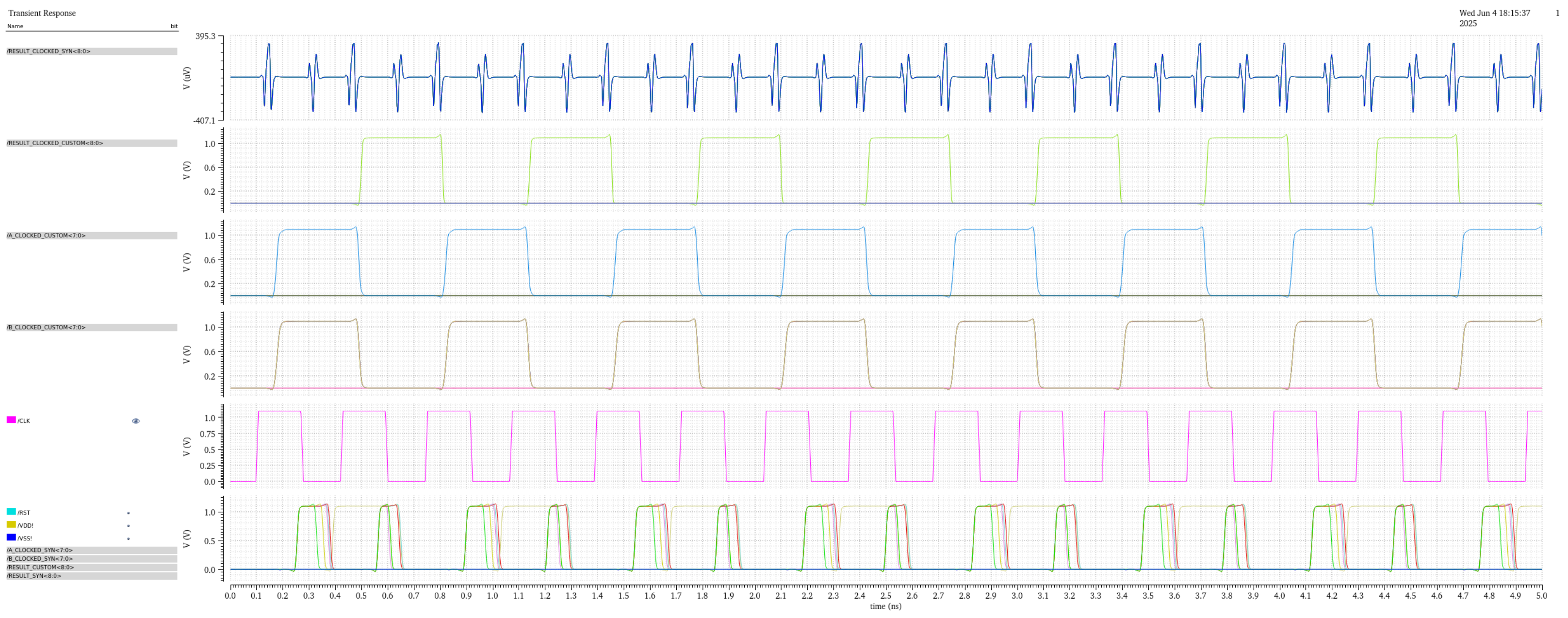
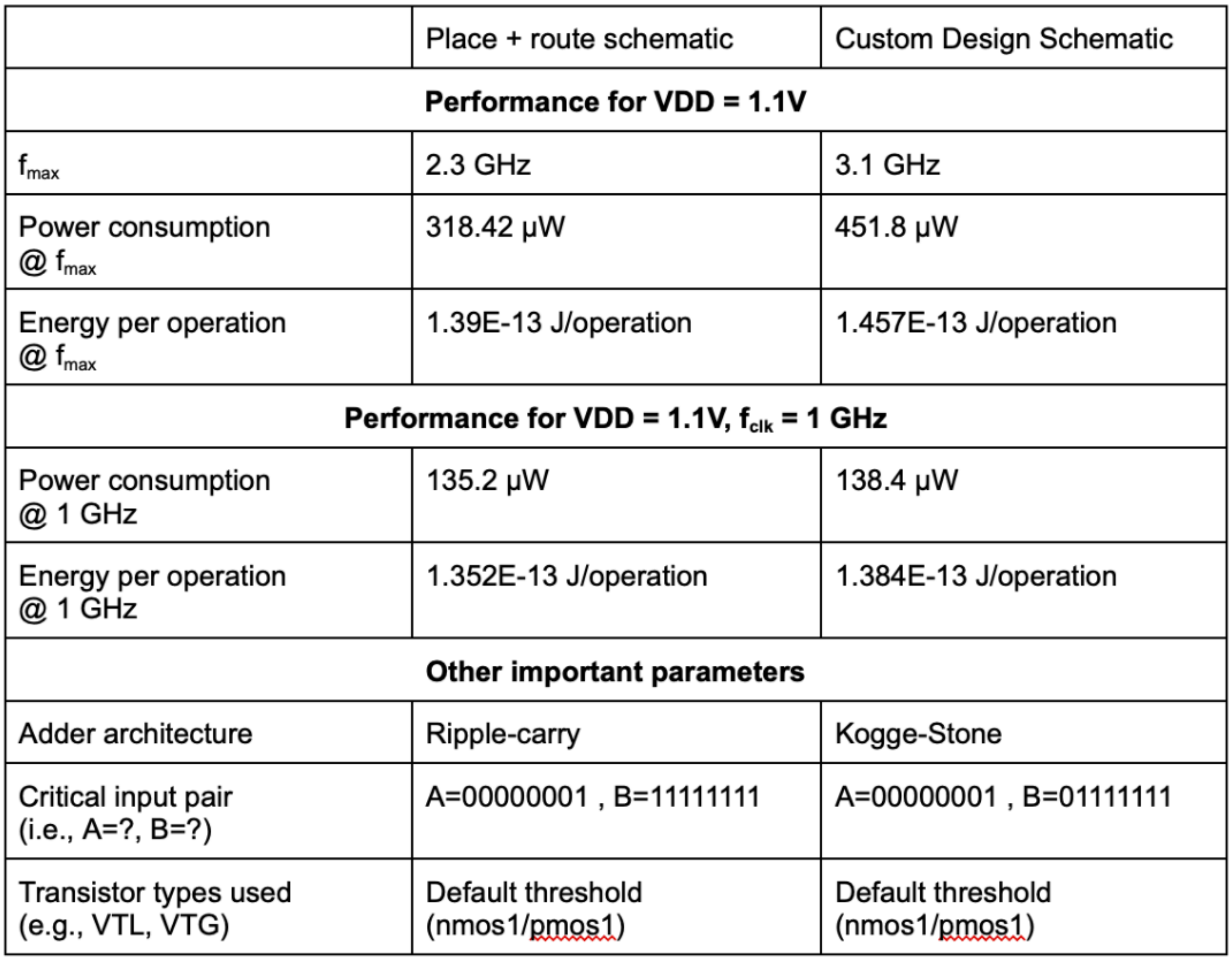
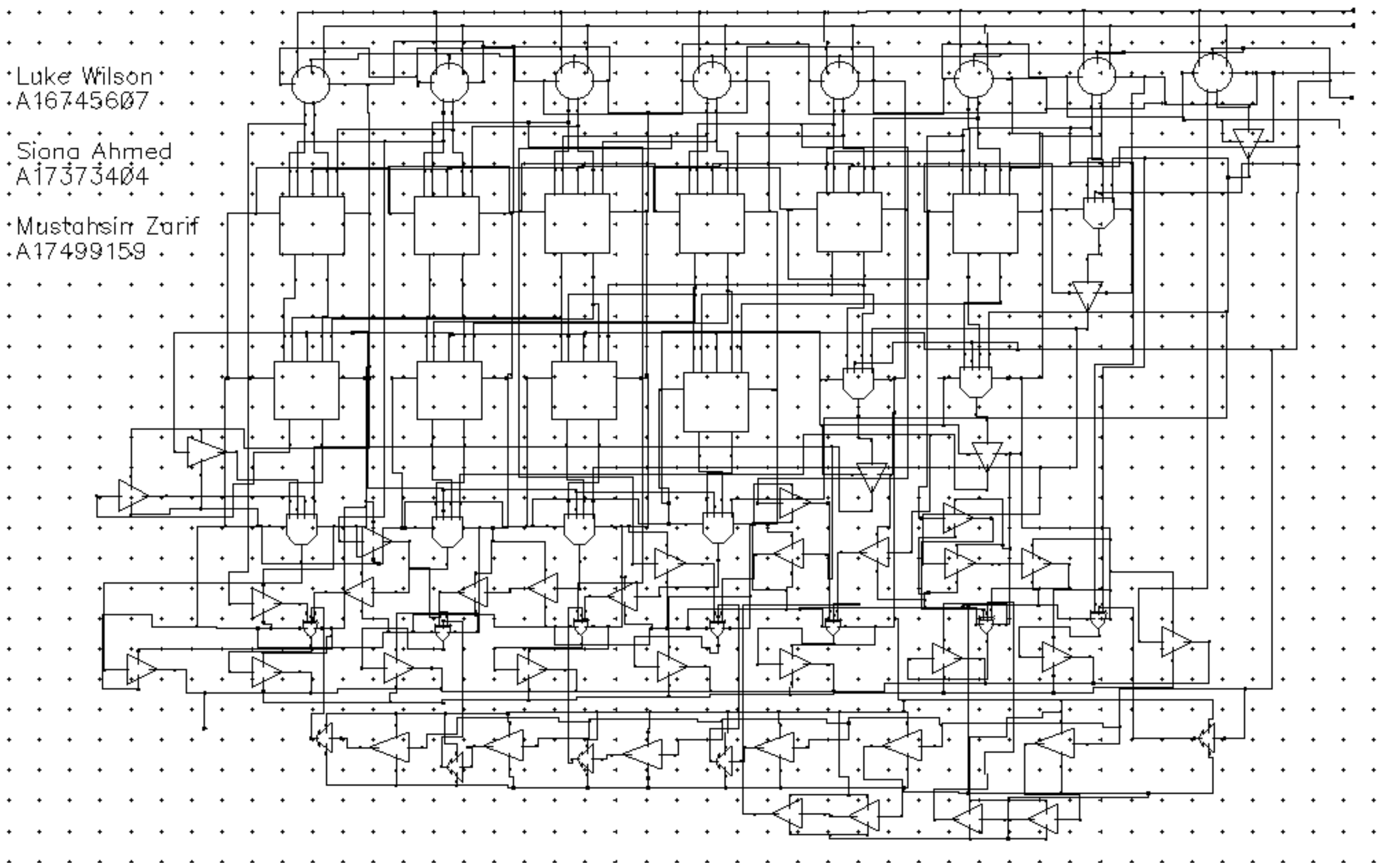


Figure 5 | Max Frequency Simulation results at critical Path



	Place + route schematic	Custom Design Schematic
Performance for VDD = 1.1V		
f_{max}	2.3 GHz	3.1 GHz
Power consumption @ f_{max}	318.42 μW	451.8 μW
Energy per operation @ f_{max}	1.39E-13 J/operation	1.457E-13 J/operation
Performance for VDD = 1.1V, f_{clk} = 1 GHz		
Power consumption @ 1 GHz	135.2 μW	138.4 μW
Energy per operation @ 1 GHz	1.352E-13 J/operation	1.384E-13 J/operation
Other important parameters		
Adder architecture	Ripple-carry	Kogge-Stone
Critical input pair (i.e., A=?, B=?)	A=00000001 , B=11111111	A=00000001 , B=01111111
Transistor types used (e.g., VTL, VTG)	Default threshold (nmos1/pmos1)	Default threshold (nmos1/pmos1)

Figure 6 | Insert caption here



Optional Figure 7 | Dynamic adder in final state before we switched to Static