ECE 429 - Final Project Report
Numerically Controlled Oscillator (NCO)
Matthew Janke, Evan Rose, Dan Taylor, Nick Ackerman, and Zhengtai Zhong
May 7th, 2016

I have neither given or received nor have I tolerated others' use of unauthorized aid.

Table of Contents

Functional Description

The Numerically Controlled Oscillator is a device that outputs an approximate 50% duty cycle square wave at a frequency that is a percentage of the clock frequency. The frequency of the square wave is determined by an 8-bit frequency control word (fcw[7:0]). The frequency control word along with the input clock signal are the device inputs and the one bit output square wave is the only device output.

The Numerically Controlled Oscillator (NCO) design uses two basic components; an 8 bit full adder and an 8 bit parallel input/parallel output register composed of D flip flops. Assuming some frequency control word, the 8 bit adder will add the fcw with the current contents of the 8 bit register. On the next rising clock edge, the 8 bit register will store the 8 bit sum from the adder. Upon the following rising edge, the 8 bit adder will add the fcw with the sum currently stored in the register creating a new sum that will be added with the fcw on the following rising edge. The NCO effectively increments to 256 (2^8) by the fcw. The most significant bit of the 8 bit register is used as the output for the NCO.

The frequency of the output square wave can be determined using the following formula:

$$f_{OUT} = (fcw/256)*f_{CLK}$$

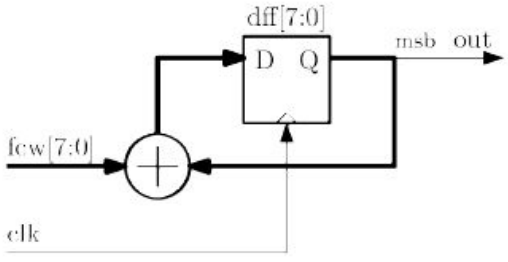

Figure 1: NCO diagram

Block Diagram

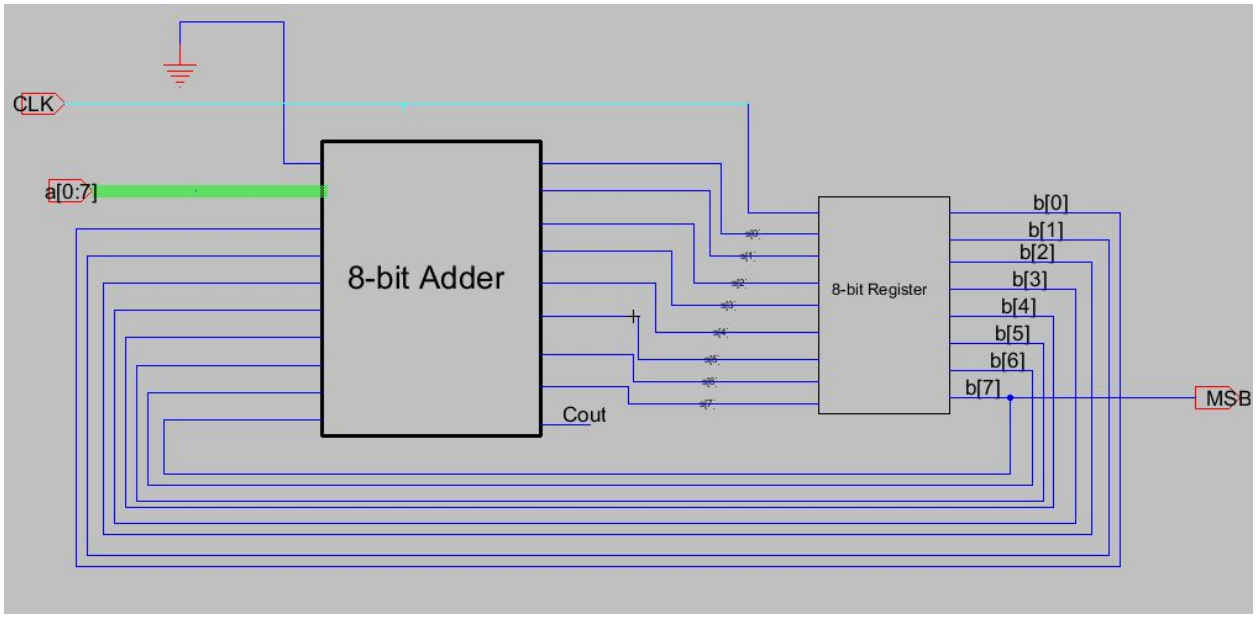

Figure 2: Block Diagram for NCO

The figure above is a high level block diagram of the NCO. The component on the left side is the 8 bit full adder and the component on the right is the 8 bit PIPO register. The *fcw* and *clk* inputs are labeled above as well as the *msb* output. A more thorough signal description can be found in the Signal Description Table (Table 1). As evident in the above schematic, the sum of the adder is always re added with the *fcw* to achieve the desired incrementing effect.

Signal Descriptions

Signal	Type	Description
fcw [7]	Input	**Data:** The *fcw [7]* is the most significant bit of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
fcw [6]	Input	**Data:** The *fcw* [6] is the part of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
fcw [5]	Input	**Data:** The *fcw* [5] is the part of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
fcw [4]	Input	**Data:** The *fcw* [4] is the part of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
fcw [3]	Input	**Data:** The *fcw* [3] is the part of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
fcw [2]	Input	**Data:** The *fcw* [2] is the part of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
fcw [1]	Input	**Data:** The *fcw* [1] is the part of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
fcw [0]	Input	**Data:** The *fcw* [0] is the part of the 8 bit frequency control word. This input bit is asynchronous to the clock (clk) and is controlled by the Serial Peripheral Interface.
clk	Input	**Clock:** The clk input signal provides the timing for the 8 bit PIPO register. All commands, adresses, or data present at the register inputs are latched to the rising edge of the clock (clk).
msb	Output	**Data:** The msb output signal is a single bit that transmits a 50% (ish) duty cycle square wave on the rising edge of the clock (clk) input.
Vcc	Power	**Device core power supply:** Source voltage
GND	Ground	**Ground:** Reference for the Vcc supply voltage

Table 1: Signal Descriptions

NCO Specifications

NCO Specifications						
Symbol	**Parameter**	**Cell Type**	**Test Conditions**	**Typical**	**Max**	**Units**
t_cq	Clock to Q Delay for NCO	Schematic	fcw = 0x1F	0.385	-	ns
		Layout	fcw = 0x1F	1.9	-	ns
f_clk	Operating Clock Frequency	Schematic	fcw = 0xEF	20	400	MHz
		Layout	fcw = 0xEF	20	500	MHz
A_total	Total Die Area for NCO	Layout	-	-	0.2597	um^2
Internal Specifications						
Symbol	**Parameter**	**Cell Type**	**Test Conditions**	**Typical**	**Max**	**Units**
t_prop	Propogation Delay through the 8bit Adder	Schematic	A=0xF7, B=0x08	1.34	-	ns
		Layout	A=0xF7, B=0x08	10.2	-	ns
t_su	Setup Time Delay for Register	Schematic	clk at 20MHz	30	-	ps
		Layout	clk at 20MHz	30.7	-	ps
t_cqr	Clock to Q Delay for 8bit Register	Schematic	0xFF	0.46	-	ns
		Layout	0xFF	0.55	-	ns

Table 2: NCO Specifications

The following specifications provided for the NCO were successfully met. The NCO shall use an 8-bit frequency control word (*fcw*). The NCO shall properly operate with a minimum clock frequency of 20 MHz. As described in the functional description, the NCO operates with an 8-bit input frequency control word. During our specification testing, we used a value of 0x10 (0001 0000) and 0xEF (0111 1111). The maximum and minimum frequency values are also detailed above and prove our design meet the minimum frequency criteria.

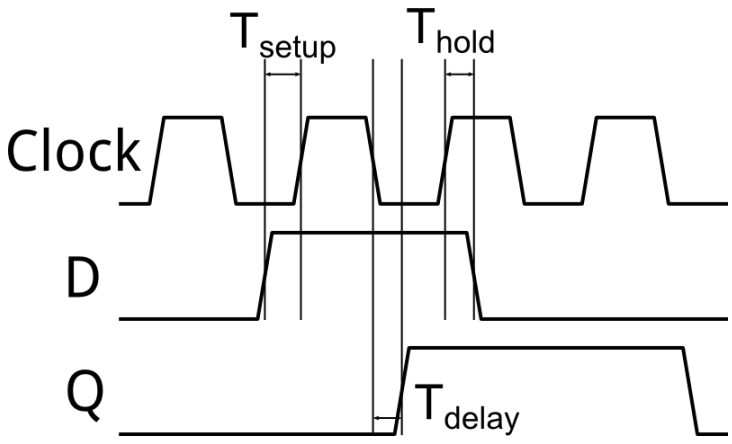

Figure 3: Timing Diagram Example

Schematic Simulations

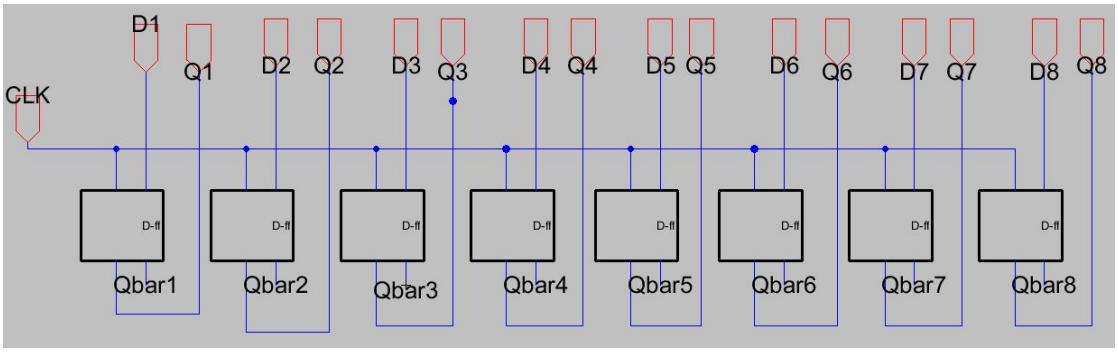

Figure 4: 8 Bit PIPO Register - Schematic

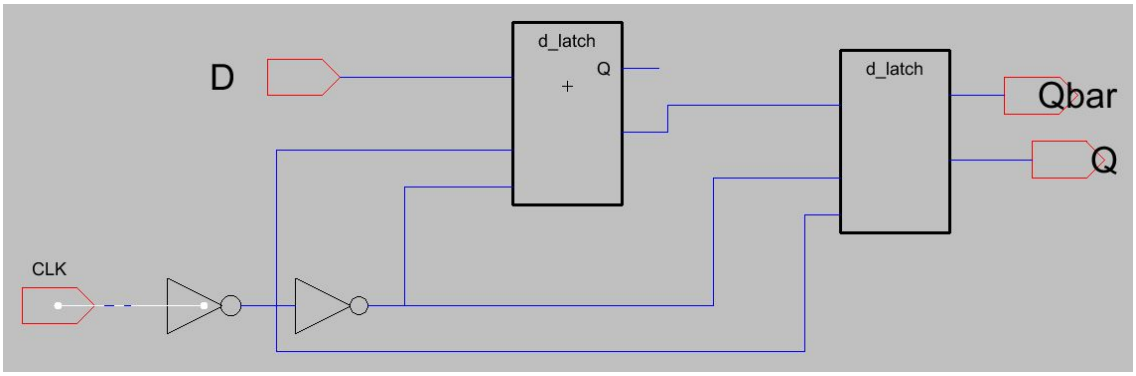

Figure 5: Data flip flop (DFF) - Schematic

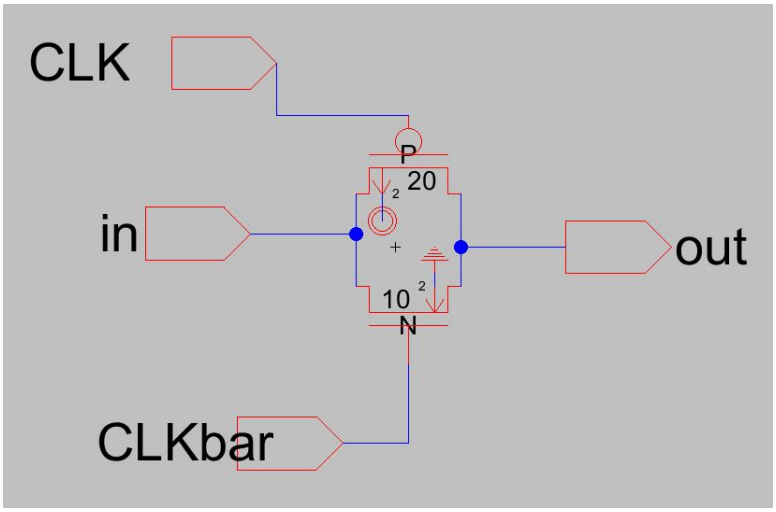

Figure 6: Transmission gate - Schematic

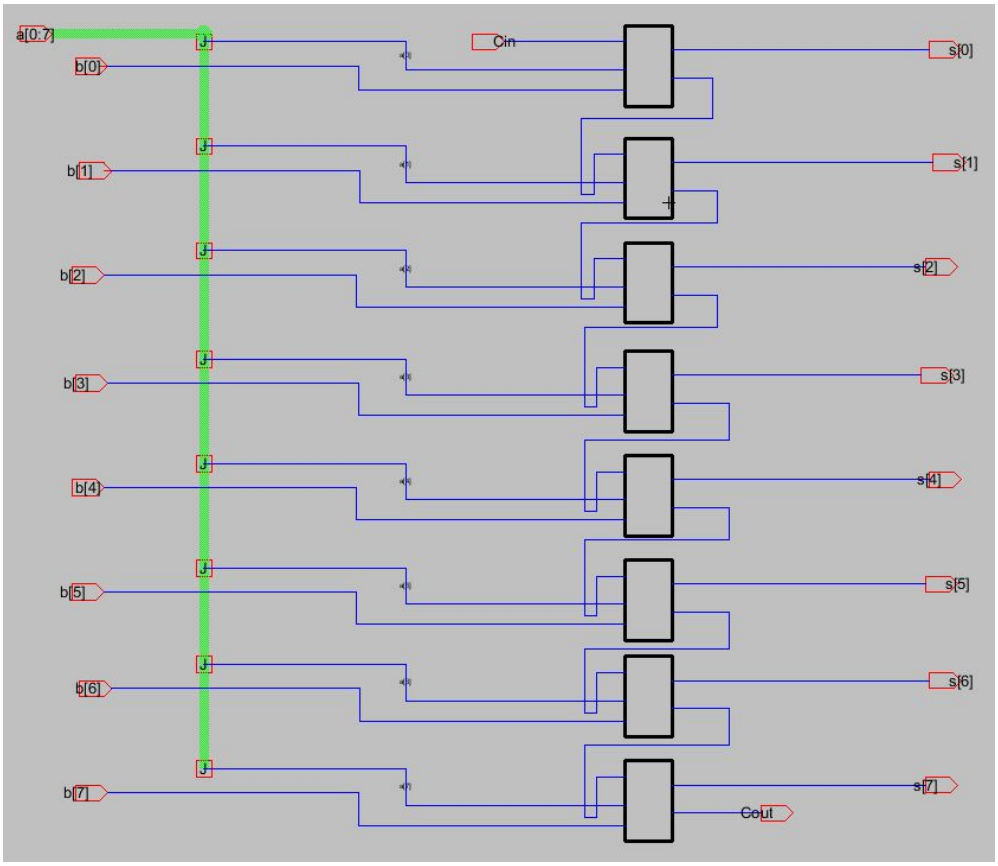

Figure 7: 8 bit ripple adder - Schematic

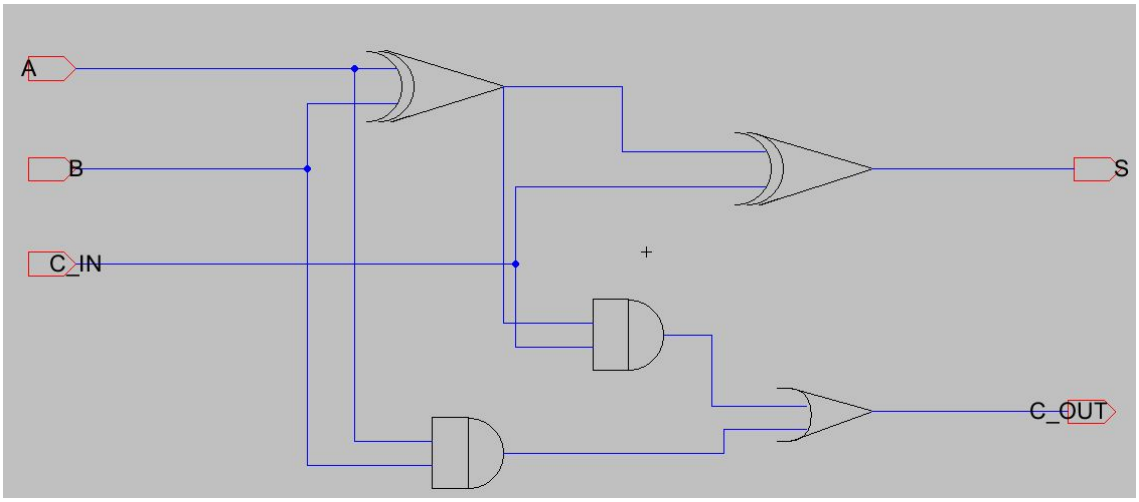

Figure 8: Full ripple adder - Schematic

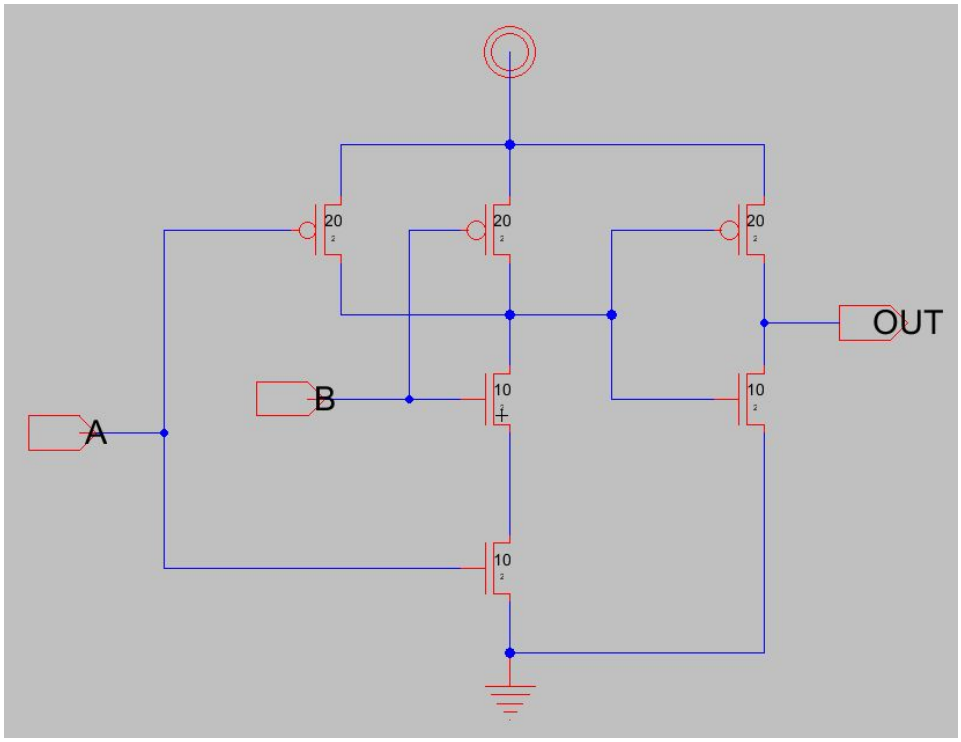

Figure 9: AND Gate - Schematic

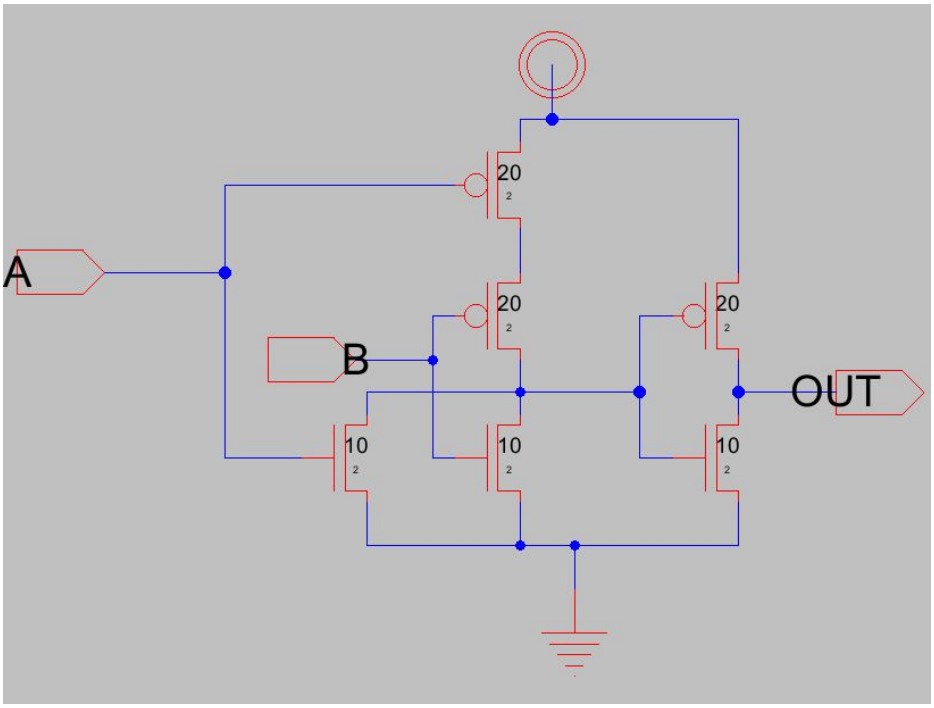

Figure 10: OR Gate - Schematic

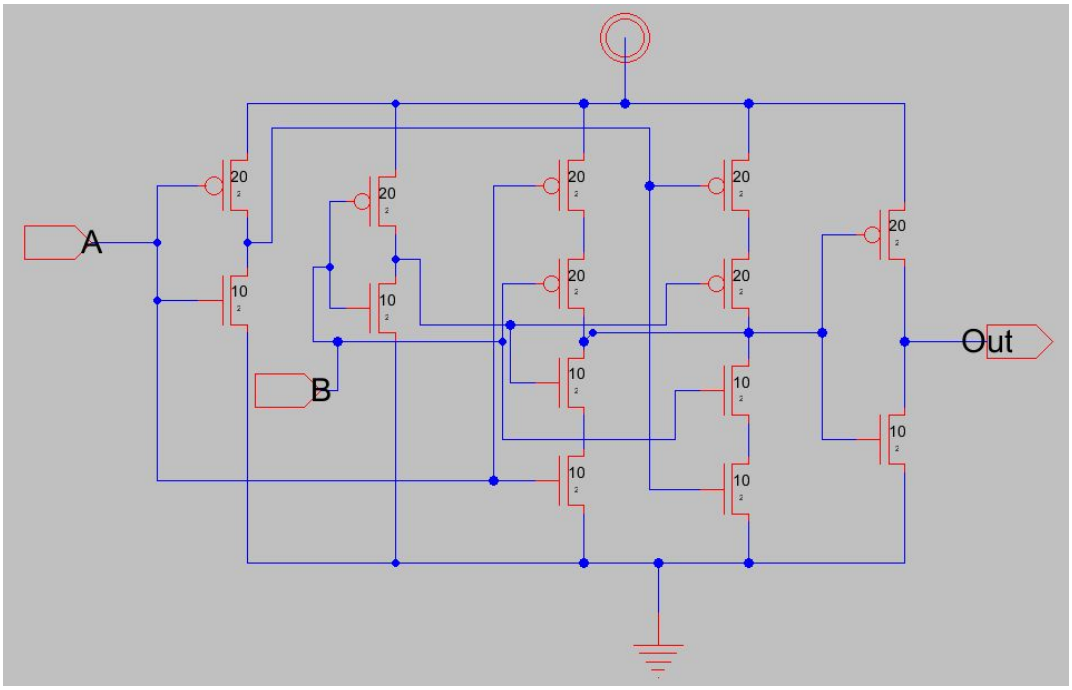

Figure 11: XOR Gate - Schematic

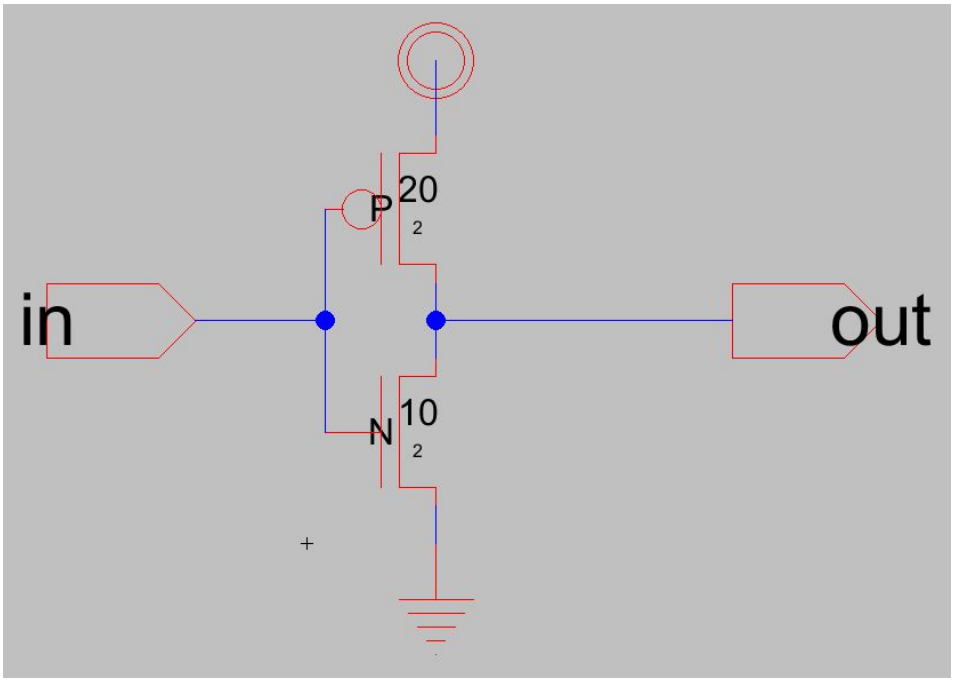

Figure 12: Inverter gate - Schematic

Figure 13: Clock to Q delay for NCO: 0.385 ns - Schematic

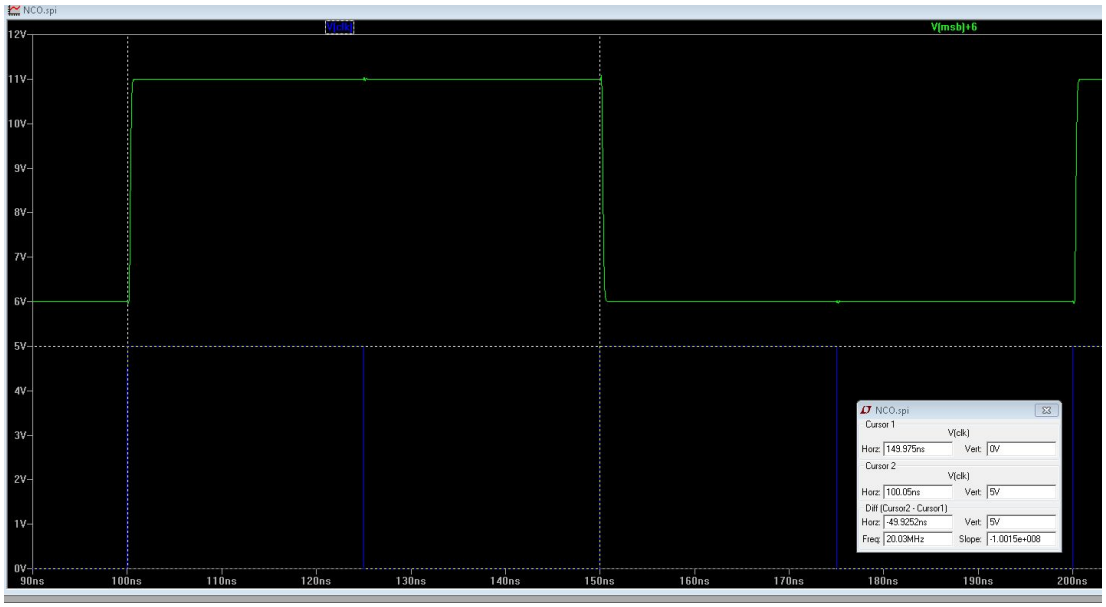

Figure 14: Waveform verifying 20Mhz functionality - Schematic

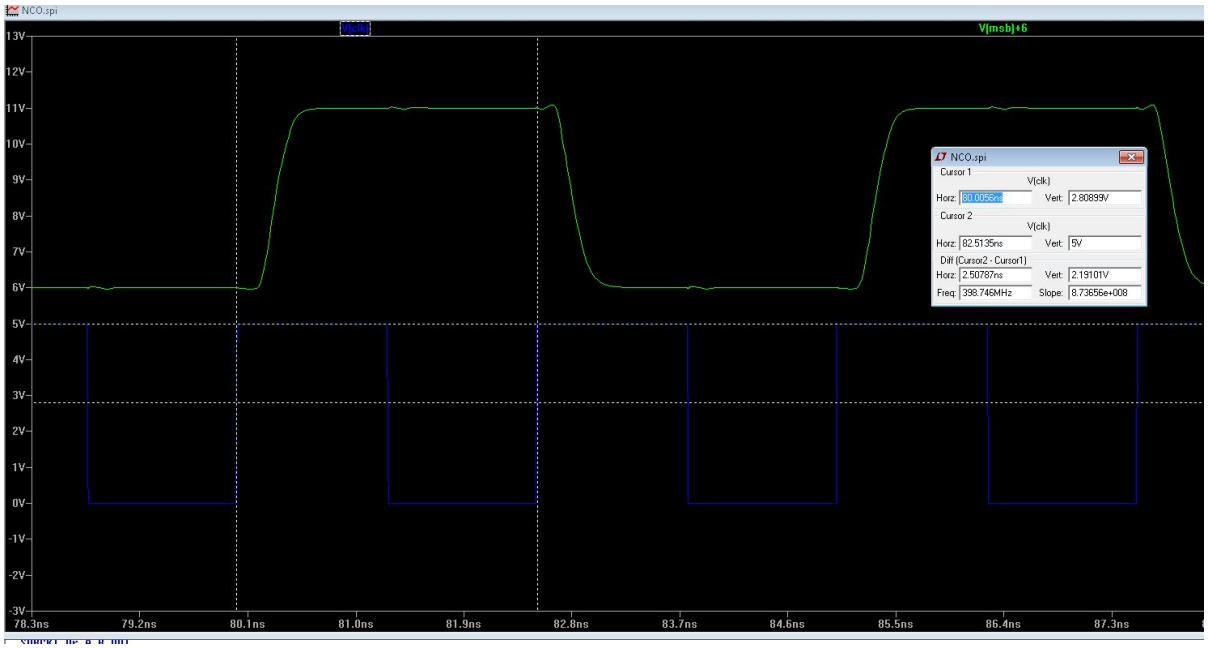

Figure 15: Waveform verifying 400 MHz functionality - Schematic

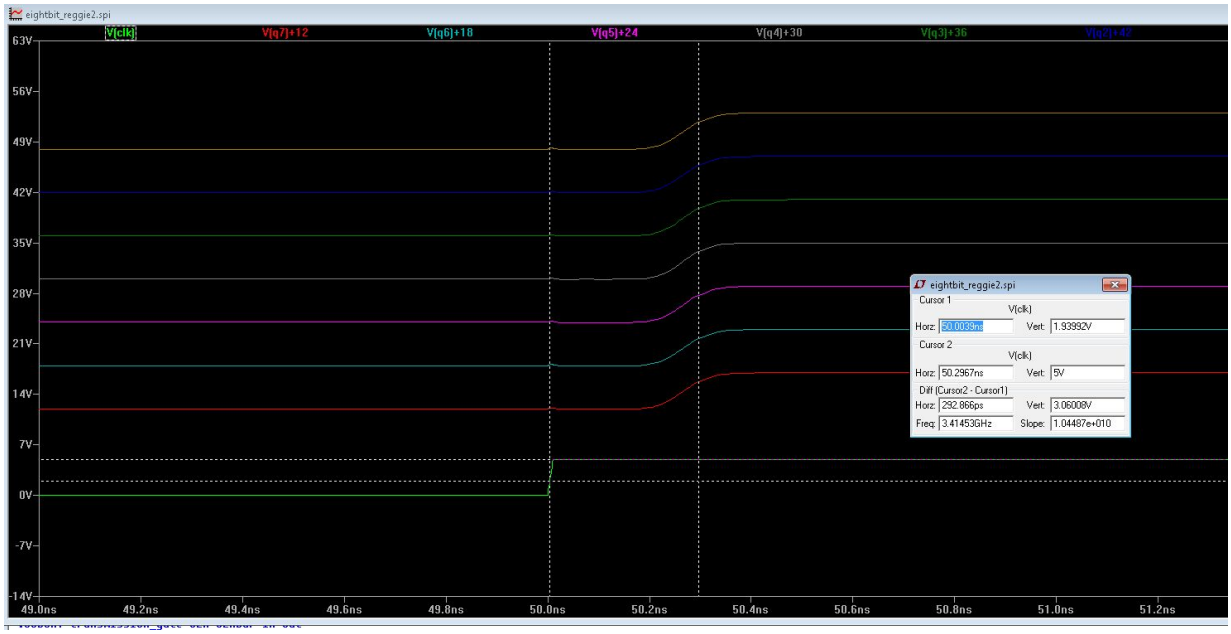

Figure 16: Waveform verifying 8 Bit Register functionality - Schematic

Figure 17: 8 Bit Adder propagation delay: 1.34 ns - Schematic

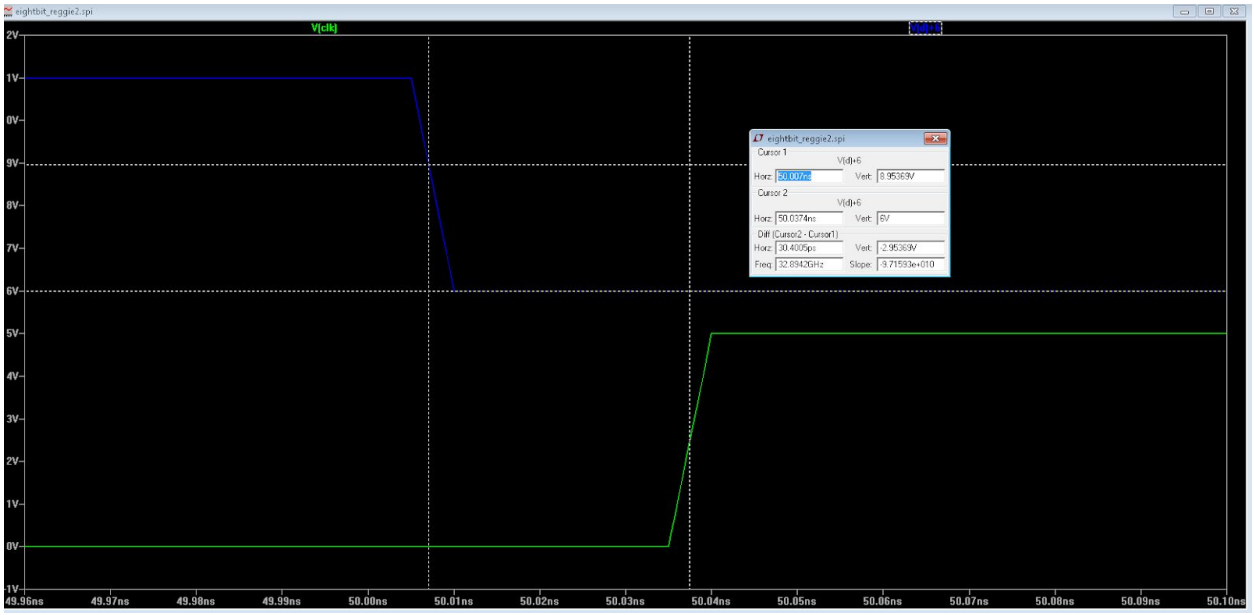

Figure 18: 8 Bit Register Setup time: 30 ps - Schematic

Layout

Total Die Size Area:	0.2597 um^2
Dimensions of minimum sized rectangle:	1951.1 x 1478.75 with 300.0 nm scale

We designed the layouts hierarchically so we started at the transistor level. It was essential to have the transistor level schematics so that we could determine where the drains, sources, and gates should be positioned in the layouts in order to optimize die area as well as the number of transistors used. For example with the AND gate, the two NMOS transistors in series became one transistor in the layout by connecting the n-active regions (along with the source connected to ground and the drain connected to output). For the PMOS transistors in parallel, the n well regions were connected by a contact between them and then having the sources of each transistor go to Vcc and the center contact in the n well region as the output. The outputs of the PMOS and NMOS transistor networks were connected to the same node. Again, this design was used to minimize die area and optimize transistor usage.

Once the gates were built, we were able to copy the existing designs into other layouts and easily build more complex designs including the data flip flop (DFF) and full adder. For both the adder and DFF, we first built a single bit slice and then copied the functional instance eight times to create the full layout. During the layout design phase, we set rules for each metal layer in order to simplify the design and prevent any arc crossings. For any given layer, all arcs contained ran parallel with each other, and in any adjacent layer (above or below), the arcs ran perpendicular. Again, this reduced any arc errors and simplified the design process.

We also ensured that all nodes were consistently labeled the same on every schematic to ensure seamless functionality.

Regarding the timing specifications of the layout components, see the specification table (Table 2) for details.

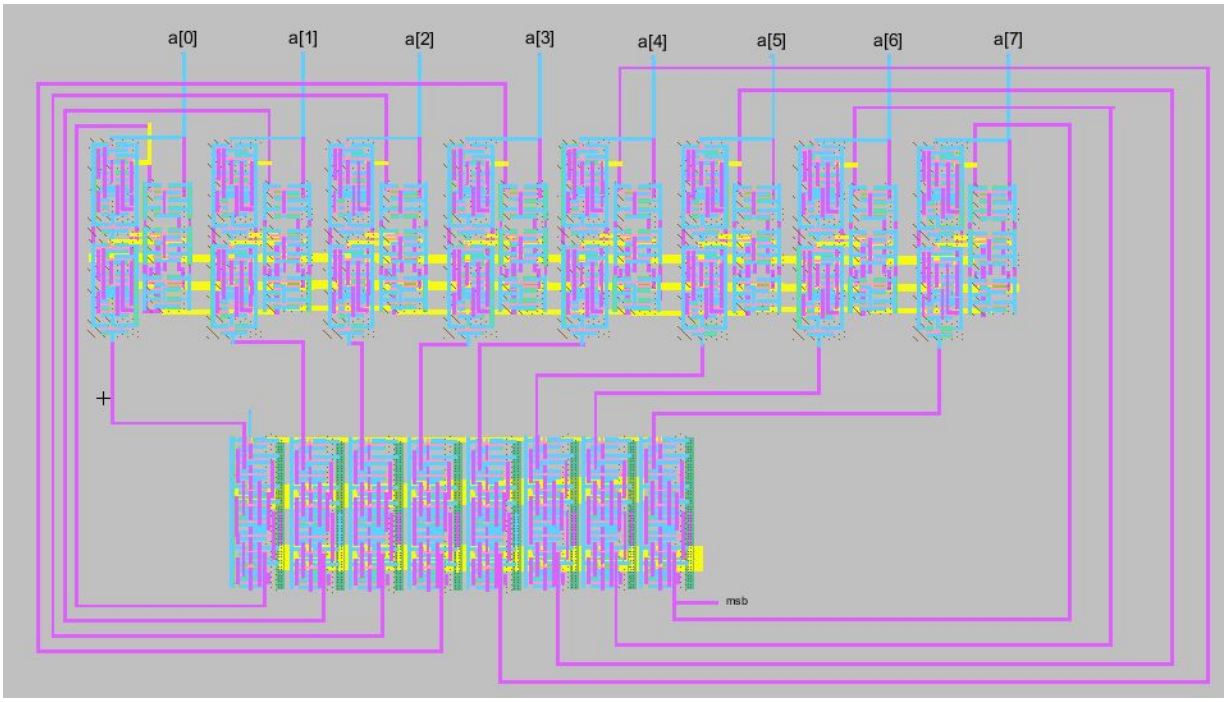

Figure 19: NCO Layout

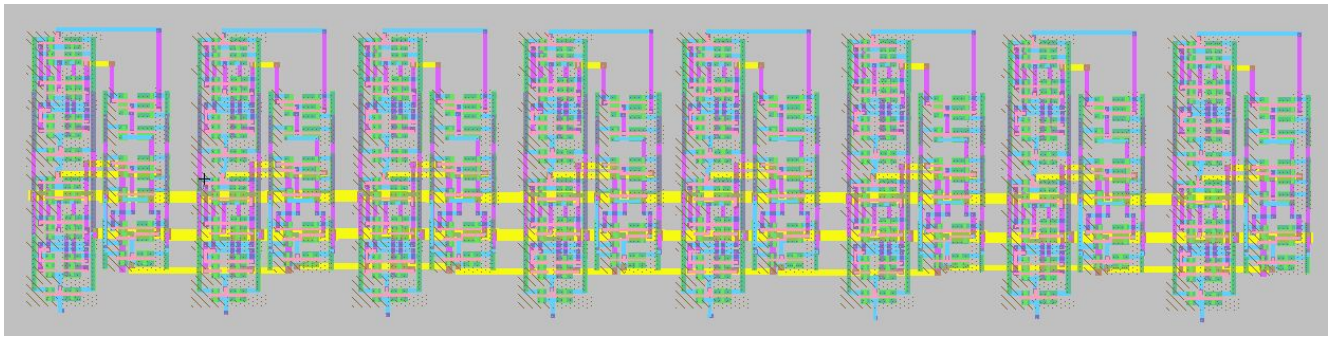

Figure 20: 8 Bit Adder

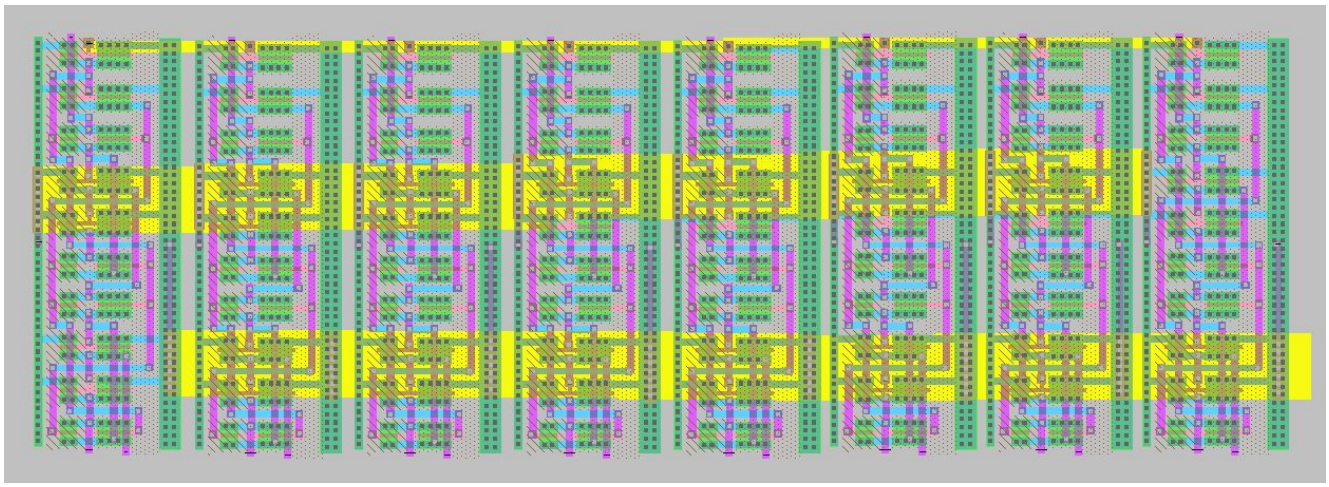

Figure 21: 8 Bit Register

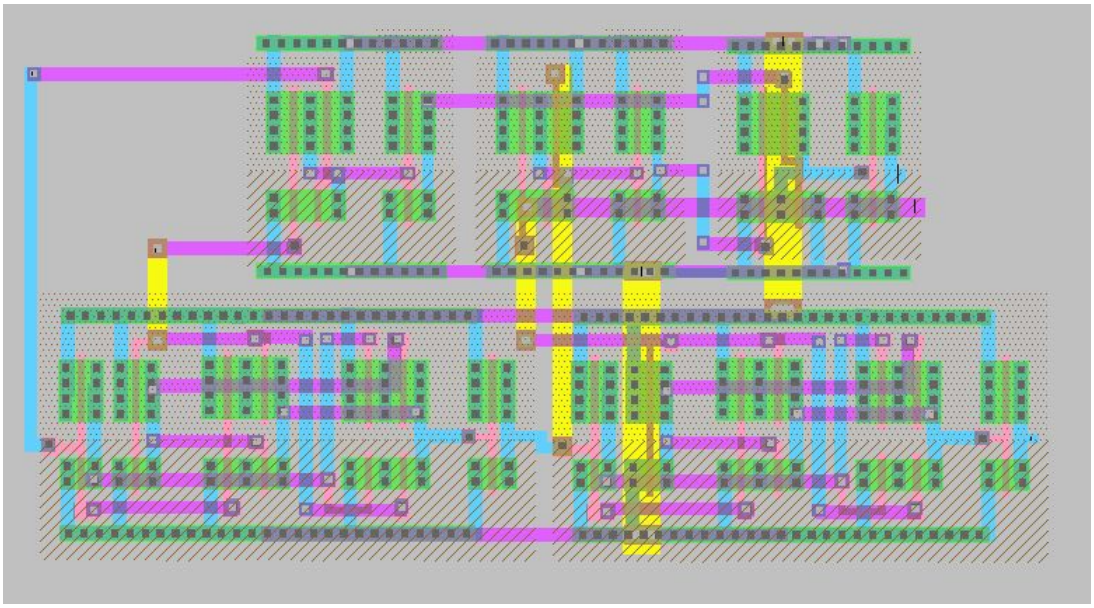

Figure 22: Full Ripple Adder

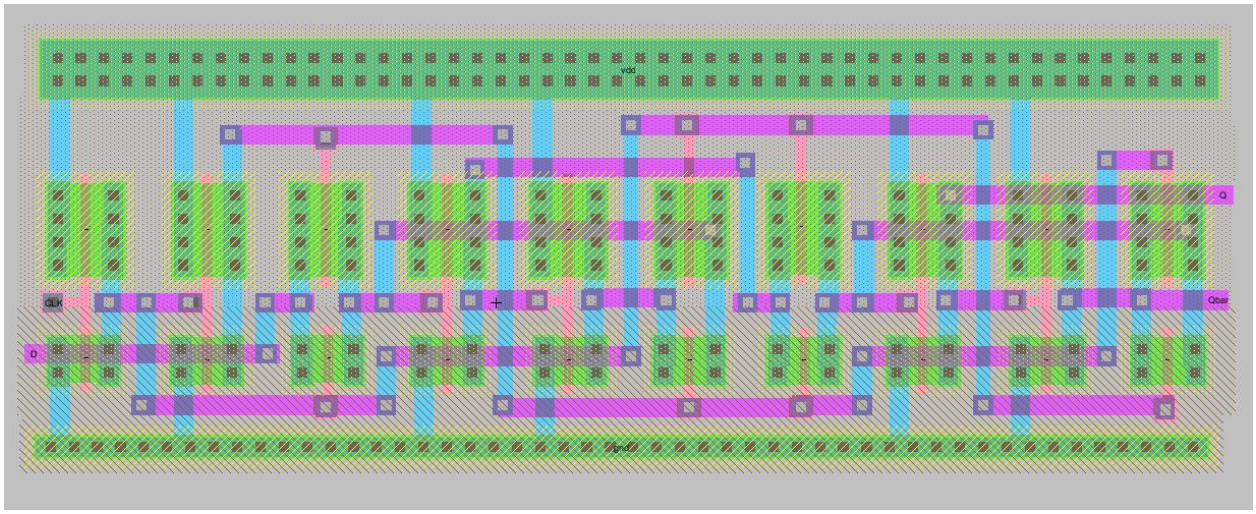

Figure 23: DFF layout

Post-Layout Simulation

Figure 24: Clock to Q delay NCO: 1.9ns - Layout

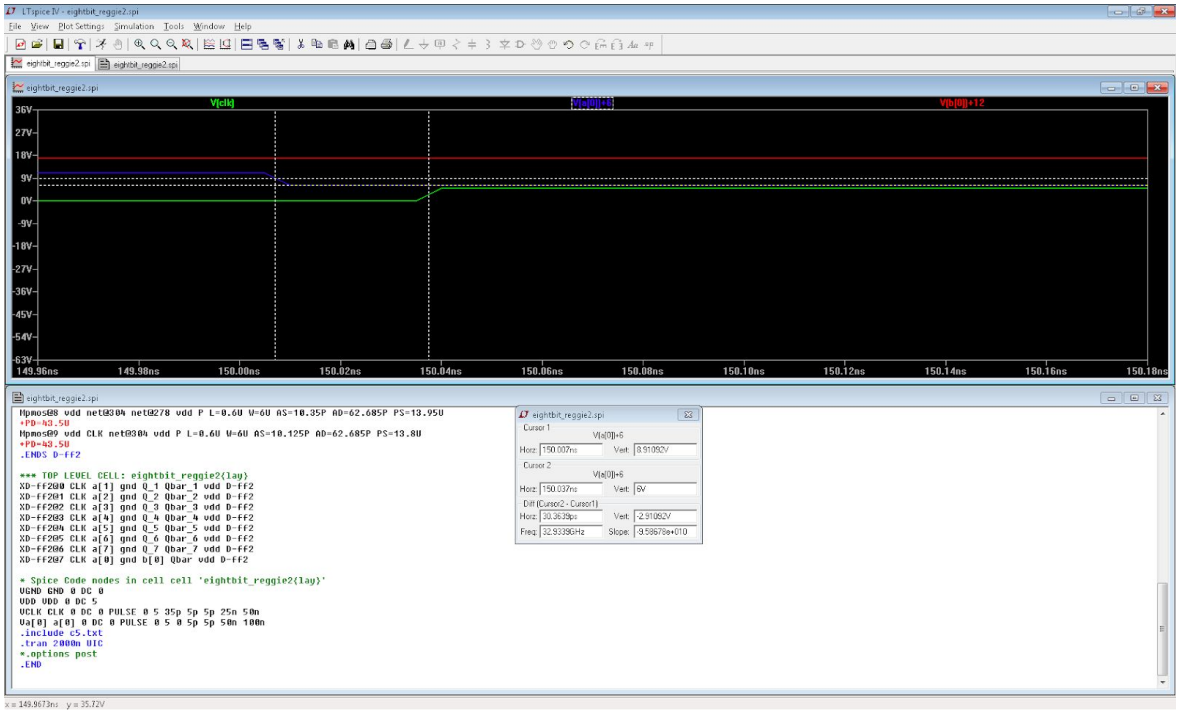

Figure 25: Minimum Setup time for register: 30.7ps - Layout

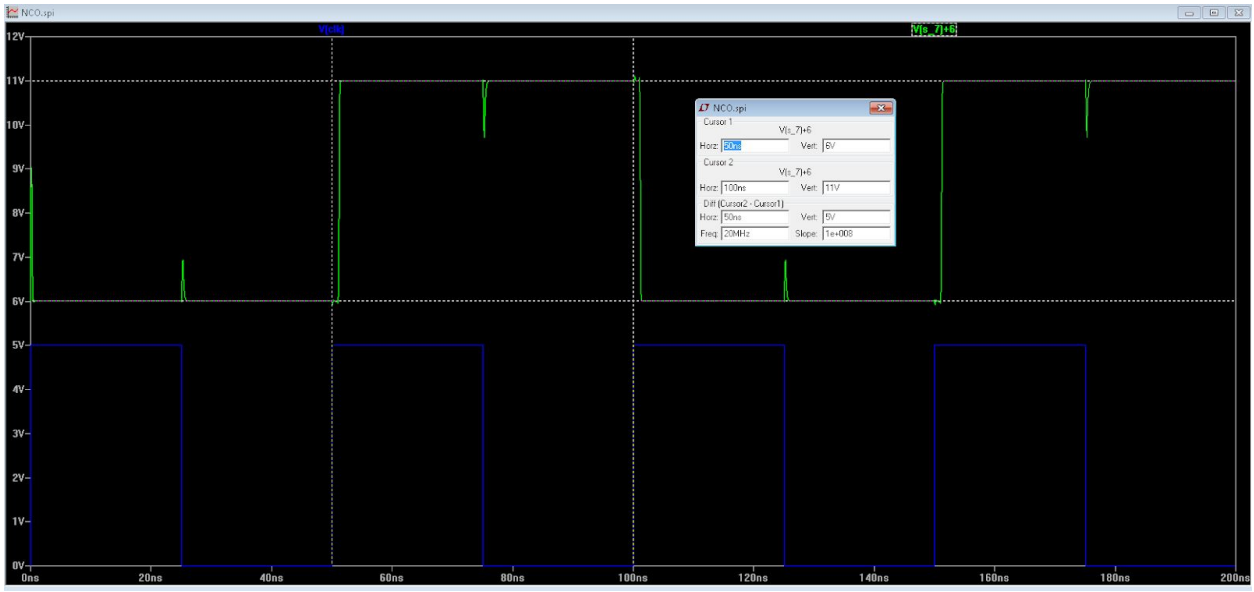

Figure 26: Waveform verifying functionality at 20MHz - Layout

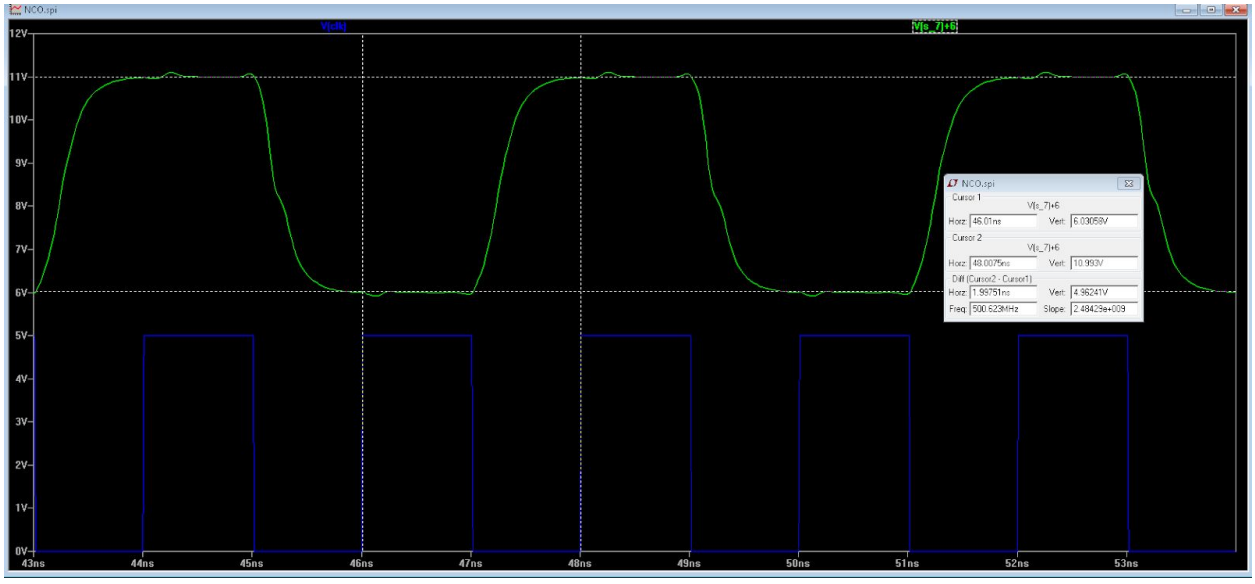

Figure 27: Waveform verifying functionality at 500 MHz - Layout

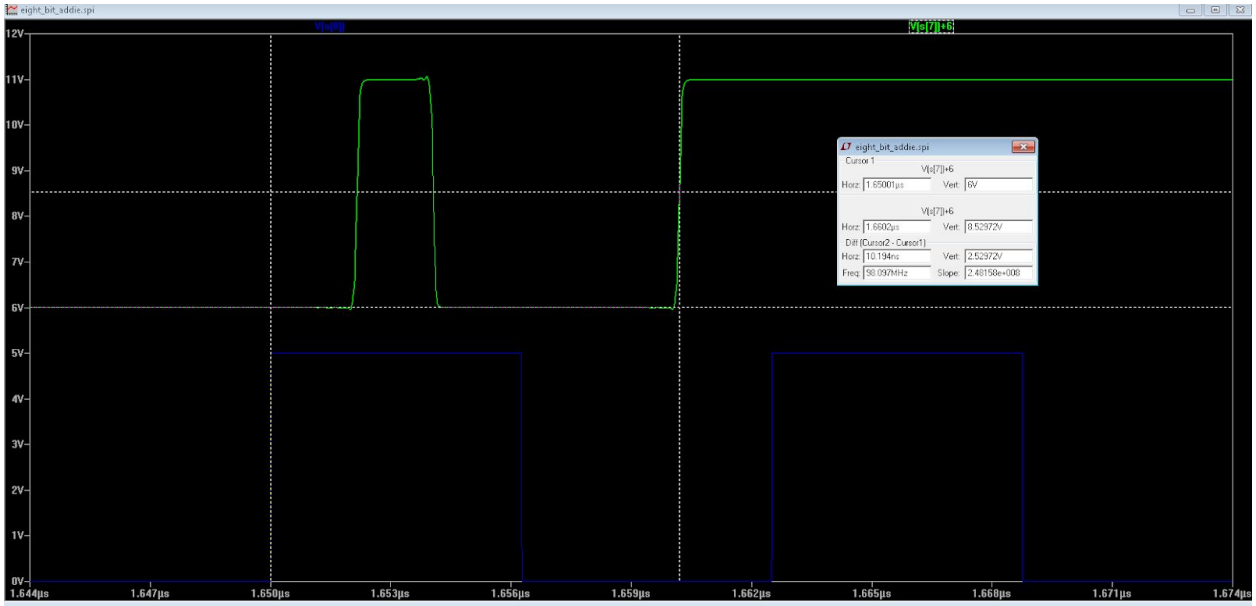

Figure 28: 8 Bit Adder propagation delay: 10.2 ns - Layout

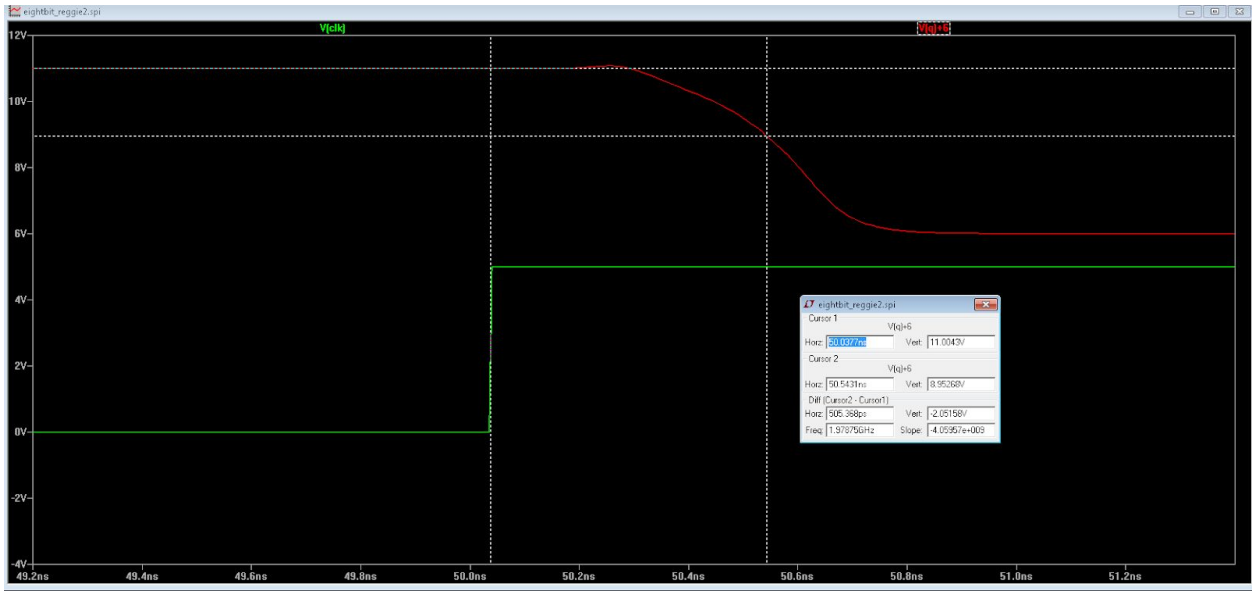

Figure 29: 8 bit Register Clk to Q: 0.55ns - Layout

The major timing differences between the Layout and Schematic simulations were seen in the Clock to Q delay for the NCO as well as the propagation delay through the 8 Bit Adder. The Clock to Q delay for the Layout was approximately 1.9 ns (Figure 24) while the Clock to Q delay for the Schematic was only 0.4 ns (Figure 13). This is about an 80% increase from the Schematic to Layout delay.

Additionally, we discovered a significant timing difference in the propagation delay through the 8 Bit Adder. For the Schematic, we measured a delay of 1.34 ns (Figure 17) while for the Layout, we measured a delay of 10.2 ns (Figure 28). This is an increase of almost 660%. We attributed all of the timing differences observed to parasitic capacitances present in the Layout design.

Project Commentary

The most difficult part of this project was designing in Electric. The computer software did not always cooperate and syncing up various libraries and designs proved to be troublesome. We had multiple instances where our layout design successfully NCO'd and then we would retest the design on a later date and Electric would tell us we had lots of errors. We never really determined the source of these issues. We solved them by opening and closing Electric and re opening the relevant libraries and eventually things would fix themselves.

The most time consuming portion of the project was verifying the functionality of the 8 bit adder as well as the layouts. There are so many different places where mistakes could be made that it was a time consuming process to make sure everything was perfect. Especially with the adder, we had to analyze waveforms with 25 different inputs and outputs and there was never a convenient method of testing (none of us knew Python). The layouts also took a considerable amount of time because Electric has many rules and specifications and it is not easy or convenient making sure every rule is met. We also encountered many small errors in the layouts where we thought things worked for a particular case and then would test another and encounter issues. Overall, we found being extremely attentive and flogging every aspect of our design with testing as the most effective method for ensuring functionality.

When moving to the layout from the schematic, we didn't have to change much. We began with the layouts for the basic logic gates, transmission gates, the full adder, and then worked our way up to the 8 bit adder and register.

Regarding the work breakdown, Dan Taylor and Zhengtai Zhong initially worked on the registers and layouts and Matt Janke, Evan Rose, and Nick Ackerman did most of the initial work with the adders. However, once the project reached the final stages of testing, all group members worked collaboratively to troubleshoot and debug. Additionally, Matt Janke took the lead on compiling the report and was supported by Zhentai Zhong and Nick Ackerman with measuring the timing specifications of the NCO while Dan Taylor and Evan Rose worked extensively on finalizing the layout design.

The next time we have a large group project, we would first determine exactly what level of performance was needed before jumping into the design phase. Without doing any baseline tests, we initially decided a carry-lookahead adder would be best however, after extensive issues troubleshooting the design, we switched to a ripple adder which proved much easier to build. If we were to do this again, we would be smarter in our brainstorming and not make our lives any harder than they need to be.