MADVLSI Final Project: 1.8V to 3.3V Boost Controller

Will Fairman, Jillian MacGregor, Bryce Mann

May 11, 2021

1 Introduction

The goal of our final project was to capture, simulate, and layout a DC-DC boost converter controller. Our final product is capable of boosting 1.8V up to 3.3V at a load of 1A. Additionally, users of this controller are able to select both the switching frequency of the circuit and the control scheme (either voltage or current mode control) through 3 external components. This report will detail both control architectures we implemented, simulation results for the critical sub-circuits involved, simulation results for each control scheme, and an implemented layout.

1.1 Code Repository

https://github.com/wFairmanOlin/MADVLSI_final

2 Control Schemes

Our first step in completing this project was to get a sense of how we could implement a control scheme to keep the output voltage at the value we wanted. Through our research we learned about and decided to explore both voltage and current mode control. In order to start schematic capture and simulation with a solid background we started by using PLECs to create boost circuits with both control schemes so that we could verify our understanding of the systems and get a better sense of what components would be required to make the controller work.

2.1 Voltage Mode Control (VMC)

We started with VMC because it is easy to extend to current mode control and we wanted to try and implement different control schemes. Our final PLECs simulation harness for VMC can be seen below in figure 1.

For this control scheme, the PI controller is fed the difference between a known reference and a scaled down output voltage, through a high resistance voltage divider connected between the output voltage and ground. This PI controller will then output a value fed into the positive input of a comparator, where the negative input is some type of triangle or sawtooth wave set to a high frequency. Since the comparator output is fed directly into the power MOSFET's gate, the PI controller output will determine the circuit duty cycle. We had to take care in both PLECs and when simulating in xschem that this PI value would stay at reasonable values within our power rails, so that the circuit wouldn't get stuck with the MOSFET always in a single state. We pulled all of the external component values from PLECs into xschem when simulating our controller, but due to the non-ideal nature of some of the components and sub-circuits we made (specifically the op-amps involved in the PI controller) our integral controller coefficient did not end up matching what we simulated in PLECs, although it did still work.

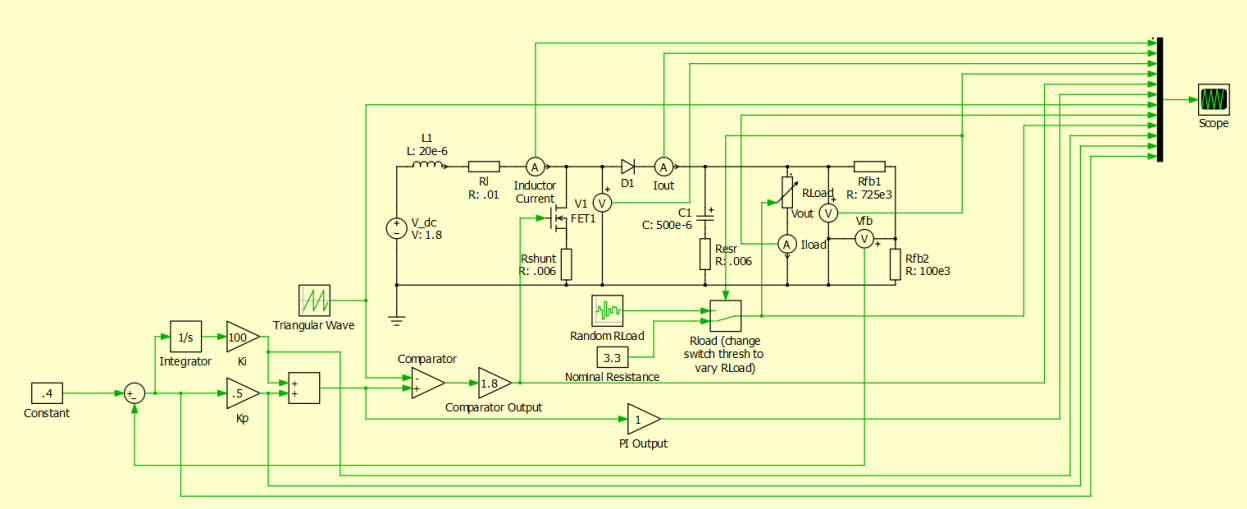

Figure 1: PLECs Voltage Mode Control schematic.

2.2 Current Mode Control (CMC)

Once our VMC simulation was working in PLECs we moved on to implementing CMC; our PLECs schematic can be seen below in figure 2.

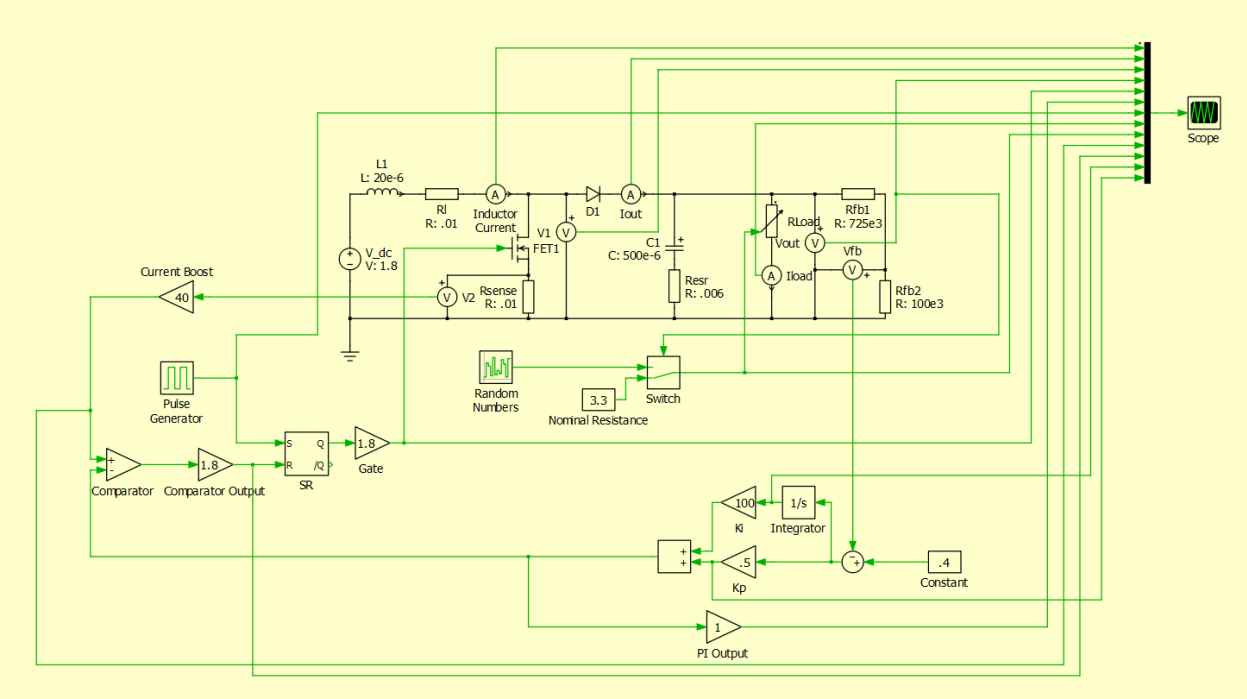

Figure 2: PLECs Current Mode Control schematic.

The reason CMC was relatively simple to implement after VMC is because the voltage feedback portion of the circuit is exactly the same (in both xschem and PLECs we are able to switch between VMC and CMC without changing any of the voltage feedback loop parameter values). In this control scheme we can set a small sense resistor below the power MOSFET and amplify the voltage at that node before feeding it into

the positive input of a comparator. The negative input of the comparator gets the voltage feedback loop PI controller output, causing the comparator to output high when the current through the power MOSFET reaches a certain level compared to the voltage feedback. This then resets an SR latch whose ”set” input is being fed with thin pulses at a high frequency. This latch then controls the duty cycle of the power MOSFET. By directly comparing the feedback voltage and the current through the power MOSFET in the feedback loop a more robust control system is possible, at the expense of more complexity.

By starting off with a PLECs implementation we were able to get a sense of exactly what sub-circuits we would need for our system:

- Voltage subtraction and addition
- PI Controller
- Amplification and Comparison
- Temperature and supply voltage independent voltage reference (bandgap)
- Switching Circuits
- Miscellaneous Digital Logic

In order to realize these sub-circuits we needed a mixture of analog, digital components, and passive components (both internal and external to our controller).

3 Sub-circuit Schematics

3.1 Op-Amp Circuits

The most important and widely used component involved in our controller was a self-biasing, rail-to-rail op amp. This schematic was provided by Brad and worked very well for us. Of the 6 bullets above, the first three were implemented exclusively with op-amps and passive components. Amplification and comparison are pretty self explanatory, so we won't go into the details on those but they are present and necessary for the system to work.

Note before starting these sections that for LVS purposes our op-amp symbol in these images is without any VP or VN ports. Within the op-amp schematic we set the positive rail to Vdd and the negative rail to GND so those ports were not necessary.

3.1.1 Error Amplifier

To implement the subtraction block from the PLECs simulation to feed into our PI controller we used an op-amp circuit. We started with the typical difference amplifier that we first analyzed in ISIM, but after doing a bit of research into why our output voltage wasn't as stable as we would like it (VMC compensation and CMC compensation), we implemented a compensated error amplifier circuit instead. The circuit and passive values we ended up using are below in figure 3 and work well in both VMC and CMC.

Starting with this error amplifier and ending with the summing amplifier after the PI controller, all of our op-amps in the control loop are referenced to 400mV (denoted by Vref in figure 3 and by the constant .4 in the PLECs schematic). This allowed us to keep our control loop calculations in the middle of the op-amps' rails where they are most accurate. This is also a fairly easy voltage for our bandgap reference to produce.

In isolation this circuit and its simulation results aren't the most enlightening (which is a bit of a theme for most of these op-amp circuits within the control loop), but when we evaluate the entire circuit results we will be able to see that it functions properly within the larger context of our system.

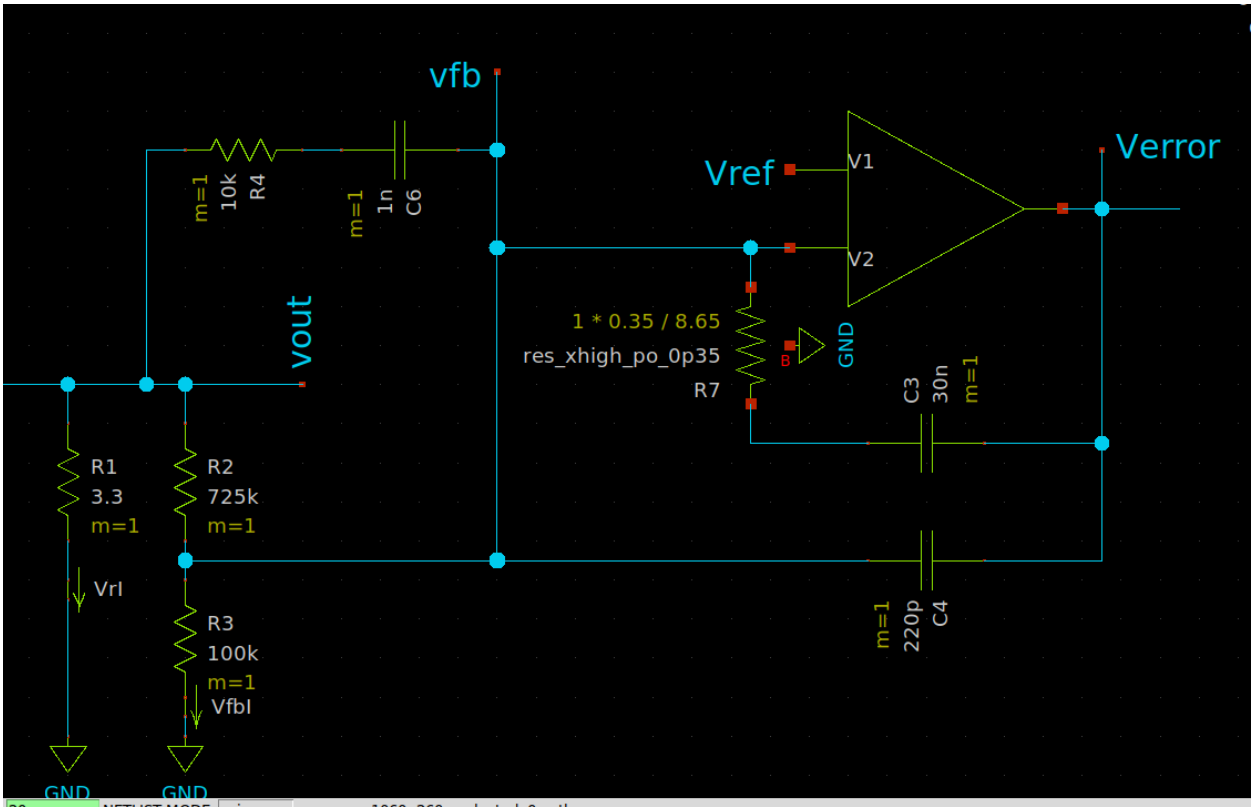

Figure 3: Compensated error amplifier using one on chip resistor (R7) and 4 external components. We decided that R4 and C6 would remain off-chip so as to not bring a voltage higher than 3.3V onto the chip (since our skywater models are only rated for up to 1.8V.) C3 and C4 are off-chip because for capacitor values in the 100s of picofarads and above we saw that they would end up being bigger than all of our other components combined and wanted to avoid that during layout.

3.1.2 PI Controller

The main portion of our control loop is the PI controller. We implemented an ideal continuous controller in PLECs, and found that it was fairly challenging to convert that into a reliable circuit, but we got there eventually. Our PI controller sub-circuit is below in figure 4, where the proportional control is handled by the 2 op amps on the bottom, and the 2 on the top take care of the integral control.

Looking at the two parts to the controller, we can see that we were able to use the same proportional coefficient as in our initial PLECs simulations; the buffered error signal goes through a voltage divider (referenced to .4V) with equal resistor values before it is buffered again and sent to the next portion of the control loop.

The integrator on top was much harder to get to behave, but after some trial and error and studying the waveforms on the integral terms we were able to settle on component values that work pretty well. Each stage of the integrator block is referenced to .4V, with the first op amp doing the actual integration. The nature of this circuit is such that it's output is an inverted version of the integral of the input (integrated around that reference voltage), so the op-amp in the top right is wired up as an inverter as well so the Vint output signal isn't inverted relative to the error signal anymore. Each op-amp in the integrator has a DC gain of 5; the total gain of the integrator could not get as high as it was in the PLECs simulation without causing the op-amps to saturate to a power rail immediately.

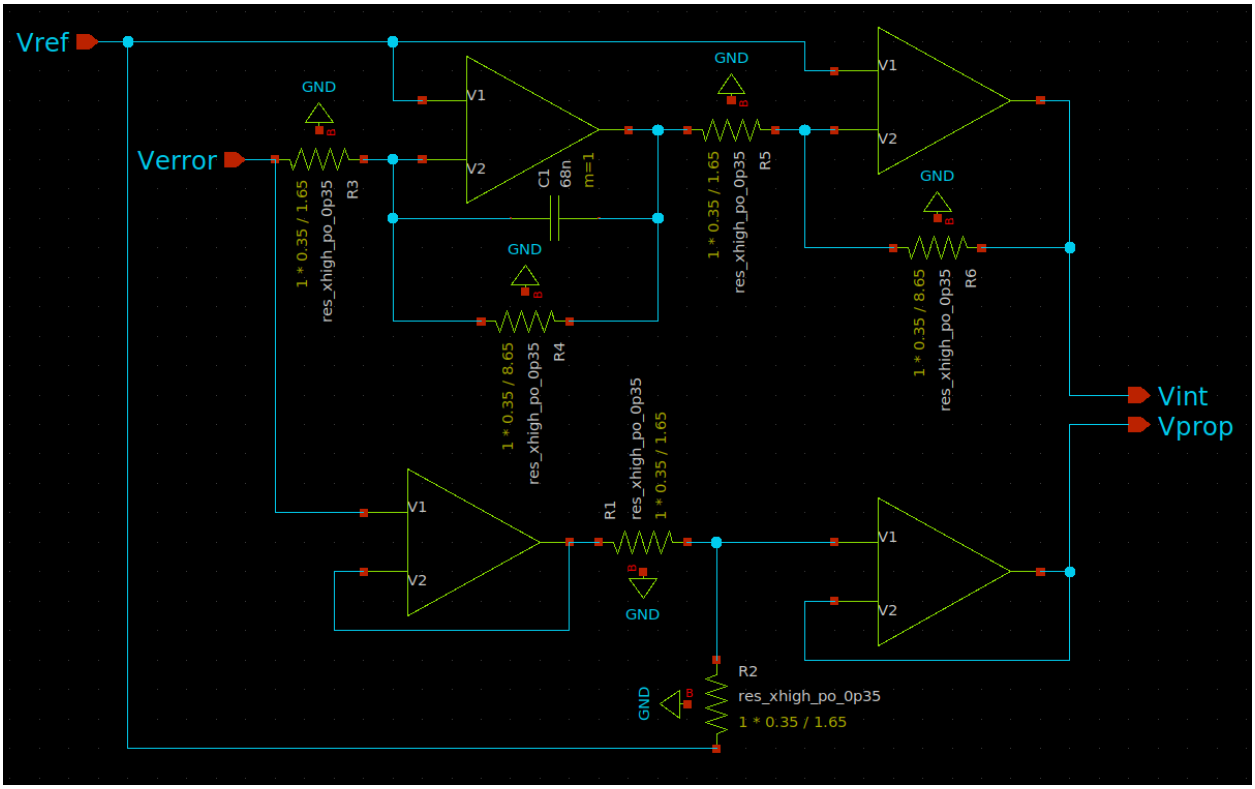

Figure 4: PI controller subcircuit. C1 is the only off chip component, as 68 nanofarads was too big for us to bring into our layout.

3.1.3 Summing Amplifier

In both VMC and CMC, the outputs of the proportional and integral controllers must be added back together, which we implemented with another op-amp circuit, seen below in figure 5.

This is a very standard and well analyzed circuit, so the only thing to note here is that this is the last stage of the control loop that is referenced to .4V. The output voltage expression for this circuit contains a "-Vref" term, so before sending the PI output to the comparator (in either control scheme) we can use that Vref to shift our PI output back down to be referenced to GND, as opposed to inside the PI controller where everything was referenced to .4V.

3.2 Temperature and Supply Voltage Independent Voltage Reference

We rely on several reference voltages in our PI control loop. If these references vary, the math in the control circuit will be modified and the controller could potentially go unstable. Because of this we need to be able to generate a series of voltages that are stable across a wide range of temperatures and power supply voltages. To generate these voltages we recreated a common bandgap voltage reference circuit shown in 9. This circuit in particular is from "A CMOS Bandgap Reference Circuit with Sub-1-V Operation": which also served as the source for most of our knowledge about this circuit.

The primary mechanics of the circuit involve utilizing temperature dependent pnp transistors to create two voltage over temperature curves with one increasing in voltage as temperature increase (PTAT) and the other decreasing in voltage as temperature increases (CTAT). These two curves can then be summed together to create a output voltage that is flat across a range of temperatures. It is important to note, however, that if the slope of the PTAT and CTAT curves differ, the final output voltage will still change with temperature. In order to manipulate the curves to create the smallest change in voltage over temperature, we modified the resistors values in figure 9. The output voltage of our bandgap circuit is shown in figure 7.

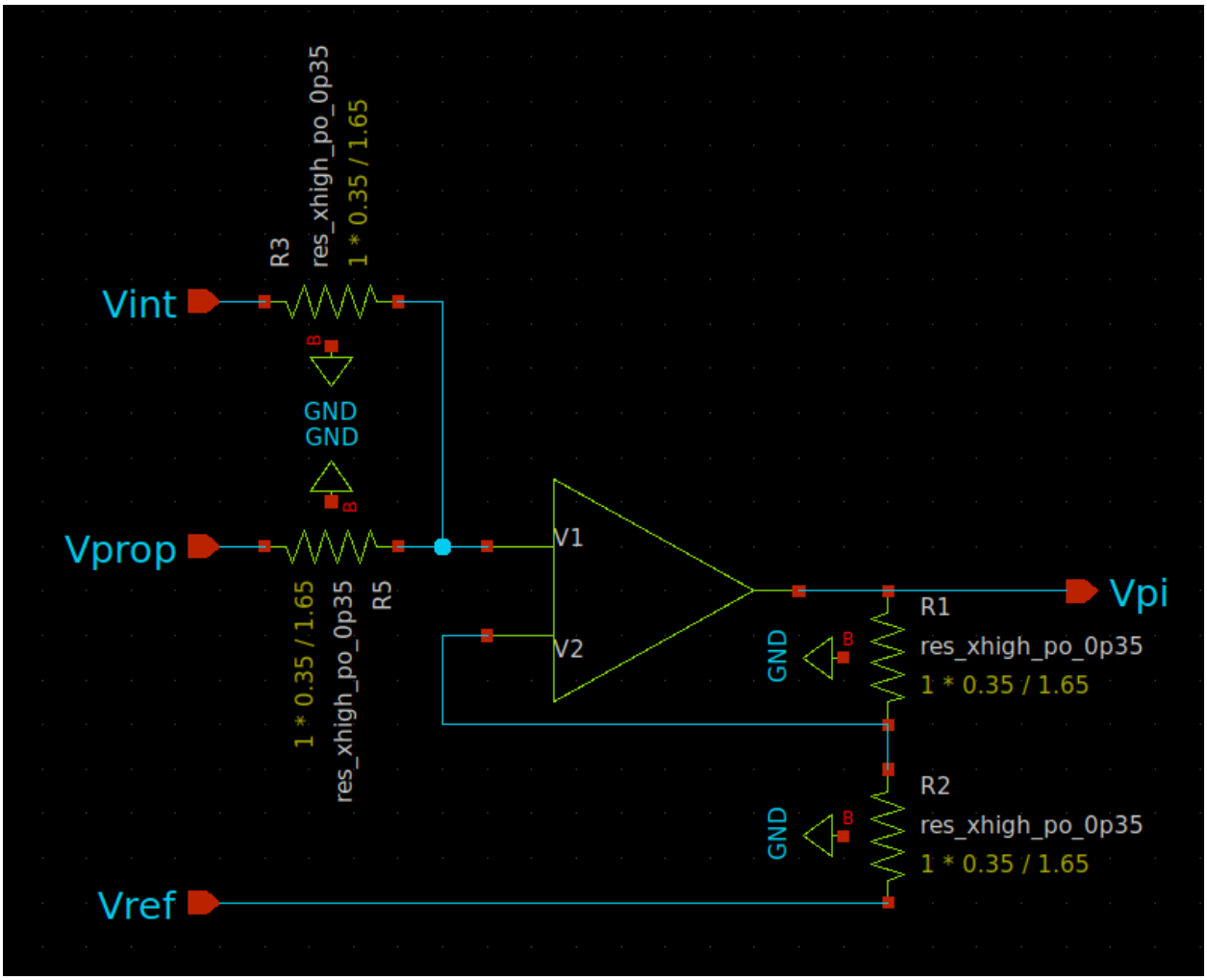

Figure 5: Summing op-amp circuit for combining proportional and integral controller terms.

While we modified these resistors to make the voltages at temperatures $100C$ and $-40C$ nearly identical, the plot in figure 7 references the layout version of the schematic which has slightly different resistance values than our original design, causing the voltage at 100C to be a few milivolts high. The overall parabolic shape of the temperature response is common for the simple bandgap circuit we chose for our project: more complex designs include circuits to cancel out the parabolic curve as well. Overall, we noticed that our circuit was fairly sensitive to the chosen resistor values. If we wanted to make the control circuit more robust, increasing the area of the resistors used in this design would decrease the potential variation in resistance values.

In addition to temperature, we simulated the output voltage of the bandgap circuit across different power supply voltages. Figure 8 shows that the output voltage is unstable until the power supply crosses over $1.4V$ after which the voltage levels off at $1.08V$.

To get to the reference voltage values needed in the rest of the circuit ($800mV$ and $400mV$), we attached the output voltage to a simple resistive voltage divider circuit with opamp voltage buffers shown in figure 9.

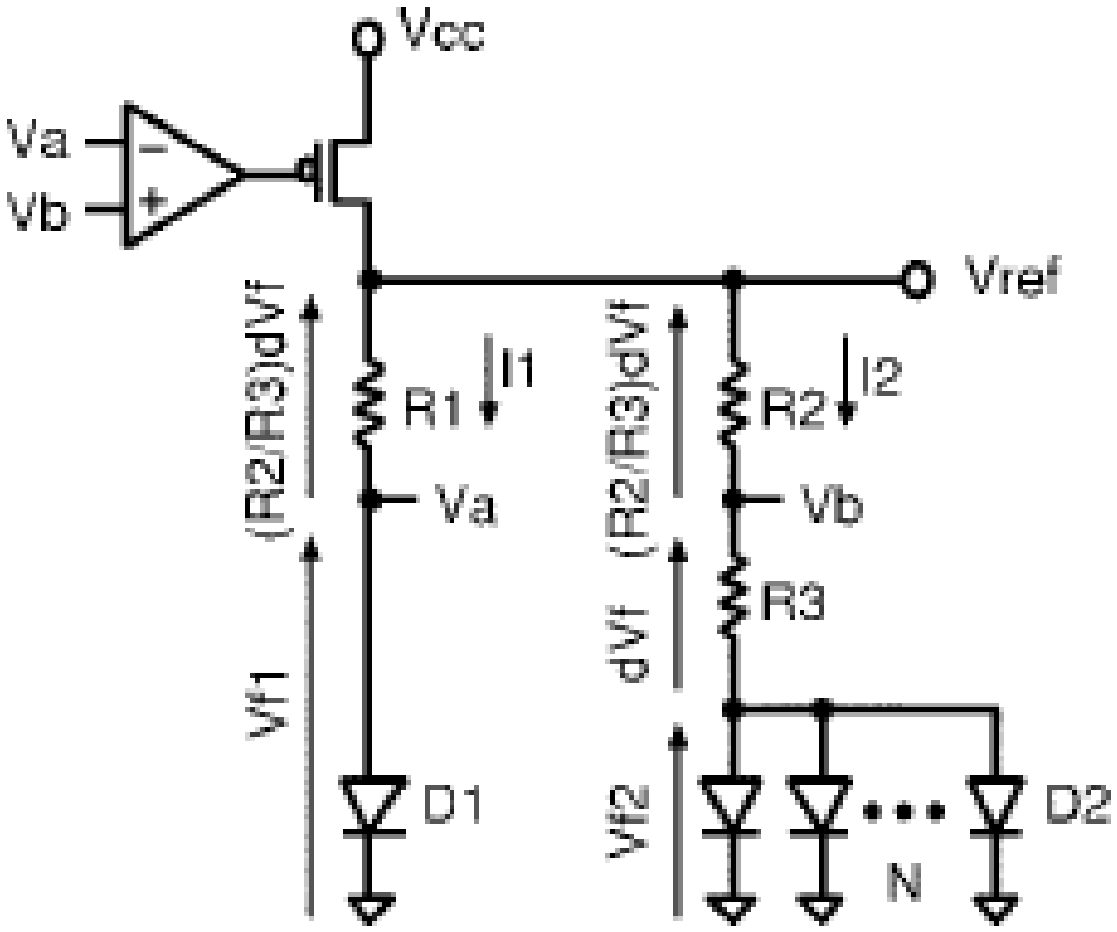

Figure 6: Simple bandgap voltage reference circuit.

3.2.1 Transistor Level Schematic

3.3 Switching Circuits

One of our goals at the outset was to not rely on any external voltage or clock signals for the switching action of the circuit. Through research into multivibrators and relaxation oscillators we chose to implement a self oscillating square wave generator using schmitt inverters.

3.3.1 Transistor Level Schmitt Inverter

Having never designed a schmitt inverter before, we used this paper on schmitt triggers as a guide and added an inversion stage to create the inverters we needed. The schematic and transistor geometries we used are in figure 10; those sizes and parameters were chosen to make the low-to-high and high-to-low threshold voltages as close to each other as possible, which allows for more symmetrical square wave generation out of the oscillator.

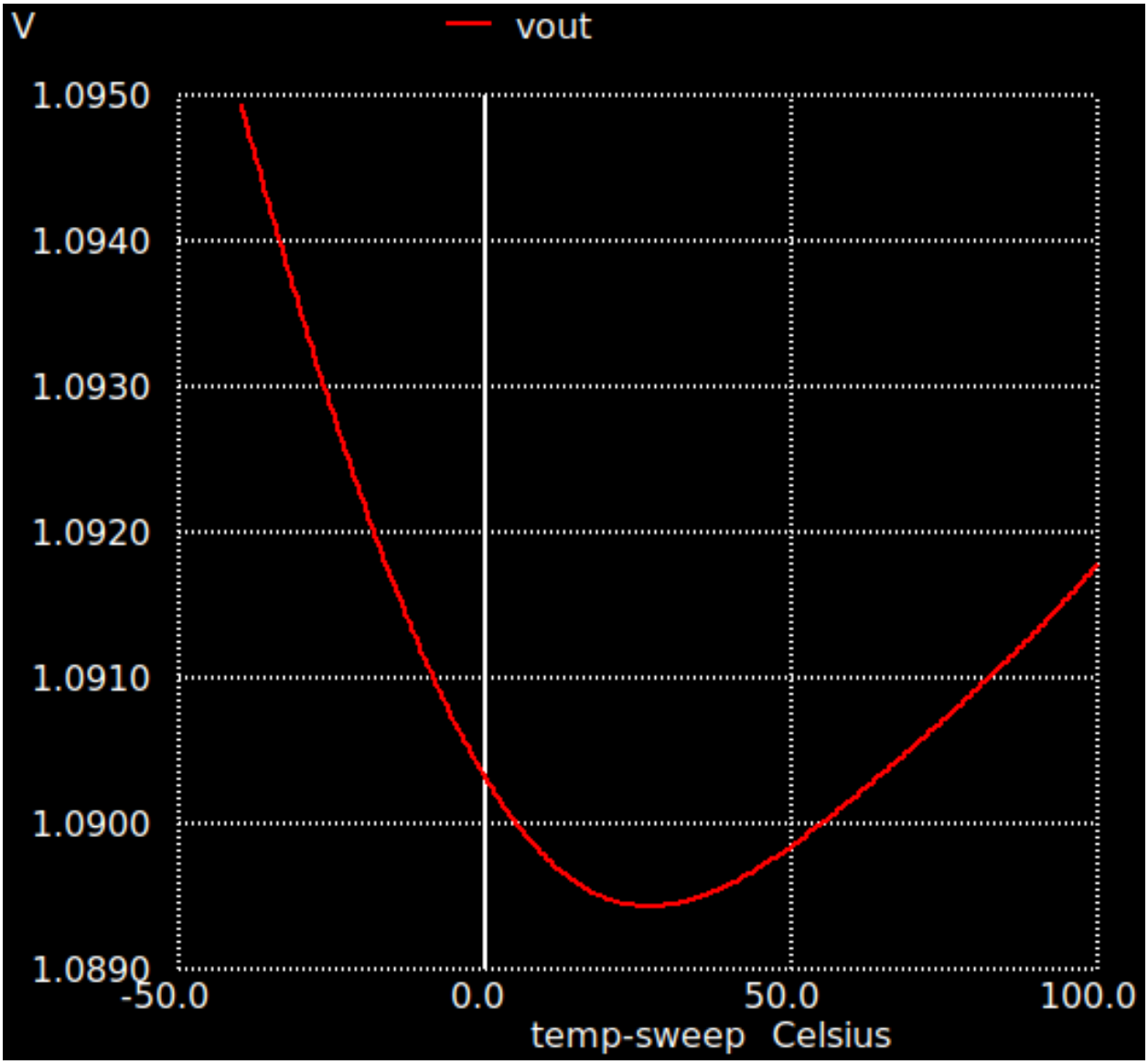

Figure 7: Vout over changes in temperature.

3.3.2 Square Wave Generator

With the schmitt inverters built we moved on to square wave generation. We used a fairly standard topology for this circuit, with an RC tank connected in feedback around 3 schmitt inverters. We also added a standard AND gate to the input of the first inverter so we could turn the oscillator on and off with an external voltage signal. We also buffer the output with 2 more schmitt inverters so that we can produce sharp square waves without unwanted oscillations. Finally, we used 2 on-chip resistors for the RC tank, but left the capacitor as an external component. This capacitor can be used to set the frequency for the oscillator, anywhere from 10kHz to 1MHz with less than 3% frequency error for the values tested. For any frequency setting in the range we also saw 50% duty cycles across the board, with a maximum error of about 2% as well (and that maximum error was only at the high end of the frequency range). The equation used to set the frequency is:

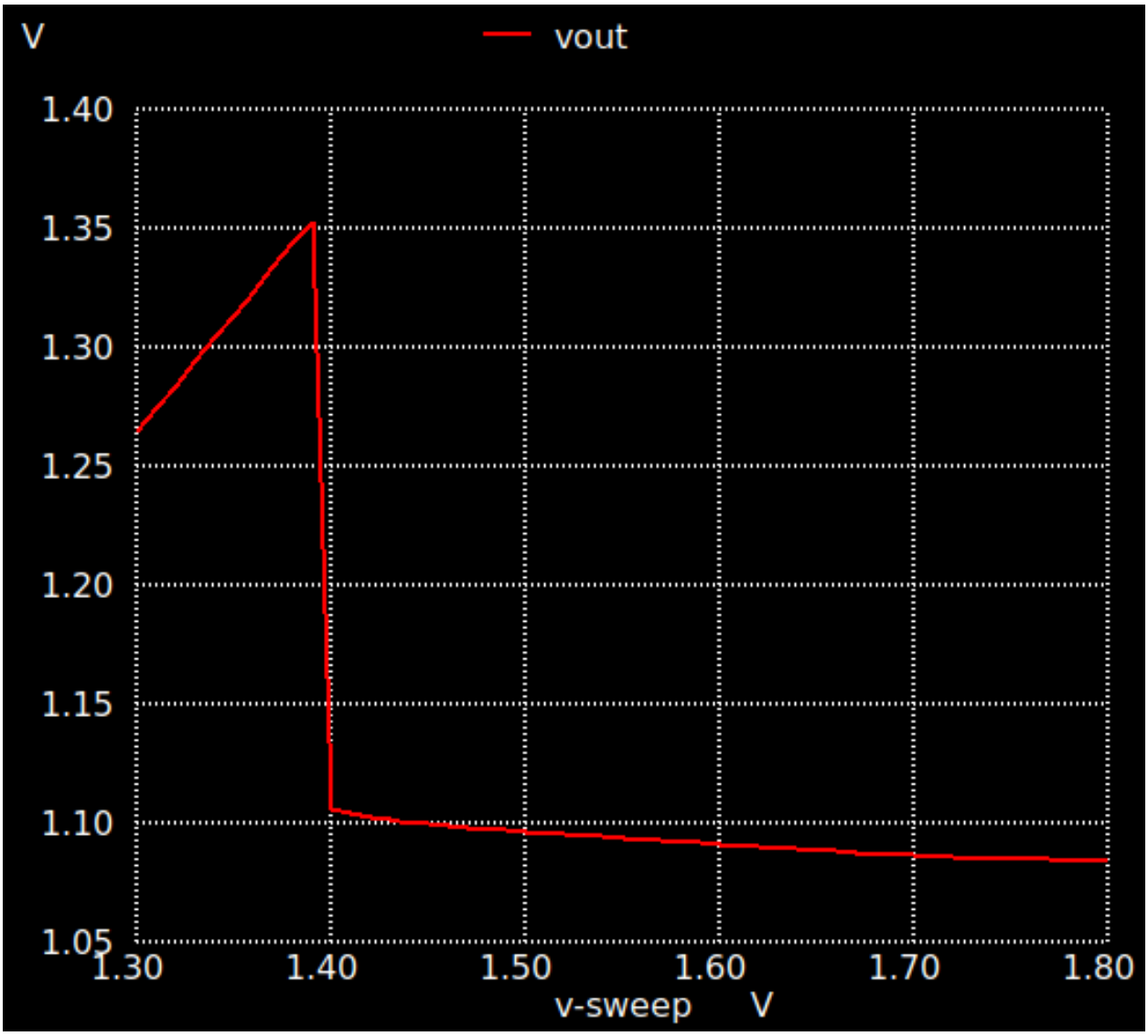

Figure 8: Vout over changes in power supply voltage.

$$C = \frac{1}{(2.3)(10k\Omega)(f_{Hz})}$$

The oscillator schematic is below in figure 11 and simulation results for frequency set to 100kHz is in 12.

3.3.3 Triangle Generator

Now that we could generate a clock signal at whatever frequency we needed, we turned our attention to converting it into forms that could be used in our boost converter. For VMC we needed some type of saw-tooth or triangle wave, which we accomplished by putting our square wave signal through an integrator. This looked very similar to the integrator used in the PI controller, but with a higher total gain and a different capacitor value. Since we were making a triangle wave, the reference voltage coming from the bandgap is also different; we found that centering the signal at a reference voltage of .8V gave the best looking result. Additionally, when we were testing this we saw a lot of drift when just using the first op-amp to generate

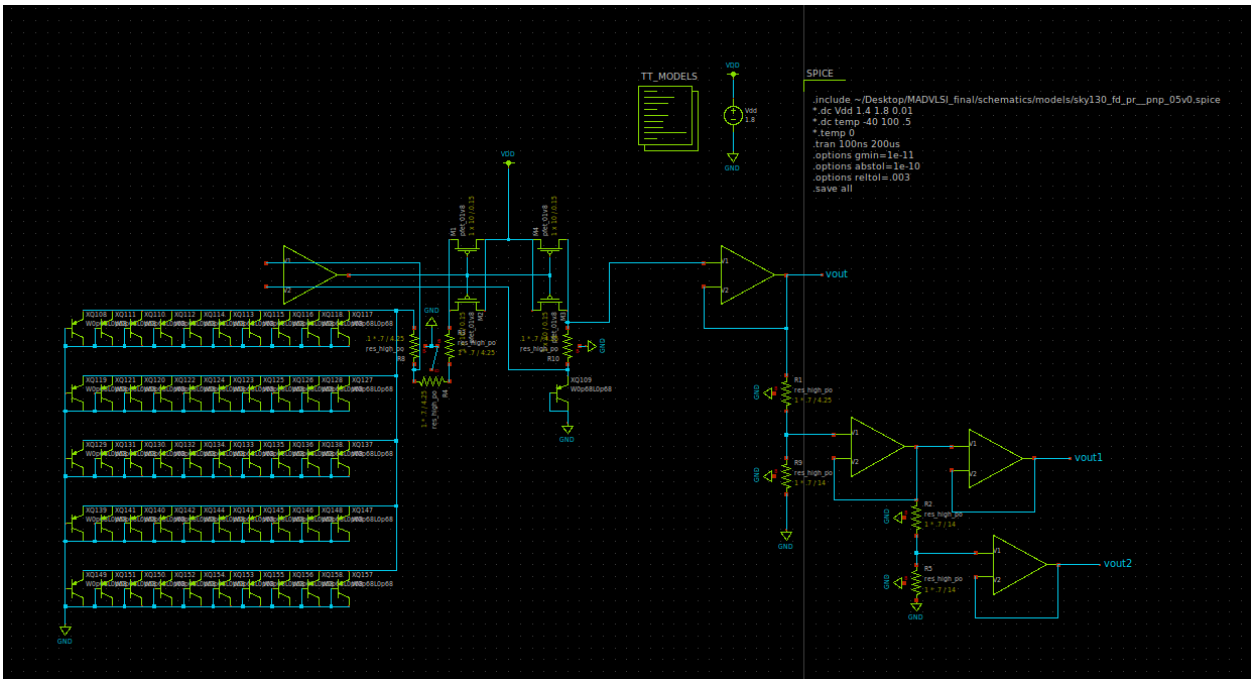

Figure 9: Simple bandgap voltage reference circuit.

a triangle, so the second op-amp was put in as an inverter to add more gain, but also with a 470pF series capacitor to act as a high pass filter and counteract the low frequency drift. It did this pretty well, and we can see both the full schematic used for the triangle generator and simulation results in figures 15 and 14.

3.3.4 Pulse Generator

We weren't able to use the square or triangle wave for CMC, as it requires very narrow pulses to be sent to the SR latch. We can convert our clock signal to thin pulses by using an XOR gate and a time delay circuit. The XOR gate was made out of 4 NAND gates and the time delay circuit was made out of a string of 48 schmitt inverters to act as a buffer (the choice of using schmitt inverters was probably not the most efficient in terms of layout size, that is probably something we would revisit if we had more time); both circuits can be seen in figures **??** and **??**, as well as the full pulse generation circuit in figure 17.

For any frequency clock signal that we tested, there was a constant delay (which is the same as the pulse width) of about 10ns. This was constant for all of the possible frequencies and produced duty cycles of around .03 to .0003, depending on the input clock frequency; duty cycles in this range worked in CMC simulation in both PLECs and xschem. Additionally, because of the arrangement of the circuit (where there are two [1, 0] input time spans to the XOR gate for each clock cycle) the resulting pulses are at twice the frequency of the clock signal input. Simulation results for the pulse generator with our square wave oscillator set to 100kHz can be seen in figure 18.

3.4 SR Latch

The final main sub-circuit we needed to make was the SR latch for CMC. We used the typical cross-coupled NOR gate configuration and it behaves as expected in simulation. The schematic used is below in figure 20.

3.5 Enable and Mode Select

The last goal we had with the circuit was to add usability features. We already give the ability to set the switching frequency with an external capacitor; we then tried to design a way for an external voltage signal

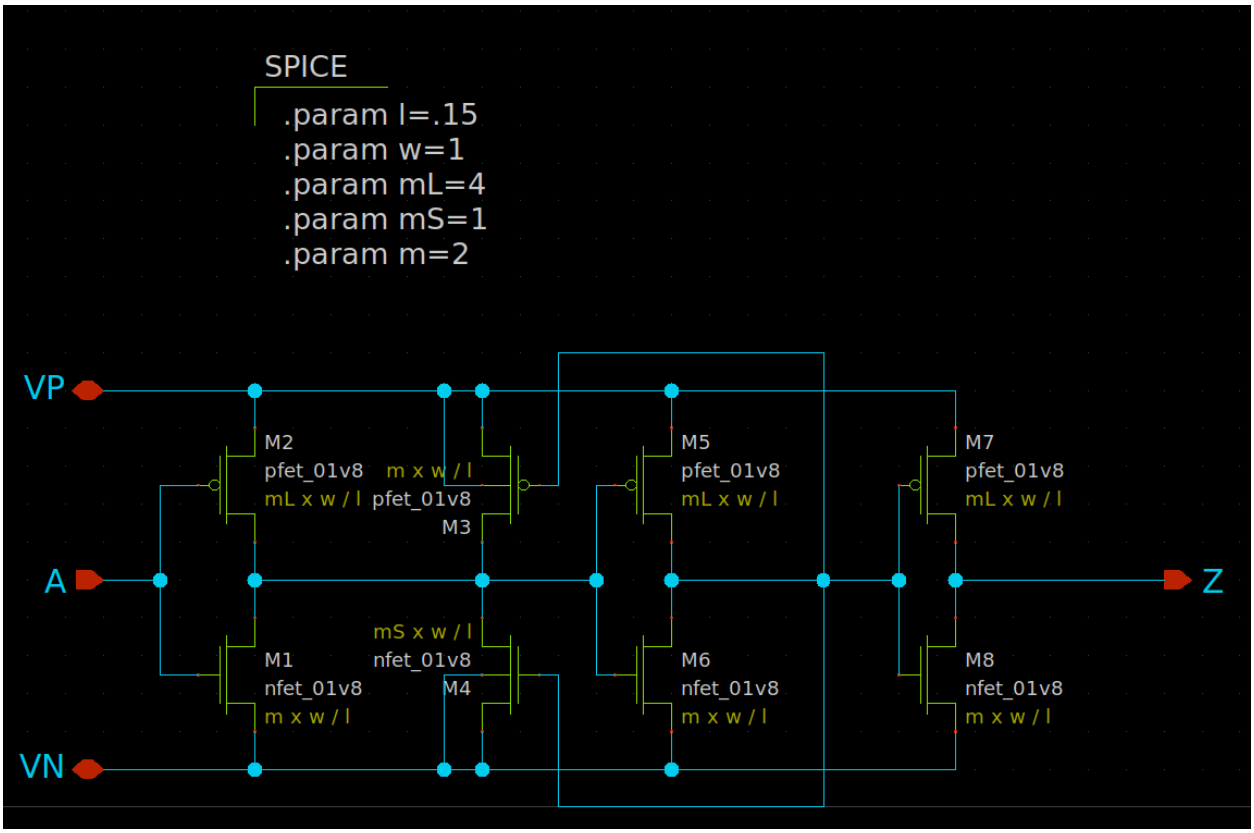

Figure 10: Transistor level schematic of the schmitt inverter we used throughout our controller.

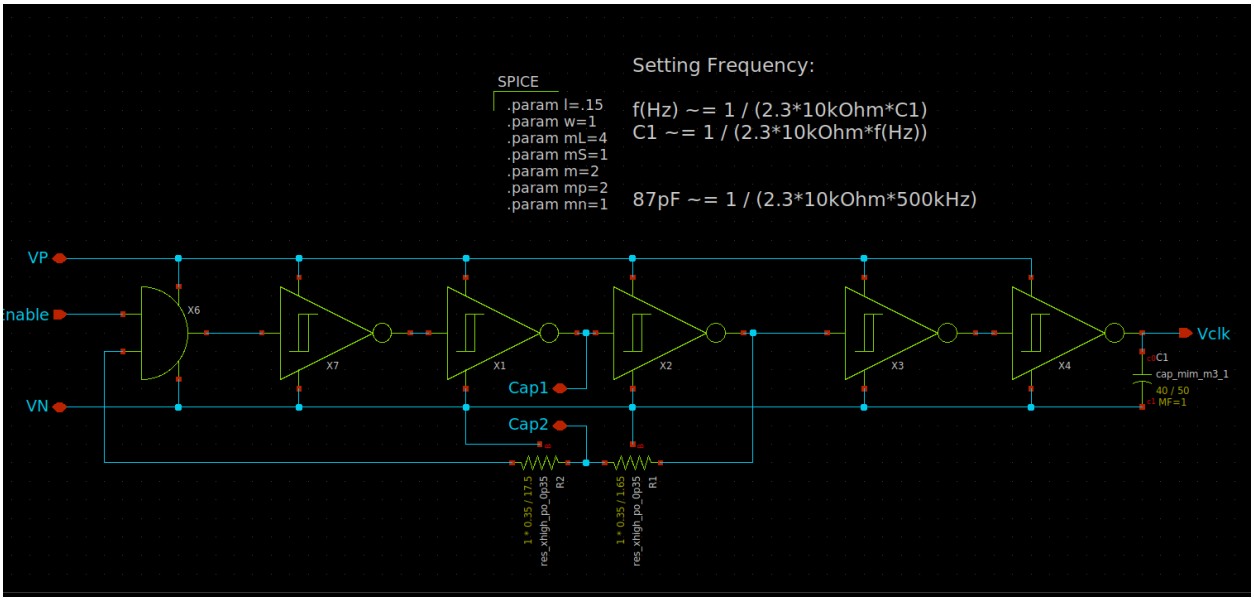

Figure 11: Schmitt inverter relaxation oscillator, 50% duty cycle.

to turn the controller on and for the control scheme to be selected by an external voltage signal. Turning on

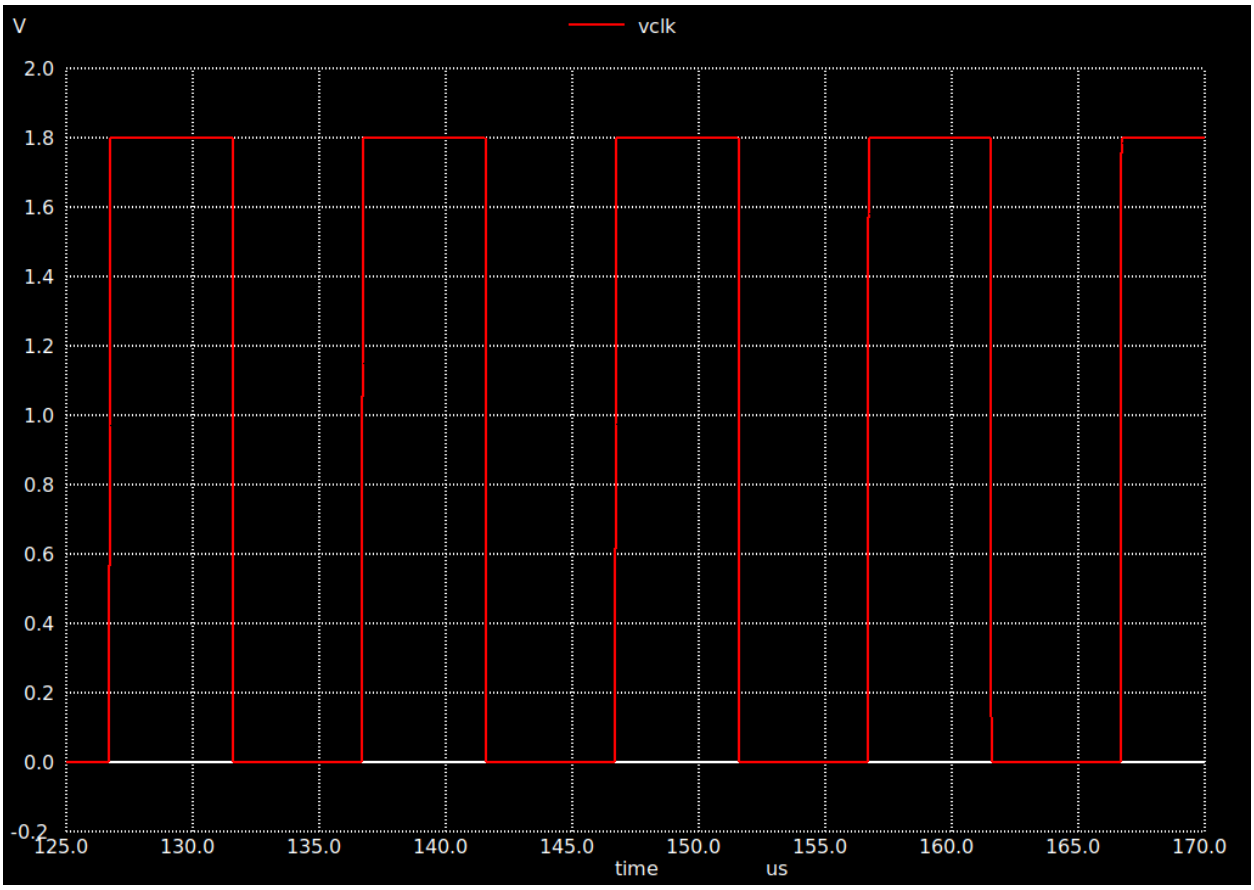

Figure 12: Schmitt inverter relaxation oscillator simulation results, 100kHz (430pF).

the clock is the easy part, as the schmitt oscillator in figure 11 contains an AND gate that allows us to turn the clock on very quickly once the enable signal goes high; this simulation result can be seen in figure **??**.

Turning the clock on and off means we can turn the entire controller off and on when we want, but we still need some way to choose which mode the controller is in, which turned out to require a bit more logic. At first we thought it would be as simple as putting an AND gate in between the clock and the triangle generator and pulse generator, with a ”mode select” (MS) signal going into one AND gate directly and inverted before going into the other one (figure 21).

We thought the circuit above was necessary because we wanted to make sure that the only switching happening in the chip would be due to the desired control scheme, and this circuit gives us a way to ensure the only high frequency switching on the chip is due to one of the pulse generator or triangle generator, but not both. We ran into an issue, however, when the triangle generator is turned off: unlike the pulse generator that rests at GND when Vms goes low, when Vms goes high to turn CMC on the triangle generator can jump around in the middle of the rails, due to Vref still being passed to the circuit. This could cause some switching that might propagate to the power MOSFET gate if the PI controller output reaches the value that the triangle generator rests at during CMC. This behavior can be seen in figure 23.

In order to combat this potential issue we put the CMC and VMC power MOSFET gate signals through their own logic. This makes sure that if mode select is high we stay in CMC, the pulse generator is the only source of switching in the circuit, and the SR latch output is the only thing driving the gate of the MOSFET; and vice versa if mode select is low. The circuit that implements this logic is below in figure **??**.

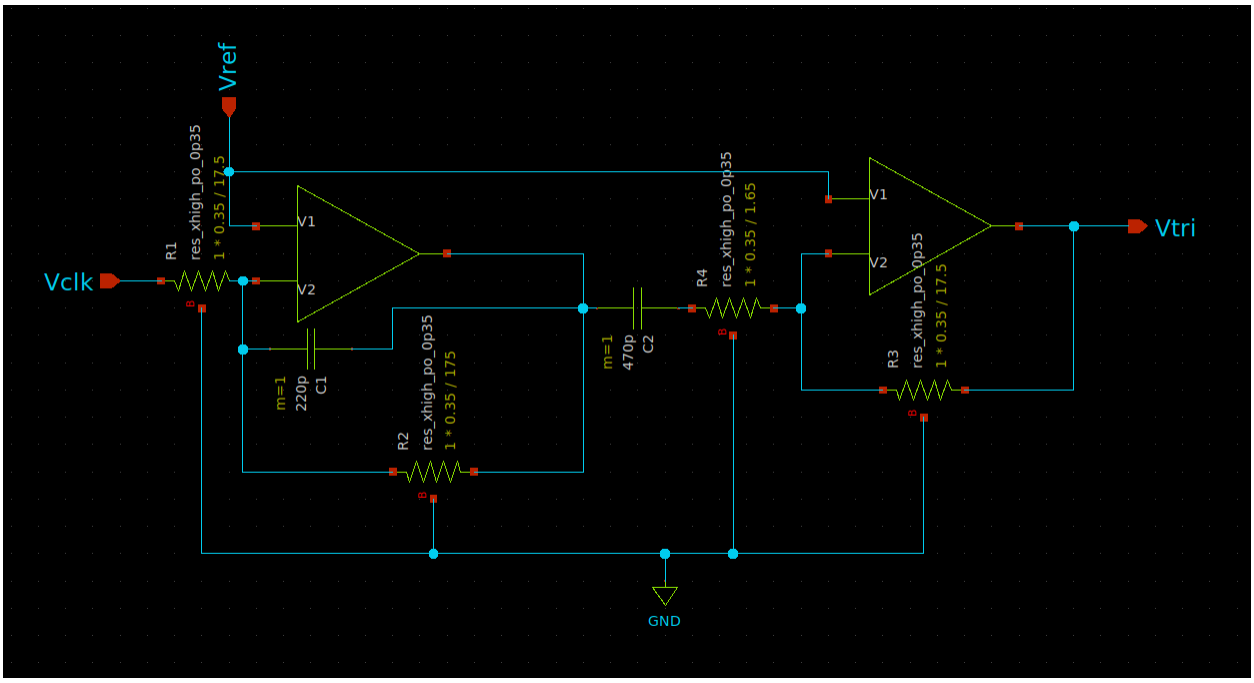

Figure 13: Square wave to triangle wave conversion circuit for VMC. C1 and C2 are both in the 100s of picofarads and as a result have been specified as external components.

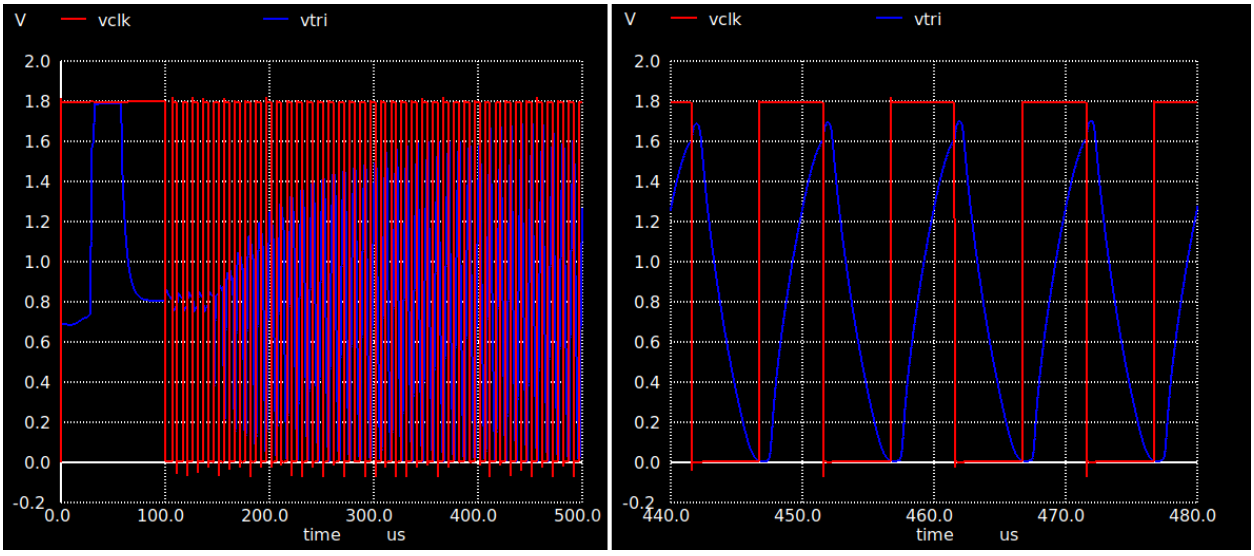

Figure 14: Square wave to triangle wave conversion simulation results with a 100kHz square wave from the schmitt inverter oscillator. The one drawback to this triangle generator that we couldn't fully figure out was that it seems to take some time to settle into it's nice triangle wave oscillations. This probably has something to do with the capacitor/integration, but we couldn't figure out if there was a way to speed the start-up process up.

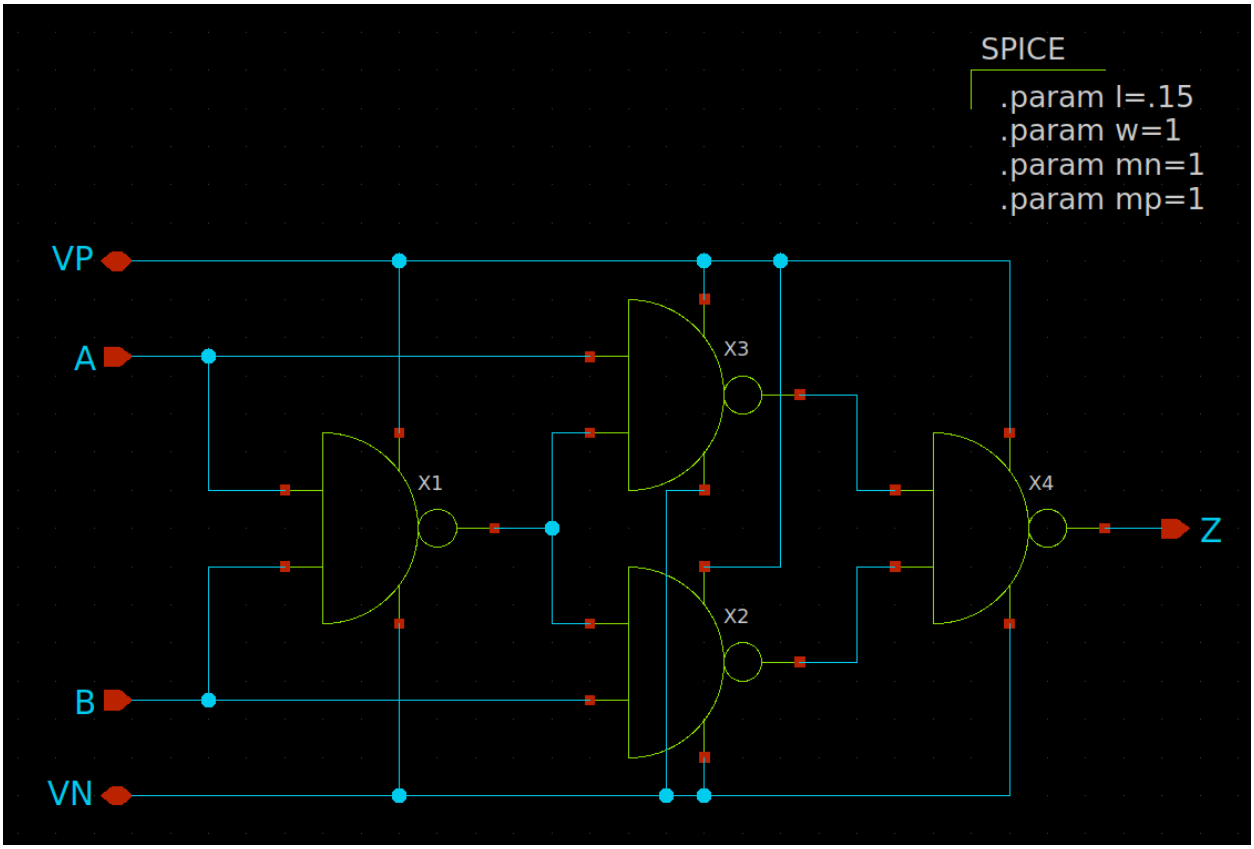

Figure 15: XOR logic gate constructed from 4 NAND gates.

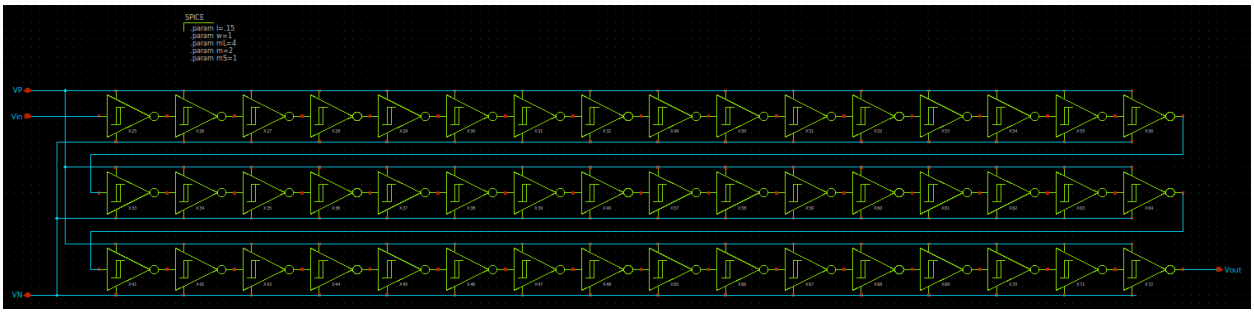

Figure 16: Schmitt buffer time delay circuit.

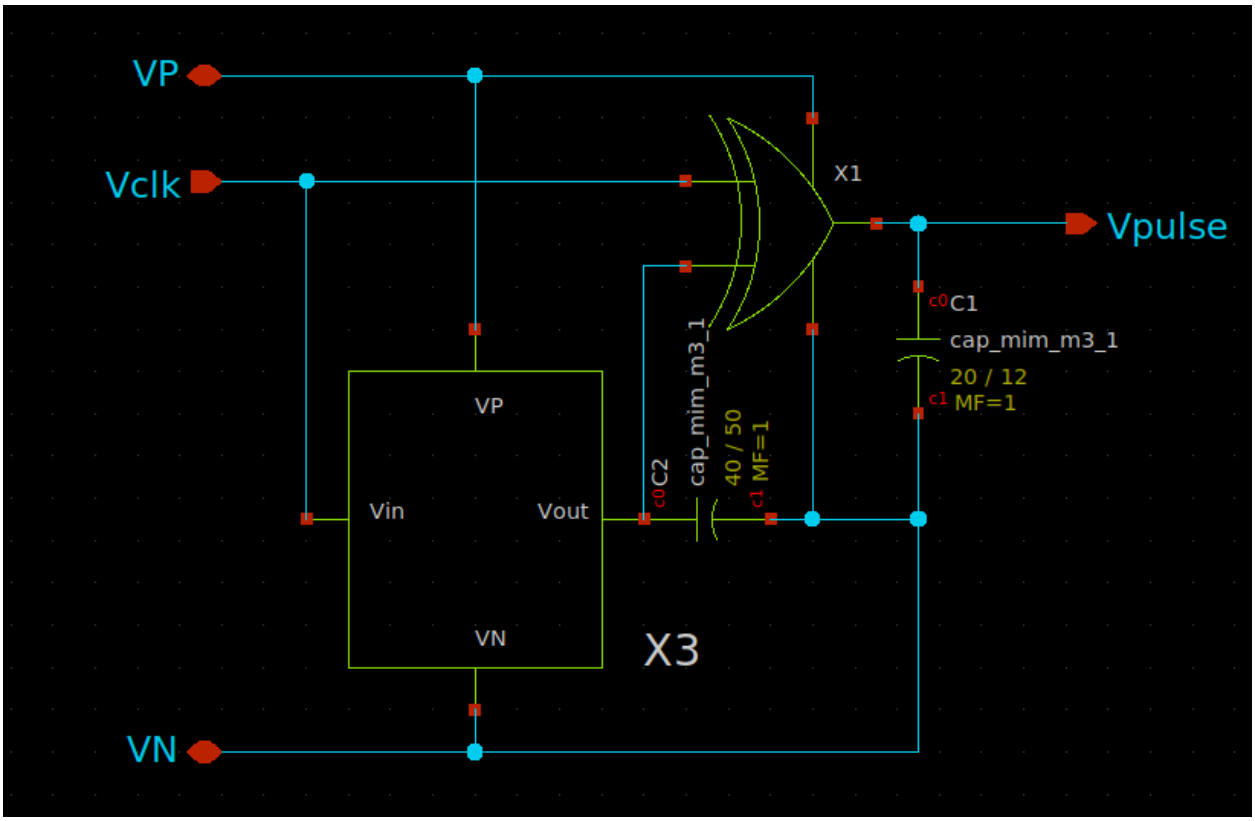

Figure 17: Square wave to thin pulse generator circuit.

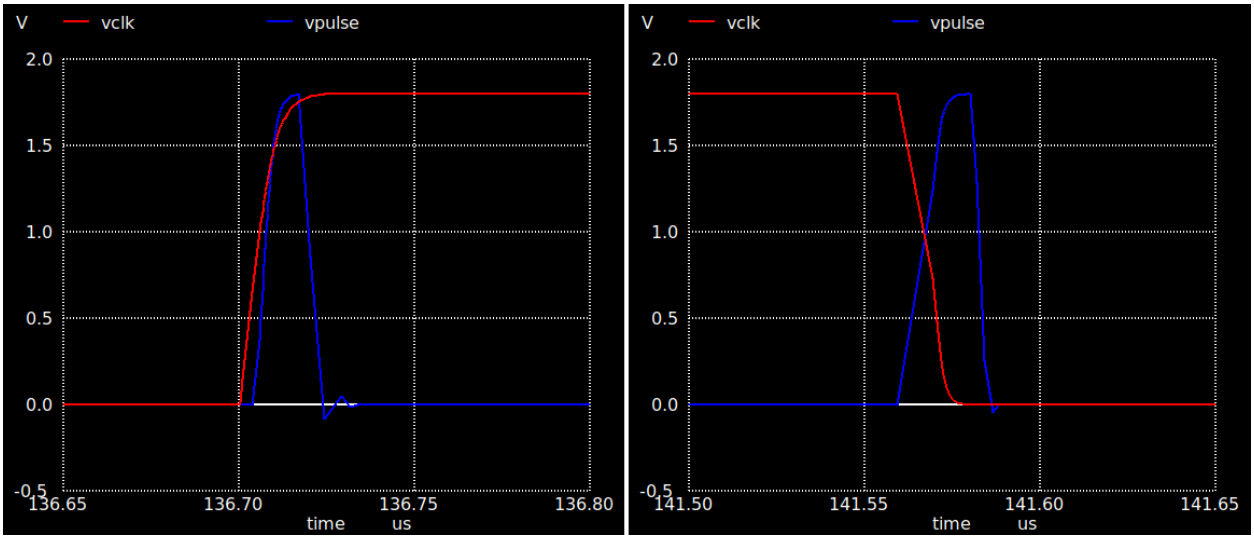

Figure 18: Rising and falling edges of the input clock signal and resulting pulses.

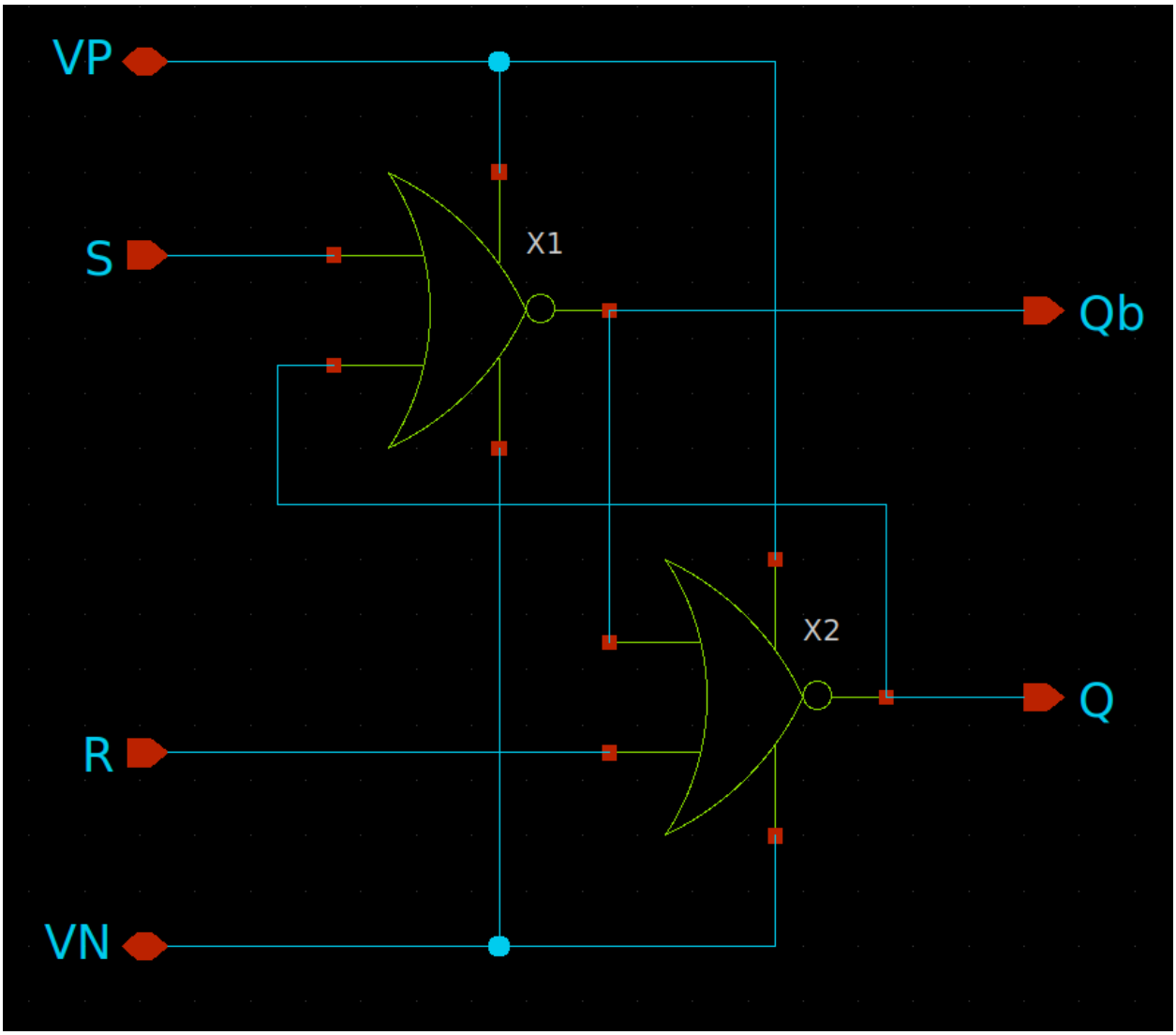

Figure 19: Typical SR latch with an enable pin.

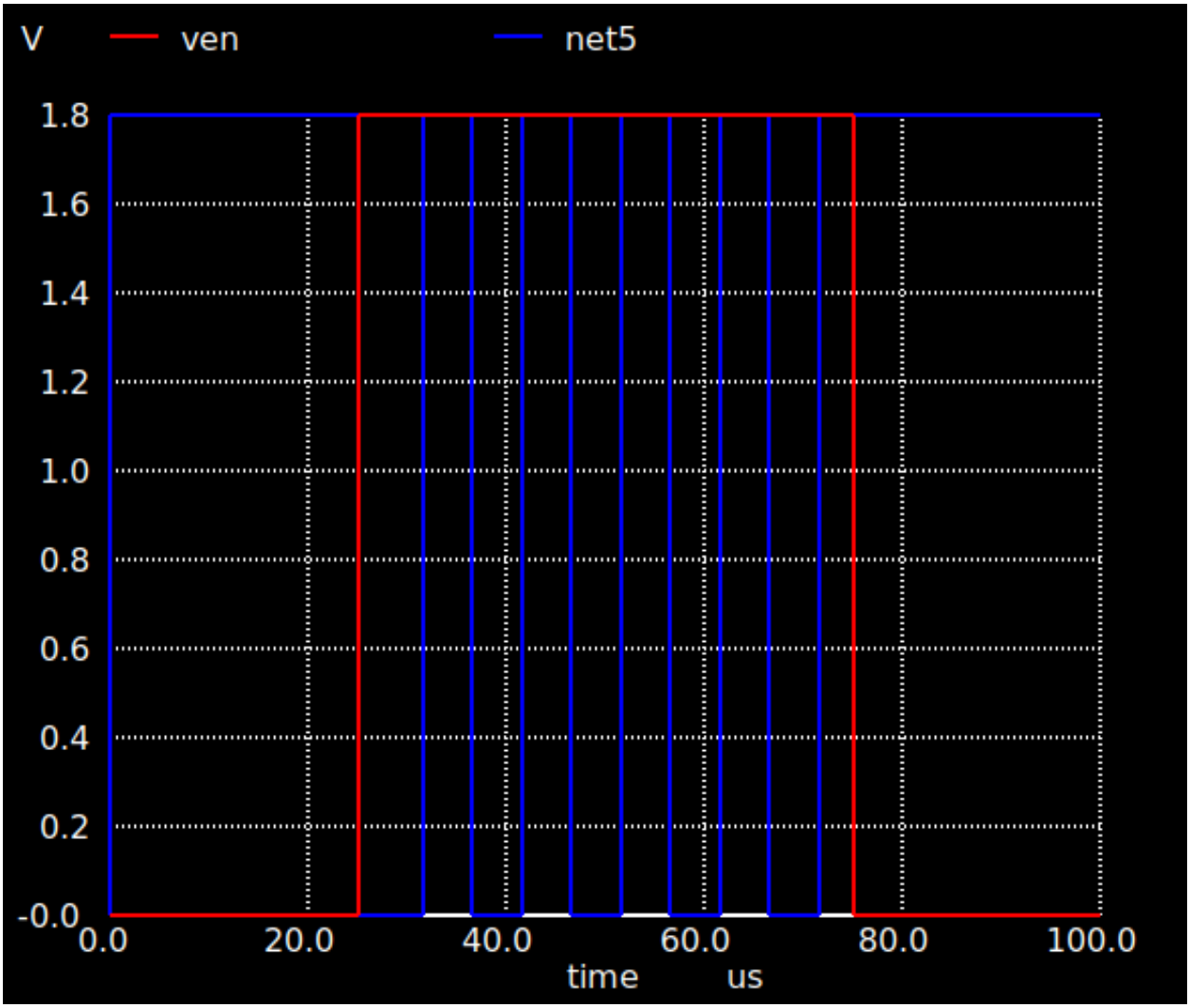

Figure 20: Simulation results showing the oscillators response to an ″enable″ or ″start-up″ signal.

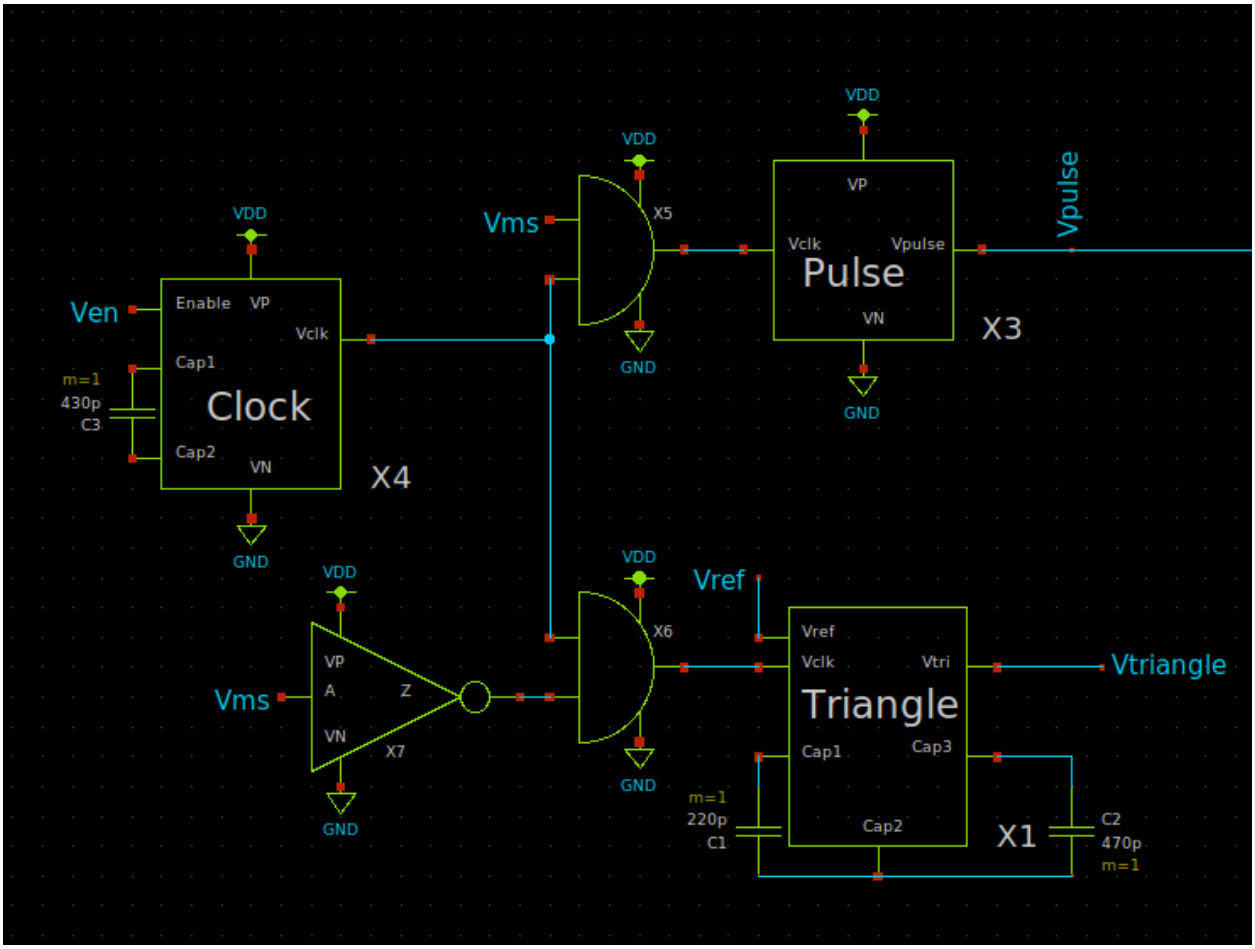

Figure 21: Mode select logic on the clock side. Setting Vms high turns on the pulse generator, while setting it low turns on VMC. As we will see shortly this logic by itself is not sufficient.

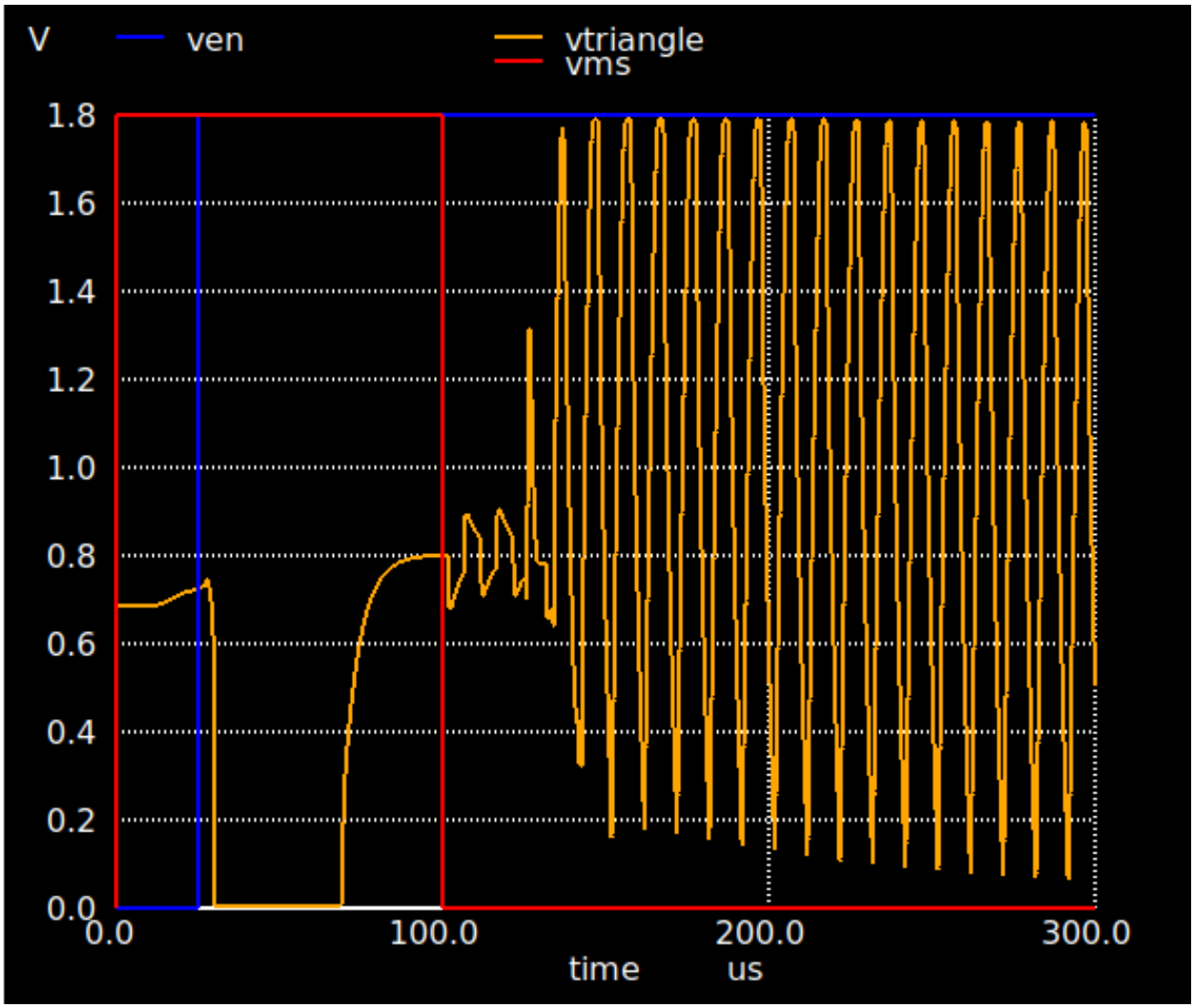

Figure 22: Simulation results showing what happens to the triangle generator output when Vms is toggled. When Vms is high CMC is on and the triangle generator just does random things; when Vms goes low the pulse generator turns off and the triangle wave starts to turn itself on.

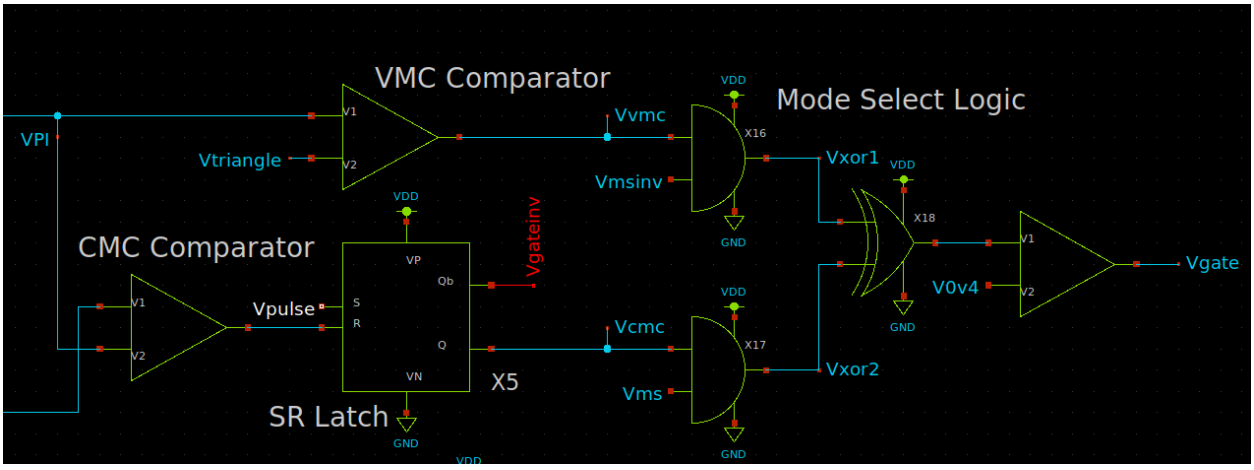

Figure 23: Mode select logic outputting a gate signal to the power MOSFET; Vms is the same mode select signal from figure 21 and Vmsinv is the inverted Vms signal. Vms set to high will allow CMC control to propagate through the circuit while Vms set low will allow VMC control.

4 Simulation Quirks and Results

4.1 Limitations

One thing we found out pretty early on is that our switching circuits and bandgap voltage reference are very complex, involve many components, and, in the case of the switching circuits, require very small time-steps and prohibitively long times to simulate accurately (e.g. when testing the mode select logic with the clock, pulse generator, and triangle generator the simulation won't return accurate results unless the maximum time step is set to 10ns or less). This has made it pretty much impossible to simulate the entire boost controller, or even to include the bandgap, clock, pulse generator, or triangle generator in any of the boost circuit simulations because they slow everything down a lot (when the simulations already take a long time to run) and threaten to overload our VM's memory.

Unfortunately this means that all of our full boost circuit simulations need to be with ideal voltage sources setting the reference voltages and creating the triangle/pulse signals. While this is disappointing, we know that any of these signals could be buffered by op-amps before being sent wherever they need to be in the circuit, so we think it is a valid choice for us to use ideal voltage sources to closely mimic the results we saw by testing our sub-circuits individually.

4.2 3rd Party Spice Models

To accurately simulate a boost converter, we needed access to PCB mounted transistor and diode models, since the skywater models that are used for VLSI design are not the same type of device as a power MOSFET; the same is true of the diode models we saw we had access to. The power MOSFET we picked was the DMG8601UFG-7, which we chose because it was capable of handling the drain voltages and currents we were expecting, and its maximum threshold voltage is only 1V, which is within the range that we can drive with our 1.8V capable models. The diode we picked was the SBR10U40CT, which we chose because of its relatively low forward voltage and current rating.

4.3 Simulation Problems

Unfortunately our simulations started to act up at the last minute. The op-amps that we have been using successfully the entire project have stopped behaving in the VMC and CMC simulations; we think that this is because there are so many more transistor models for the simulator to deal with once we made them layout driven schematics and tried to simulate multiple sub-circuits together at these high switching frequencies. For example, figure 24 shows plots of our VMC comparator positive and negative inputs on the right and it's output on the left during a VMC simulation, which is obviously not correct.

Figure 25 below was from the exact same op amp schematic, with the exact same triangle wave going into its negative input, with the same .4V Vref on the positive input that we saw in the VMC simulation and we can see that its output is completely different.

4.4 VMC Simulation Results

Fortunately we have some images saved from simulations before and during this weekend before we started to replace a lot of our subcircuits with the layout driven schematic versions. Our VMC output voltage plot is below in figure 26

4.5 CMC Simulation Results

We had also saved CMC output voltage simulation results, seen below in **??**. We were surprised that this method seemed to be slightly noisier and less controlled than the VMC results, but still within reason.

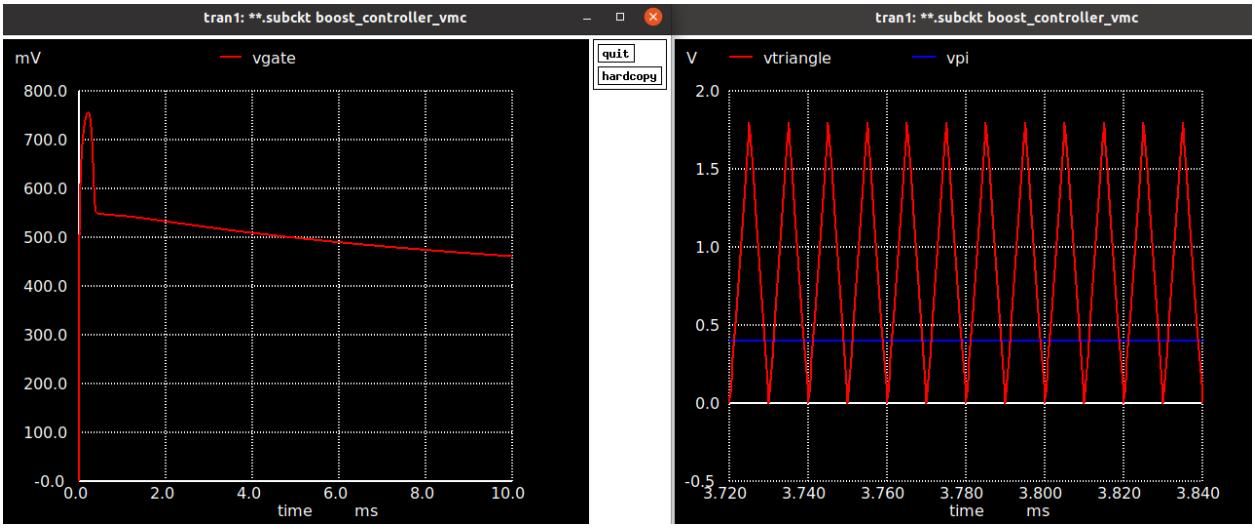

Figure 24: From our most recent attempt to fix our VMC schematic, the op-amp comparator's positive input is Vpi and its negative input is Vtriangle. This is not very fast (100kHz) and it is obvious that the op-amp isn't behaving correctly. Compare this with figure 25 below.

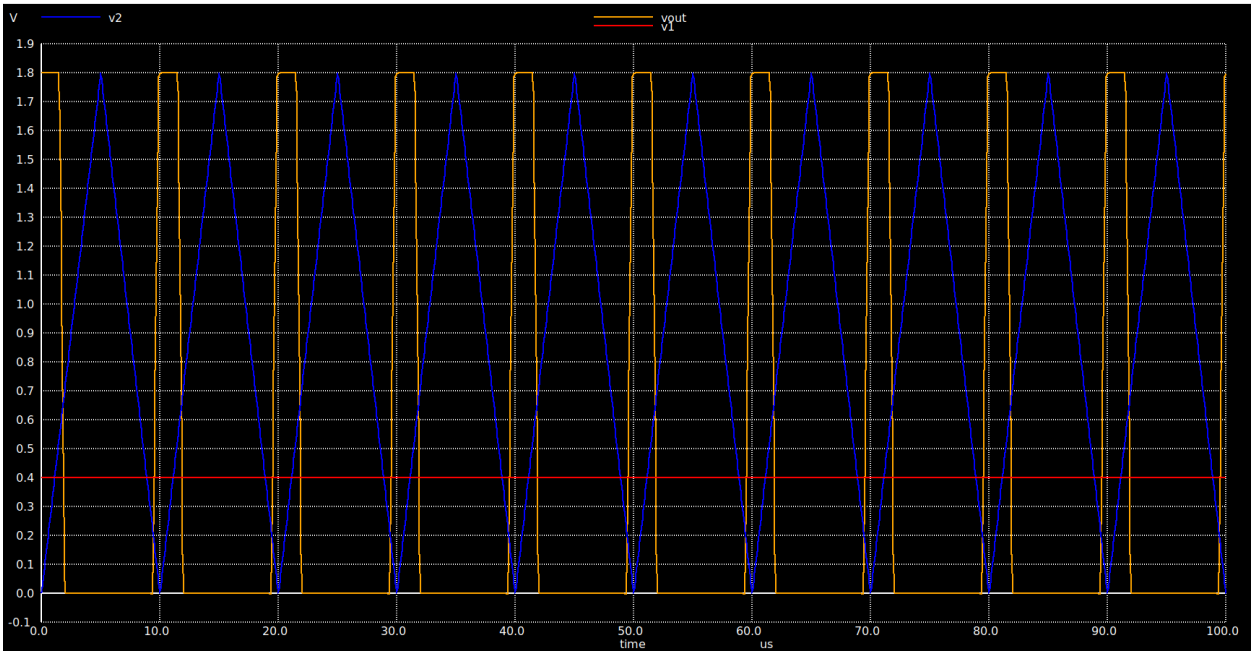

Figure 25: Op-amp simulation by itself. When we ran into issues with the VMC simulation and saw the op-amp waveforms we decided to test it in isolation and can see that with a 100kHz triangle wave and a positive input at .4V the device should behave correctly.

4.6 Full Controller Circuit

Even though the circuit can't really simulate well anymore with the layout driven schematic sub-circuits included, we went ahead and put the entire boost controller circuit together, including how it would ideally interface with a the external circuit. This can be seen below in figure 28.

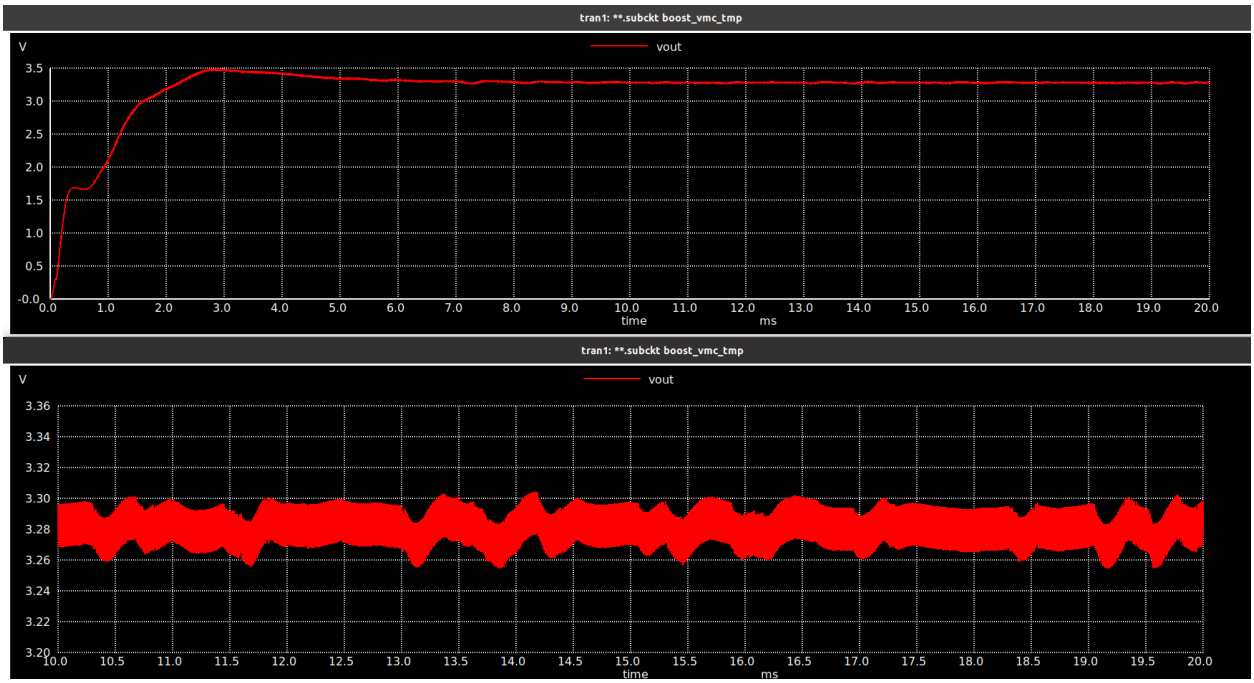

Figure 26: An image we had saved from this weekend showing our output voltage with a 1A load using the **voltage mode control** scheme. Our output voltage ripple was able to stay within about 1%-2% and we were centered on about 3.28V, less than 1% off of our goal of 3.3V. The results of this simulation and our CMC simulation in figure **??** were both achieved using the exact same circuits and component values that we showed in the sections above, just before the individual sub-circuits had been converted to layout driven schematics.

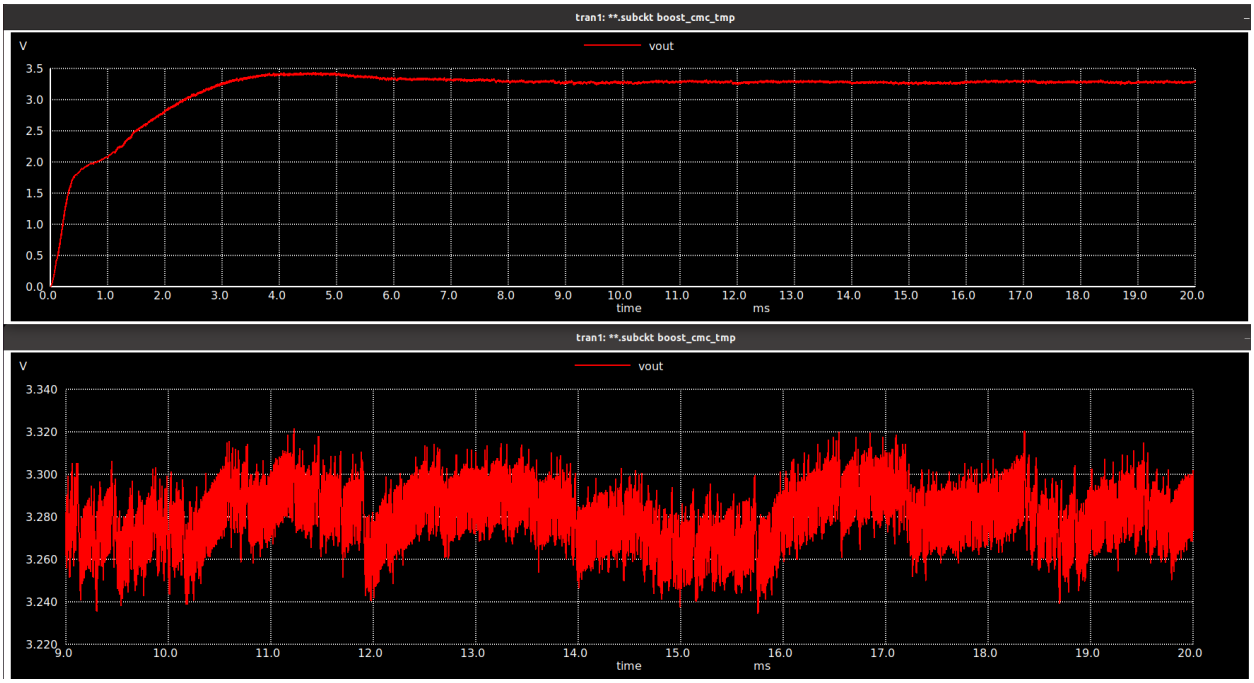

Figure 27: An image we had saved from this weekend showing our output voltage with a 1A load using the **current mode control** scheme. This time our output voltage looked to have ripple of about 2.5% and we were still centered around 3.28V, less than 1% off of our goal of 3.3V. The results of this simulation and our VMC simulation in figure **??** were both achieved using the exact same circuits and component values that we showed in the sections above, just before the individual sub-circuits had been converted to layout driven schematics.

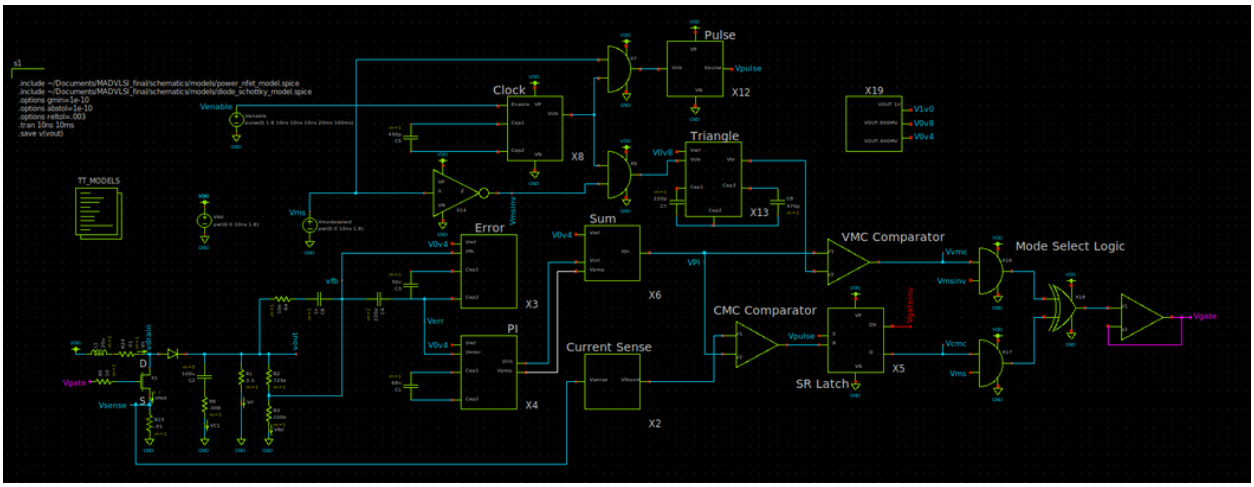

Figure 28: Our full boost controller circuit, including both VMC and CMC control schemes, the ability to switch between them by toggling Vms, adjustable switching frequency, and enable pin to start and stop the clock.

5 Layouts

The following images depict the layouts of all of our sub-circuits. Unfortunately, we were not able to combine all of these sub-circuits together to form the entire system in one magic file. We do believe, however, that we covered the most technical aspects of the layout. Smaller layouts for logic gates were not included in the report but can be found in our repository.

5.1 Clock Generator

5.2 Current Boost

5.3 Error Amplifier

5.4 Operational Amplifier

We have two different form factors for the opamp to help with layout.

5.5 PI controller

5.6 Pulse Generator

5.7 Schmitt Trigger

5.8 Summing Amplifier

5.9 Bandgap Voltage Reference

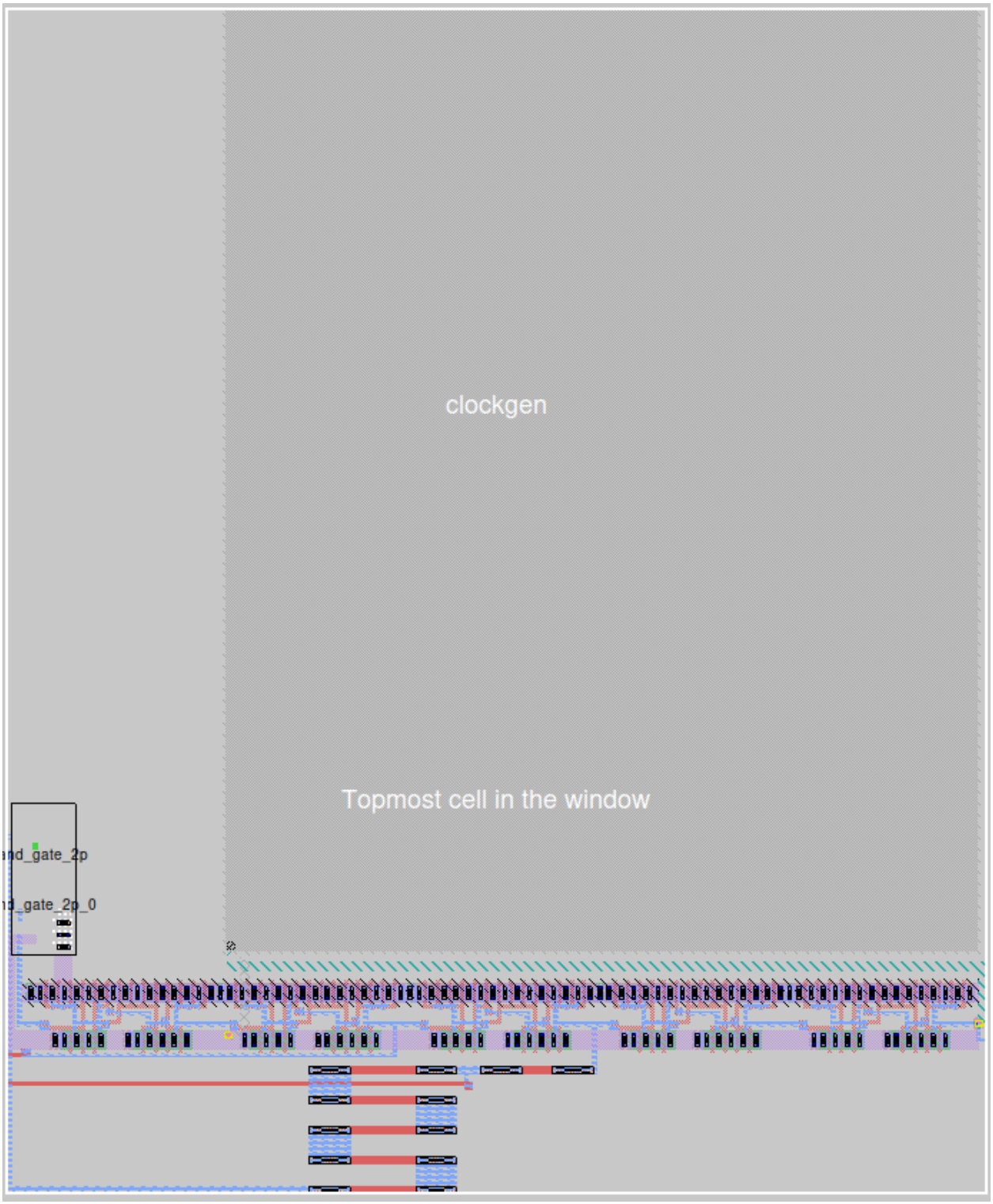

Figure 29: Clock Generating Circuit.

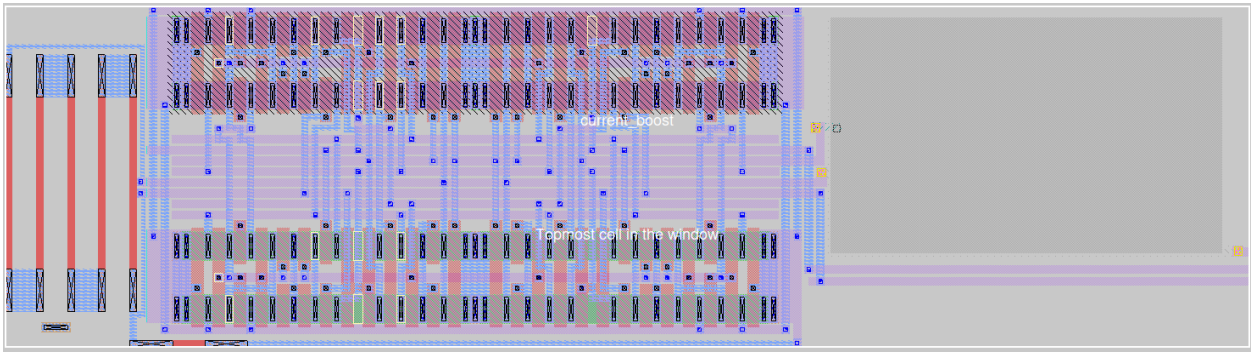

Figure 30: Current Boost circuit.

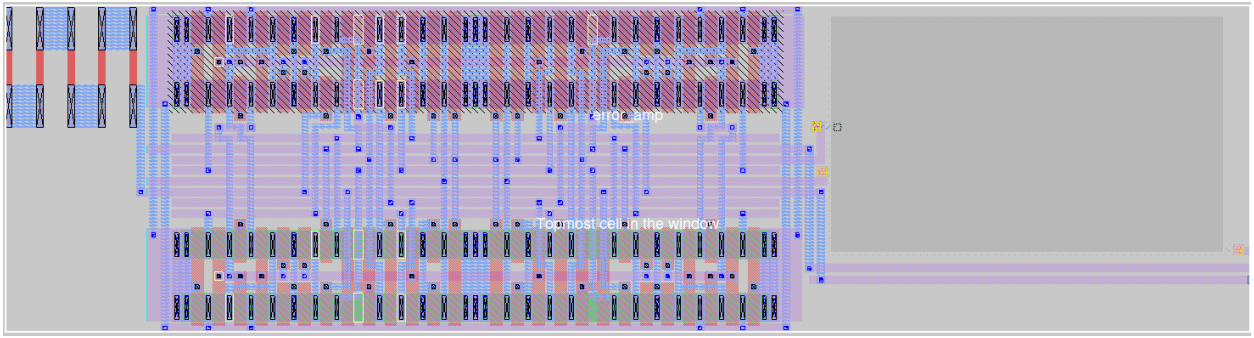

Figure 31: Error Amplifying circuit.

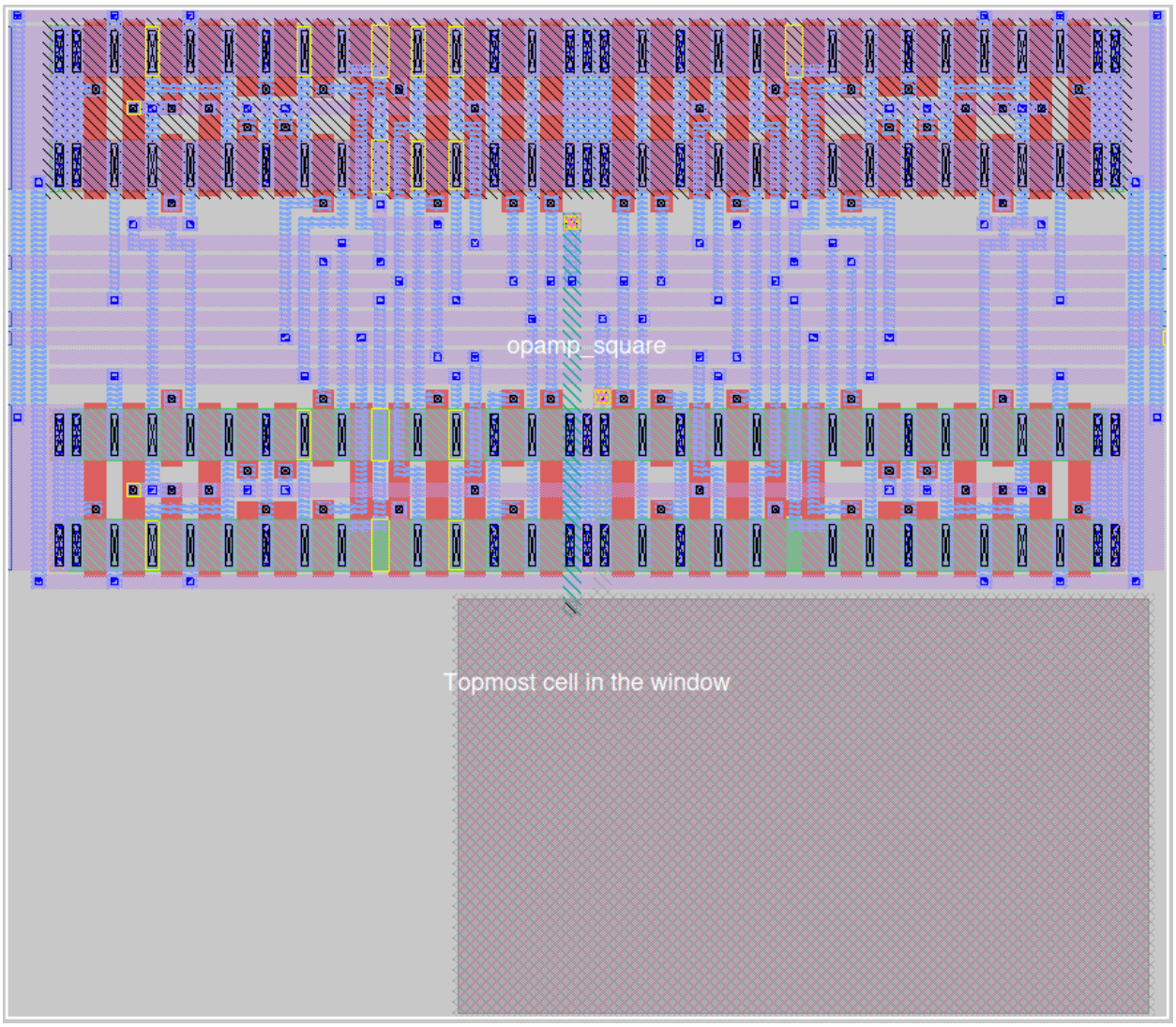

Figure 32: Square Opamp circuit.

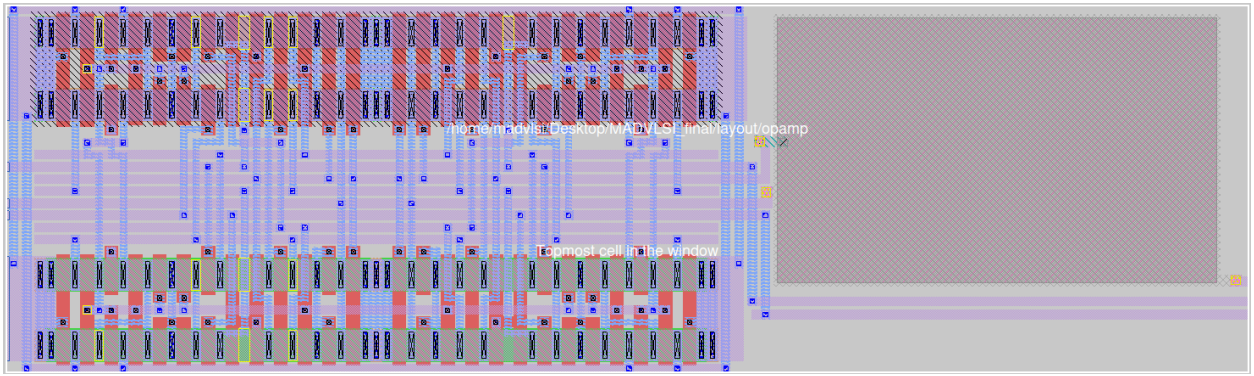

Figure 33: Rectangular Opamp circuit.

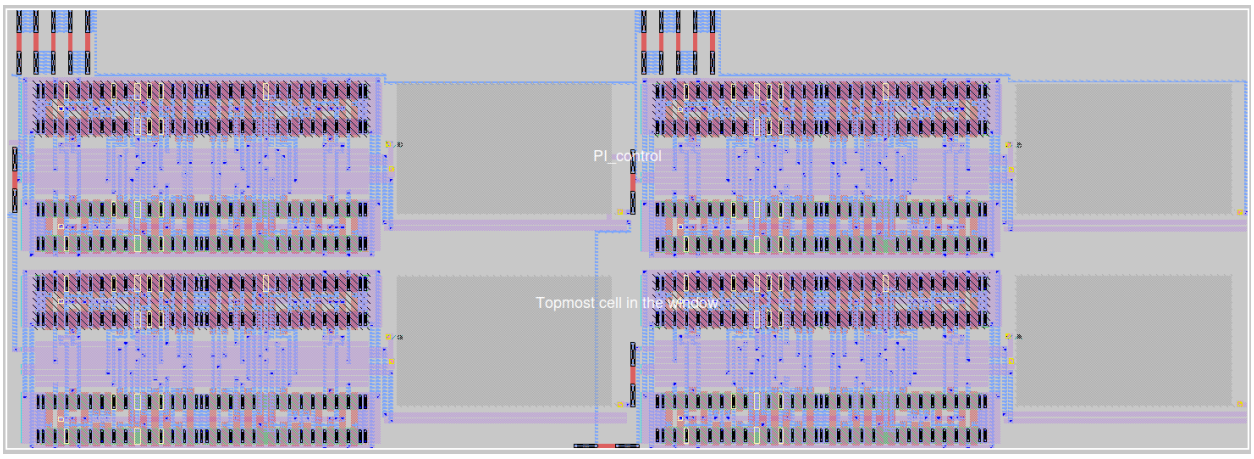

Figure 34: PI controller.

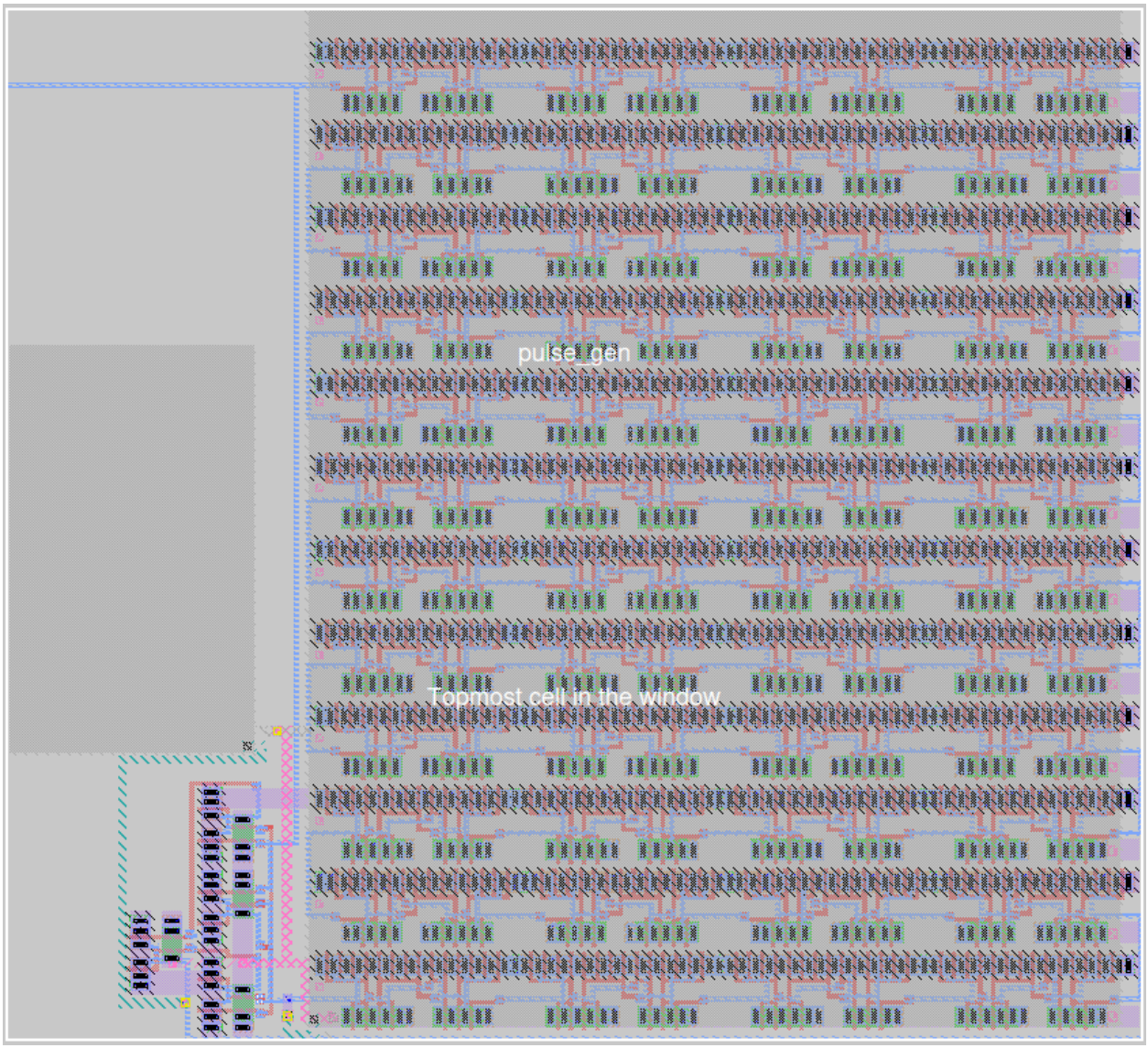

Figure 35: Pulse Generator.

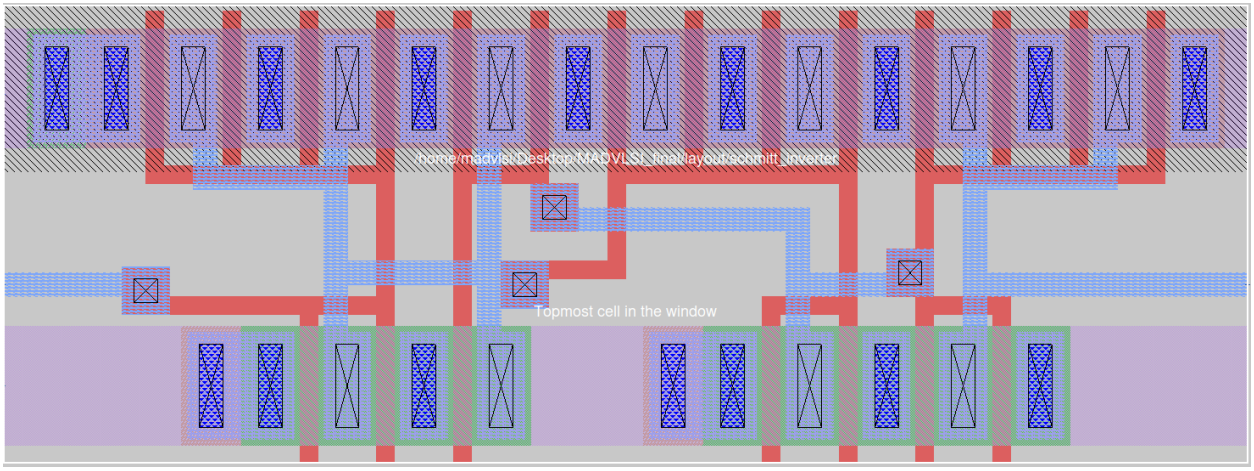

Figure 36: Schmitt Trigger

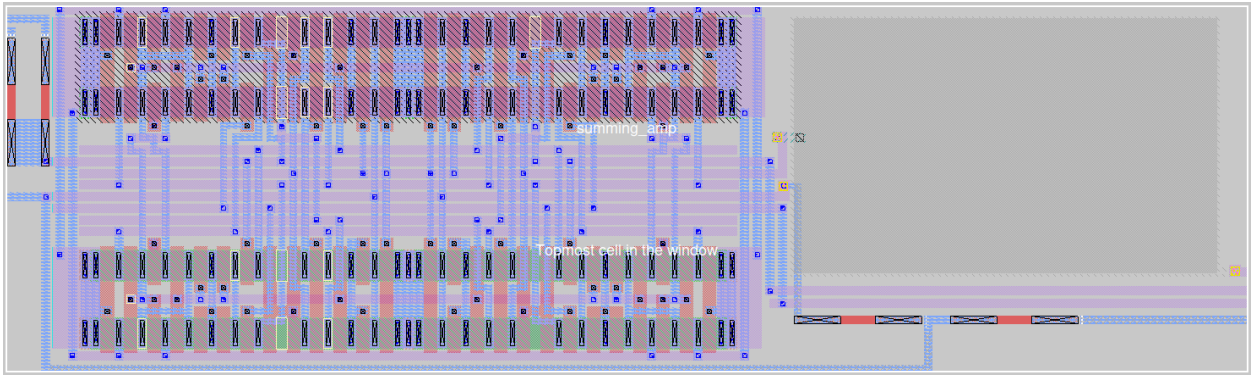

Figure 37: Summing Amplifier Circuit

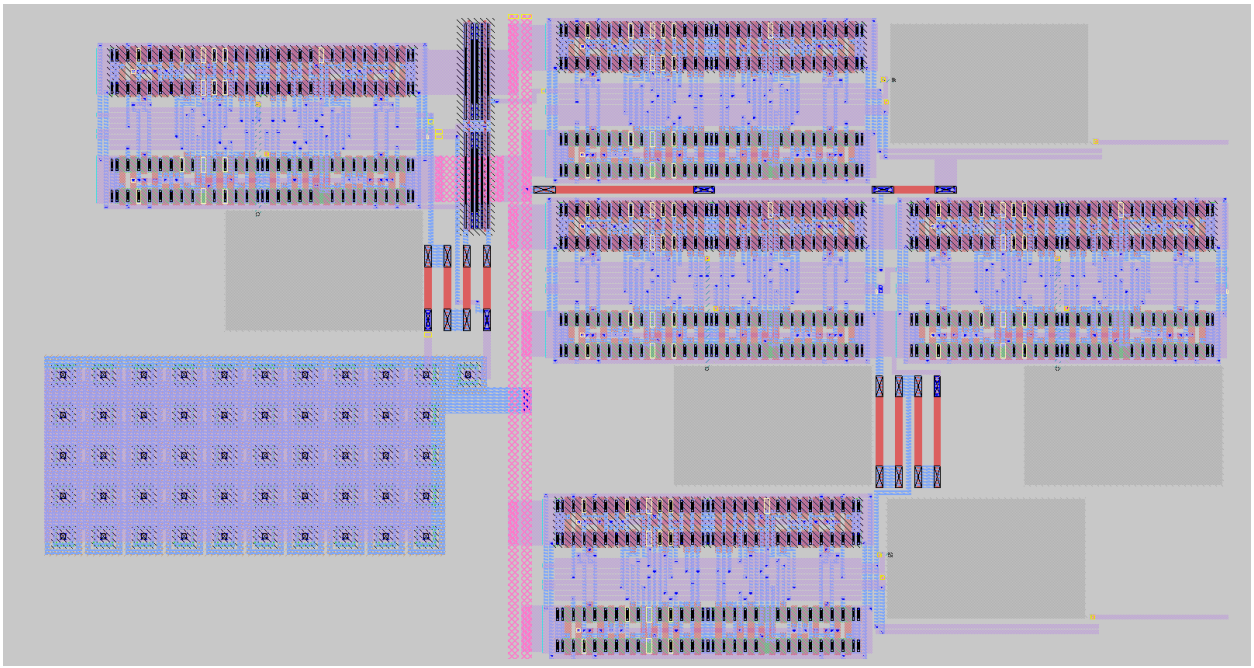

Figure 38: Magical layout of bandgap circuit.

6 Layout Versus Schematic (LVS)

6.1 Error Amplifier

```
Flattening unmatched subcell k_res in circuit error_amp_magic.spice (1)(1 instance)
Equate elements:  no current cell.
Equate elements:  no current cell.
Equate elements:  no current cell.
Equate elements:  no current cell.
Class opamp:  Merged 33 devices.
Class opamp:  Merged 33 devices.

Subcircuit summary:
Circuit 1: opamp                          |Circuit 2: opamp
------------------------------------------|------------------------------------------
sky130_fd_pr__pfet_01v8 (35)              |sky130_fd_pr__pfet_01v8 (35)
sky130_fd_pr__nfet_01v8 (36)              |sky130_fd_pr__nfet_01v8 (36)
sky130_fd_pr__cap_mim_m3_1 (1)            |sky130_fd_pr__cap_mim_m3_1 (1)
Number of devices: 72                     |Number of devices: 72
Number of nets: 30                        |Number of nets: 30
-------------------------------------------------------------------------------------
Resolving automorphisms by property value.
Resolving automorphisms by pin name.
Netlists match uniquely.
Circuits match correctly.

Subcircuit pins:
Circuit 1: opamp                          |Circuit 2: opamp
------------------------------------------|------------------------------------------
VDD                                       |vdd
GND                                       |gnd
VOUT                                      |vout
V1                                        |v1
V2                                        |v2
-------------------------------------------------------------------------------------
Cell pin lists are equivalent.
Device classes opamp and opamp are equivalent.

Subcircuit summary:
Circuit 1: error_amp.spice                |Circuit 2: error_amp_magic.spice
------------------------------------------|------------------------------------------
sky130_fd_pr__res_xhigh_po (1)            |sky130_fd_pr__res_xhigh_po (1)
opamp (1)                                 |opamp (1)
Number of devices: 2                      |Number of devices: 2
Number of nets: 6                         |Number of nets: 6
-------------------------------------------------------------------------------------
Circuits match uniquely.
Netlists match uniquely.
Cells have no pins;  pin matching not needed.
Device classes error_amp.spice and error_amp_magic.spice are equivalent.
Circuits match uniquely.
```

6.2 Op Amp

```
Equate elements:  no current cell.
Equate elements:  no current cell.
Equate elements:  no current cell.
Class schematics/opamp.spice:  Merged 33 devices.
Class layout/opamp.spice:  Merged 33 devices.

Subcircuit summary:
Circuit 1: schematics/opamp.spice          |Circuit 2: layout/opamp.spice
-------------------------------------------|-------------------------------------------
sky130_fd_pr__pfet_01v8 (35)               |sky130_fd_pr__pfet_01v8 (35)
sky130_fd_pr__nfet_01v8 (36)               |sky130_fd_pr__nfet_01v8 (36)
sky130_fd_pr__cap_mim_m3_1 (1)             |sky130_fd_pr__cap_mim_m3_1 (1)
Number of devices: 72                      |Number of devices: 72
Number of nets: 30                         |Number of nets: 30
-----------------------------------------------------------------------------------
Resolving automorphisms by property value.
Resolving automorphisms by pin name.
Netlists match uniquely.
Circuits match correctly.
Cells have no pins;  pin matching not needed.
Device classes schematics/opamp.spice and layout/opamp.spice are equivalent.
Circuits match uniquely.
```

6.3 Summing Amplifier

```
Flattening unmatched subcell k_res in circuit summing_amp_magic.spice (0)(4 instances)
Equate elements:  no current cell.
Equate elements:  no current cell.
Equate elements:  no current cell.
Equate elements:  no current cell.
Class opamp:  Merged 33 devices.
Class opamp:  Merged 33 devices.

Subcircuit summary:
Circuit 1: opamp                           |Circuit 2: opamp
-------------------------------------------|-------------------------------------------
sky130_fd_pr__nfet_01v8 (36)               |sky130_fd_pr__nfet_01v8 (36)
sky130_fd_pr__pfet_01v8 (35)               |sky130_fd_pr__pfet_01v8 (35)
sky130_fd_pr__cap_mim_m3_1 (1)             |sky130_fd_pr__cap_mim_m3_1 (1)
Number of devices: 72                      |Number of devices: 72
Number of nets: 30                         |Number of nets: 30
-----------------------------------------------------------------------------------
Resolving automorphisms by property value.
Resolving automorphisms by pin name.
Netlists match uniquely.
Circuits match correctly.

Subcircuit pins:
Circuit 1: opamp                           |Circuit 2: opamp
-------------------------------------------|-------------------------------------------
vdd                                        |VDD
gnd                                        |GND
v2                                         |V2
v1                                         |V1
```

```
vout                                                    |VOUT
--------------------------------------------------------------------------------
Cell pin lists are equivalent.
Device classes opamp and opamp are equivalent.

Subcircuit summary:
Circuit 1: summing_amp_magic.spice                      |Circuit 2: summing_amp.spice
--------------------------------------------------------|------------------------------------------
sky130_fd_pr__res_xhigh_po (4)                          |sky130_fd_pr__res_xhigh_po (4)
opamp (1)                                               |opamp (1)
Number of devices: 5                                    |Number of devices: 5
Number of nets: 8                                       |Number of nets: 8
--------------------------------------------------------------------------------
Resolving automorphisms by property value.
Resolving automorphisms by pin name.
Netlists match uniquely.
Circuits match correctly.
Cells have no pins;  pin matching not needed.
Device classes summing_amp_magic.spice and summing_amp.spice are equivalent.
Circuits match uniquely.
```

6.4 Time Delay

```
Flattening unmatched subcell schmitt_inverter in circuit time_delay_2.spice (0)(48 instance
Flattening unmatched subcell inverter_schmitt_lvs_m in circuit time_delay_lvs.spice (1)(48
Equate elements:  no current cell.
Equate elements:  no current cell.
Class time_delay_2.spice:  Merged 624 devices.
Class time_delay_lvs.spice:  Merged 624 devices.

Subcircuit summary:
Circuit 1: time_delay_2.spice                           |Circuit 2: time_delay_lvs.spice
--------------------------------------------------------|------------------------------------------
sky130_fd_pr__nfet_01v8 (192)                           |sky130_fd_pr__nfet_01v8 (192)
sky130_fd_pr__pfet_01v8 (192)                           |sky130_fd_pr__pfet_01v8 (192)
Number of devices: 384                                  |Number of devices: 384
Number of nets: 147                                     |Number of nets: 147
--------------------------------------------------------------------------------
Circuits match uniquely.
Netlists match uniquely.
Cells have no pins;  pin matching not needed.
Device classes time_delay_2.spice and time_delay_lvs.spice are equivalent.
Circuits match uniquely.
```

6.5 XOR Gate

```
Flattening unmatched subcell nandgate in circuit xor.spice (0)(4 instances)
Flattening unmatched subcell nand in circuit xor_lvs.spice (1)(4 instances)
Equate elements:  no current cell.
Equate elements:  no current cell.

Subcircuit summary:
Circuit 1: xor.spice                                    |Circuit 2: xor_lvs.spice
--------------------------------------------------------|------------------------------------------
sky130_fd_pr__pfet_01v8 (8)                             |sky130_fd_pr__pfet_01v8 (8)
```

```
sky130_fd_pr__nfet_01v8 (8)                | sky130_fd_pr__nfet_01v8 (8)
Number of devices: 16                      | Number of devices: 16
Number of nets: 12                         | Number of nets: 12
-------------------------------------------------------------------------------
Circuits match uniquely.
Netlists match uniquely.
Cells have no pins;  pin matching not needed.
Device classes xor.spice and xor_lvs.spice are equivalent.
Circuits match uniquely.
```

6.6 Bandgap Voltage Reference

```
Equate elements:  no current cell.
Equate elements:  no current cell.
Equate elements:  no current cell.
Equate elements:  no current cell.
Class opamp_square:  Merged 33 devices.
Class opamp_square:  Merged 33 devices.

Subcircuit summary:
Circuit 1: opamp_square                    | Circuit 2: opamp_square
-------------------------------------------|-----------------------------------
sky130_fd_pr__pfet_01v8 (35)               | sky130_fd_pr__pfet_01v8 (35)
sky130_fd_pr__nfet_01v8 (36)               | sky130_fd_pr__nfet_01v8 (36)
sky130_fd_pr__cap_mim_m3_1 (1)             | sky130_fd_pr__cap_mim_m3_1 (1)
Number of devices: 72                      | Number of devices: 72
Number of nets: 30                         | Number of nets: 30
-------------------------------------------------------------------------------
Resolving automorphisms by property value.
Resolving automorphisms by pin name.
Netlists match uniquely.
Circuits match correctly.

Subcircuit pins:
Circuit 1: opamp_square                    | Circuit 2: opamp_square
-------------------------------------------|-----------------------------------
VDD                                        | vdd
GND                                        | gnd
VOUT                                       | vout
V1                                         | v1
V2                                         | v2
-------------------------------------------------------------------------------
Cell pin lists are equivalent.
Device classes opamp_square and opamp_square are equivalent.
Class opamp:  Merged 33 devices.
Class opamp:  Merged 33 devices.

Subcircuit summary:
Circuit 1: opamp                           | Circuit 2: opamp
-------------------------------------------|-----------------------------------
sky130_fd_pr__pfet_01v8 (35)               | sky130_fd_pr__pfet_01v8 (35)
sky130_fd_pr__nfet_01v8 (36)               | sky130_fd_pr__nfet_01v8 (36)
sky130_fd_pr__cap_mim_m3_1 (1)             | sky130_fd_pr__cap_mim_m3_1 (1)
Number of devices: 72                      | Number of devices: 72
```

```
Number of nets: 30                          |Number of nets: 30
-------------------------------------------------------------------------------
Resolving automorphisms by property value.
Resolving automorphisms by pin name.
Netlists match uniquely.
Circuits match correctly.

Subcircuit pins:
Circuit 1: opamp                            |Circuit 2: opamp
--------------------------------------------|--------------------------------
VDD                                         |vdd
GND                                         |gnd
VOUT                                        |vout
V1                                          |v1
V2                                          |v2
-------------------------------------------------------------------------------
Cell pin lists are equivalent.
Device classes opamp and opamp are equivalent.
Equate elements:  no current cell.
Class schematics/bandgap_LDS.spice:  Merged 2 devices.
Class layout/bandgap.spice:  Merged 2 devices.

Subcircuit summary:
Circuit 1: schematics/bandgap_LDS.spice     |Circuit 2: layout/bandgap.spice
--------------------------------------------|--------------------------------
sky130_fd_pr__pnp_05v0 (51)                 |sky130_fd_pr__pnp_05v0 (51)
sky130_fd_pr__pfet_01v8 (2)                 |sky130_fd_pr__pfet_01v8 (2)
opamp_square (3)                            |opamp_square (3)
opamp (2)                                   |opamp (2)
sky130_fd_pr__res_high_po (7)               |sky130_fd_pr__res_high_po (7)
Number of devices: 65                       |Number of devices: 65
Number of nets: 14                          |Number of nets: 14
-------------------------------------------------------------------------------
Resolving automorphisms by property value.
Resolving automorphisms by pin name.
Netlists match with 1 symmetry.
Circuits match correctly.
There were property errors.
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v060 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v00 property "area" has no mat
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v01 property "area" has no mat
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v02 property "area" has no mat
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v04 property "area" has no mat
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v06 property "area" has no mat
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v08 property "area" has no mat
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v09 property "area" has no mat
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v010 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v013 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v014 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v016 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v017 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v018 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v019 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v021 property "area" has no ma
```

```
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v022 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v023 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v025 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v026 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v027 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v028 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v029 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v030 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v031 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v032 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v033 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v035 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v036 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v037 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v038 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v039 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v041 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v042 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v043 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v044 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v045 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v046 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v047 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v048 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v050 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v053 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v054 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v055 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v057 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v058 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v059 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v061 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v062 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v063 property "area" has no ma
Circuit 2 layout/bandgap.spice instance sky130_fd_pr__pnp_05v064 property "area" has no ma
Cells have no pins;  pin matching not needed.
Device classes schematics/bandgap_LDS.spice and layout/bandgap.spice are equivalent.
Circuits match uniquely.
Property errors were found.
The following cells had property errors: schematics/bandgap_LDS.spice
```