Final Project Report

CMOS Inverter and Ring Oscillator

EENG 406 01

May 7, 2024

Ryan Beck, Amelia Ellis, Hayato Tsujii, Matthew VonWahlde, Chance Wilson, Cameron Zheng

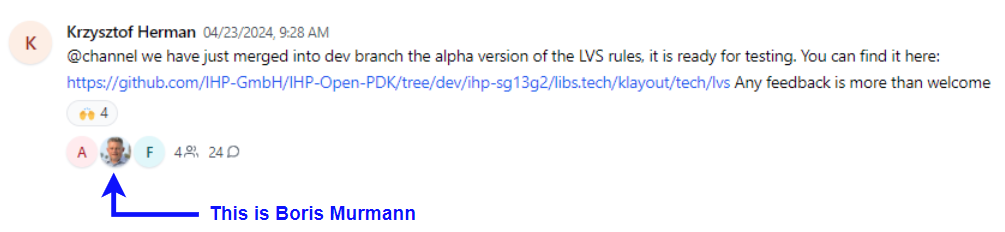
**Project Description**

The team has been tasked with creating a CMOS inverter and ring oscillator for this project. Numerous steps must be followed to reach the final post-layout versions: schematic, schematic simulation, layout, DRC, LVS, RC extraction, and post-layout simulation. The first step involves using XSchem to create a circuit for the inverter and Ngspice to run the simulation. After, the team designed a layout using KLayout. In KLayout, the Design Rule Checking (DRC) step checks to ensure the layout follows the design rules for the given technology. Then, the Layout Versus Schematic (LVS) step in Netgen checks to see if the layout matches the schematic. Afterward, the parasitic resistance and capacitances of the various wires and interconnections are calculated using Magic in the RC extraction step. Finally, Ngspice testbenches will run the post-layout simulation for the CMOS inverter. After all those steps are completed for the inverter, they will be repeated for the ring oscillator. A ring oscillator comprises an odd number of inverters connected in series (with feedback). The slight delay from each of the inverters allows the output to oscillate. The team will use five inverters to create the ring oscillator in this project.

**Technology Used**

At first, the team attempted to complete these steps using the IHP sg13g2 130nm technology. However, due to specific DRC rules, LVS file naming convention mismatches, and problems importing into Magic for RC extraction, the decision was made to switch to the sky130A technology.

The LVS scripts for the sg13g2 technology are in the alpha (testing) stage as of two weeks ago, as seen in the comment from the developers below.



**Figure** **1.** Boris Murmann responds to the update in the sg13g2 LVS technology.

Link to the thread: [**https://web.open-source-silicon.dev/c/ihp-sg13g2**](https://web.open-source-silicon.dev/c/ihp-sg13g2)

Link to the LVS script on GitHub: **https://github.com/IHP-GmbH/IHP-Open-PDK/tree/dev/ihp-sg13g2/libs.tech/klayout/tech/lvs**

**What is DRC?**

Design Rule Checking (DRC) is the stage in the integrated circuit design process where the rules of the layers are checked. Each layer has a specific size and geometric constraints. There are also many rules about the minimum spacing from another layer, or the minimum distance a layer must enclose another. This stage is done in Klayout using a pre-provided script from the creators of the technology.

Link to the IHP sg13g2 design rules: https://github.com/IHP-GmbH/IHP-Open-PDK/blob/dev/ihp-sg13g2/libs.doc/doc/SG13G2\_os\_layout\_rules.pdf

Link to the IHP sg13g2 DRC file: <https://github.com/IHP-GmbH/IHP-Open-PDK/tree/dev/ihp-sg13g2/libs.tech/klayout/tech/drc>

Link to the Skywater 130A design rules:

https://skywater-pdk.readthedocs.io/en/main/rules/summary.html

Link to the Skywater 130A DRC file: <https://github.com/laurentc2/SKY130_for_KLayout/blob/main/drc/drc_sky130.lydrc>

**What is LVS?**

Layout Versus Schematic (LVS) is the process of verifying that the layout represents the correct circuit. In this step, a spice script is created for the layout and compared with a script of the intended circuit. Then, nodes and devices are checked to make sure they match. This step was intended to be done in Klayout, but due to problems, it was done using Netgen.

Link to the IHP sg13g2 LVS file: <https://github.com/IHP-GmbH/IHP-Open-PDK/tree/dev/ihp-sg13g2/libs.tech/klayout/tech/lvs>

Link to the Skywater 130A LVS file: <https://github.com/laurentc2/SKY130_for_KLayout/blob/main/lvs/lvs_sky130.lylvs>

**What is RC Extraction?**

RC extraction is a step in the post-layout process that involves adding the resistances and capacitances of the layers to the spice layout script. In this step, Magic will create a spice file of the layout, including capacitances and resistances. This script is then run in Ngspice for circuit behavior verification.

**Inverter**

**Schematic (XSchem)**

The first step in the design process was creating the circuit schematic. XSchem was used to generate this schematic, as all team members were familiar with the tool. This process went without significant setbacks. The inverter schematic can be seen below in Figure 2. A symbol for the inverter was also generated, as seen in Figure 3. This symbol allowed for a hierarchical design when the ring oscillator was created.

A diagram of a circuit

Description automatically generated A close-up of a diagram

Description automatically generated

**Figure** **2 (Left).** Inverter schematic.

**Figure** **3 (Right).** Inverter symbol created from the schematic.

**Simulation (Ngspice)**

Once the schematic had been created, the next step was to run simulations on the design to ensure it functioned as inspected. The necessary positive and negative voltage rails were applied for the sky130A technology (1.8V and 0V), and a pulse was provided at the input of the inverter. This design can be seen in Figure 4. Transient analysis was performed to graph the output voltage compared to the input voltage. As seen in Figure 5, the circuit performed as expected, correctly inverting the input signal.

A diagram of a circuit

Description automatically generated

**Figure 5.** Simulation testbench for the inverter.

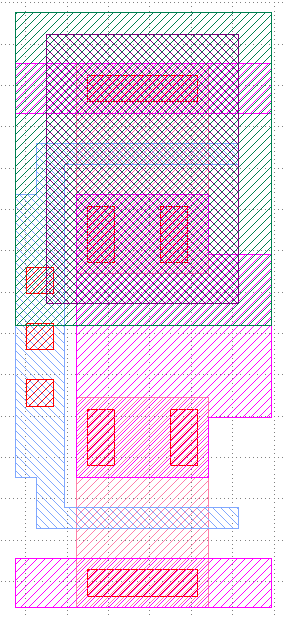
A graph with red and blue lines

Description automatically generated

**Figure 6.** Inverter waveforms from the simulation testbench. V(A) as the input (red) and V(Y)+2 as the output (blue).

**Layout (KLayout)**

At first, the layout for the inverter was done using the IHP sg13g2 technology, as seen in Figures 7 and 8. However, this design was eventually scrapped because of multiple problems throughout the design process.

A screenshot of a computer

Description automatically generated

**Figure 7. (Left)** The inverter layout in KLayout with the IHP sg13g2 technology.

**Figure 8. (Right)** The layers used to create the inverter.

Then, it was decided that the team would use Skywater 130A technology for this project. Figures 9 and 10 show the layout and layers used to create the inverter. It is important to note that the team did not build this inverter themselves. Dr. Talarico sent the magic file, and the team exported it to a GDS file that KLayout could open.

A blueprint of a house

Description automatically generatedA screenshot of a computer

Description automatically generated

**Figure 9. (Left)** Layout of the inverter using sky130A technology in KLayout.

**Figure 10. (Right)** Layers of the inverter using sky130A technology.

**Design Rule Checking (KLayout)**

In this next stage, the team uses a script to check the layout for any technology rule issues. In KLayout, a section for DRC scripts is under the tools tab. The DRC script for the technology can be imported and then run to check the design rules. Initially, when using the IHP sg13g2 technology, the team had a few checks that they could not pass, as seen in Figure 11. These rules were about the minimum percentage of a specific layer filled in an 800-micrometer by 800-micrometer chip area. The team tried using filler cells to pass this check, but the boundaries would shift, adding more chip areas to fill, as seen in Figure 12.

A screenshot of a computer

Description automatically generatedA green square with black lines

Description automatically generated

**Figure 11. (Left)** An image of the DRC screen showing the layout failing 18 tests.

**Figure 12. (Right)** An image of the layout after using filler cells to attempt to pass the tests. The number of boxes increased, and they would shift positions.

Passing the DRC phase was straightforward after switching to the Skywater 130A technology. There were minor hiccups from the conversion from a magic file to a GDS file, but all fixes were easy. Figure 13 shows the DRC screen with no errors, meaning the layout passed all design rules.

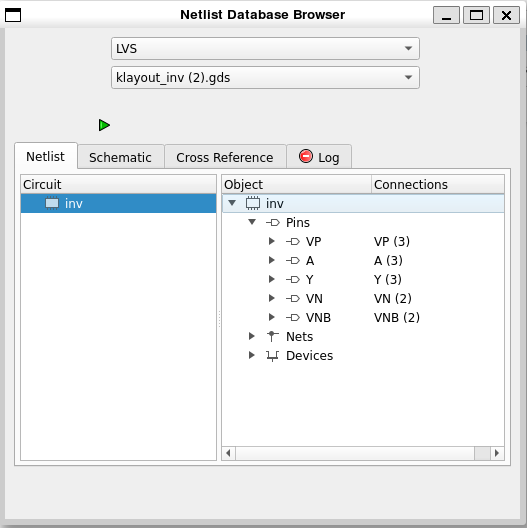
A screenshot of a computer

Description automatically generated

**Figure 13.** Inverter passing all DRC tests using the Skywater 130A technology.

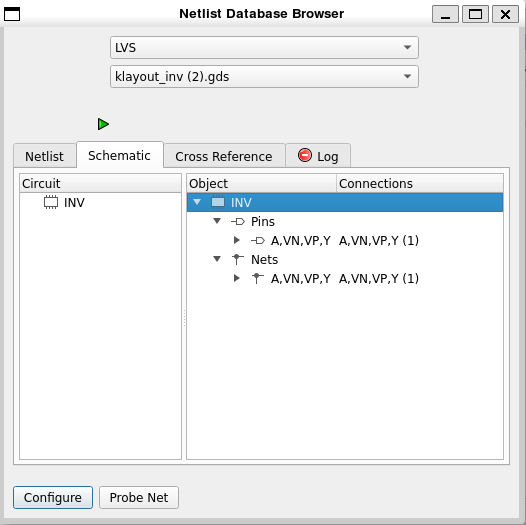
**Layout Versus Schematic (Netgen)**

The initial attempt at Layout vs. Schematic was using the sg13g2 technology in the K-Layout software, which could not be completed. As mentioned, using the sg13g2 technology, DRC could not be completed in K-Layout. When LVS was attempted, the team compared an improper layout to a correct schematic, which did not work. The LVS for the sg13g2 technology is also relatively new, and the scripts did not read the schematic correctly. Here is the layout of LVS results using the sg13g2 technology:



**Figure 14.** LVS results using the sg13g2

As mentioned, here is the improper LVS results of the schematic:



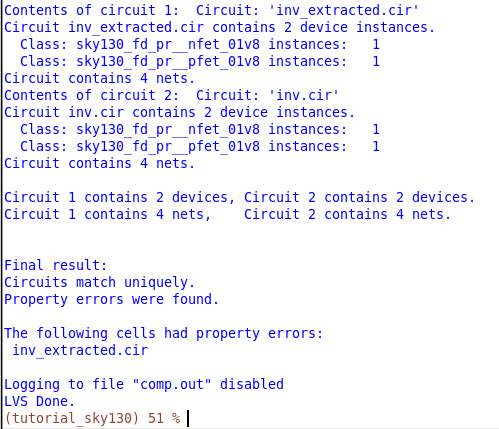
**Figure 15.** Improper LVS result using the sg13g2.

The LVS script did not find the schematic pins properly, so the LVS did not pass when it was run. After multiple attempts to correct these errors, it was determined that the sky130A technology was to be used instead. After further testing, it was found that not only did the sg13g2 technology not allow us to complete LVS, but the sky130A technology and associated LVS files would not run in K-Layout properly. However, this process could be completed much more efficiently in Netgen.

Resources from Dr. Talarico's VLSI website were used to follow the design process in Netgen: <https://web02.gonzaga.edu/faculty/talarico/vlsi/vlsi.html>

First, the layout netlist was extracted from our design in KLayout as 'inv\_extracted.cir', and the schematic netlist was extracted from xschem as 'inv.cir'. These files were used in Netgen with the provided lvs.tcl file to test circuit matching:

*lvs inv\_extracted.cir inv.cir lvs.tcl*



**Figure 16.** The inverter passed the LVS test. The phrase "Circuits match uniquely" indicates that the layout matches the schematic.

To remove all errors, the following command and associated results are below. The sky130A\_setup.tcl was in foss/pdks/sky130A/libs.tech/netgen but was moved to the current directory for easier access.

*netgen –batch lvs inv\_extracted.cir inv.cir sky130A\_setup.tcl*



**Figure 17.** The results after running the command. The circuits match uniquely, and the property errors are gone.

The layout vs. schematic was completed using sky130A technology and Netgen, and the circuits match. Aside from the problems mentioned above, a significant amount of time was spent debugging the netlist extracted from the layout. As this circuit was only one inverter, a subcircuit was not needed in the netlist, and the two transistors were the only defined parts of the netlist. The extracted netlist was generated with a subcircuit but without defining the use of the circuit, so some manual changes were needed.

The team recommends using Netgen for LVS for future project iterations until the sg13g2 technology is refined. Netgen was very streamlined and made the testing process much more efficient. Although KLayout may be able to be used, it took much longer than allowed for this project. Using Netgen and manually reviewing the layout and schematic netlists, we could learn and complete the LVS process sufficiently.

**RC Extraction (Magic)**

Once we had a layout that passed DRC and LVS, the team attempted to use Netgen to extract the parasitic components from the layout. However, we quickly realized that Netgen is primarily a tool for LVS and does not perform parasitic extraction directly. Instead, we found that this process is efficiently handled within the Magic layout tool using its built-in 'ext2spice' feature via the tkconsole interface.

Here is a step-by-step outline of the extraction process:

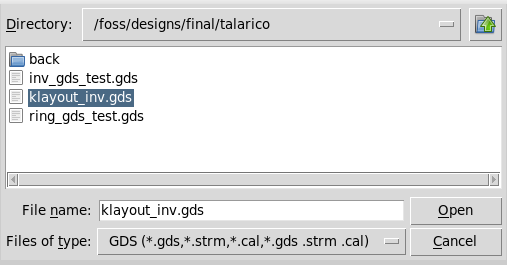
1. **Starting Magic**: Begin by typing 'magic' in the console provided by the IIC-OSIC Tools Docker. This command will open two windows: a layout window for visualization and the tkconsole for executing commands.



**Figure 18.** Command to run Magic.

1. **Opening the GDS File**:
   1. In the layout window, navigate to File > Read GDS.
   2. Locate and select the .gds file generated from the layout, then click Open.

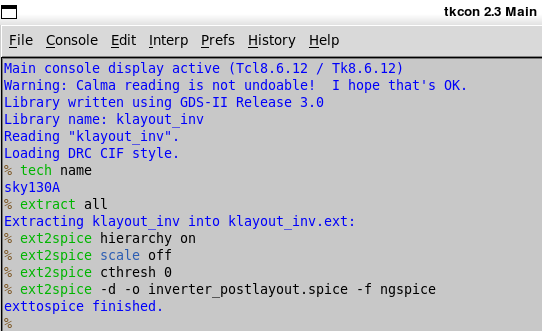
A screenshot of a computer

Description automatically generated

**Figure 19 (Left).** The button to import a GDS file in Magic.

**Figure 20 (Right).** Selecting the appropriate file for RC extraction.

1. **Executing Commands in Tkconsole**:
   1. In the Tkconsole window, execute the 'extract' and 'ext2spice' commands in the figure below.
   2. The final command, 'ext2spice -d -o <destination\_file\_name>.spice -f ngspice', will generate a netlist that includes parasitic resistances and capacitances extracted from the layout.



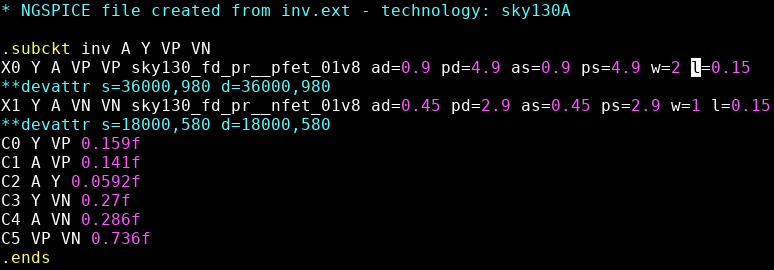
**Figure 21.** An image of the commands needed to run the RC extraction.

1. **Reviewing the Netlist**:
   1. Open the generated .spice file in a text editor to inspect the extracted RC network.
   2. The extracted file, inverter\_postlayout.spice, contains the detailed netlist with parasitic components identified from the layout.

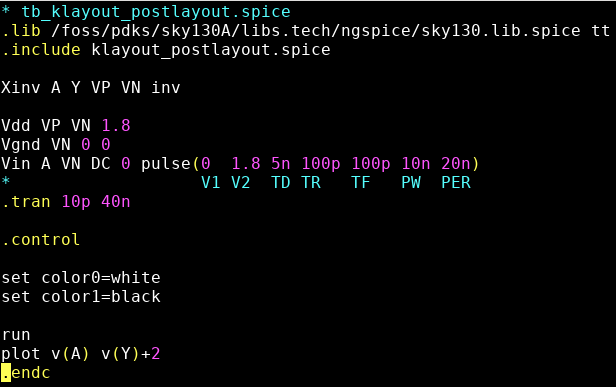
Using this approach, the team completed the RC extraction process, ensuring accurate modeling of the design's parasitic effects for post-layout simulation.

**Post-Layout Simulation (Ngspice)**

Post-layout simulation was completed in Ngspice to ensure the layout with parasitics functioned as intended. A subcircuit of the inverter was created, as can be seen in Figure 22 below. A testbench was created similar to the schematic simulation, where a pulse was applied to the input of the inverter, and the output voltage was observed in transient analysis.

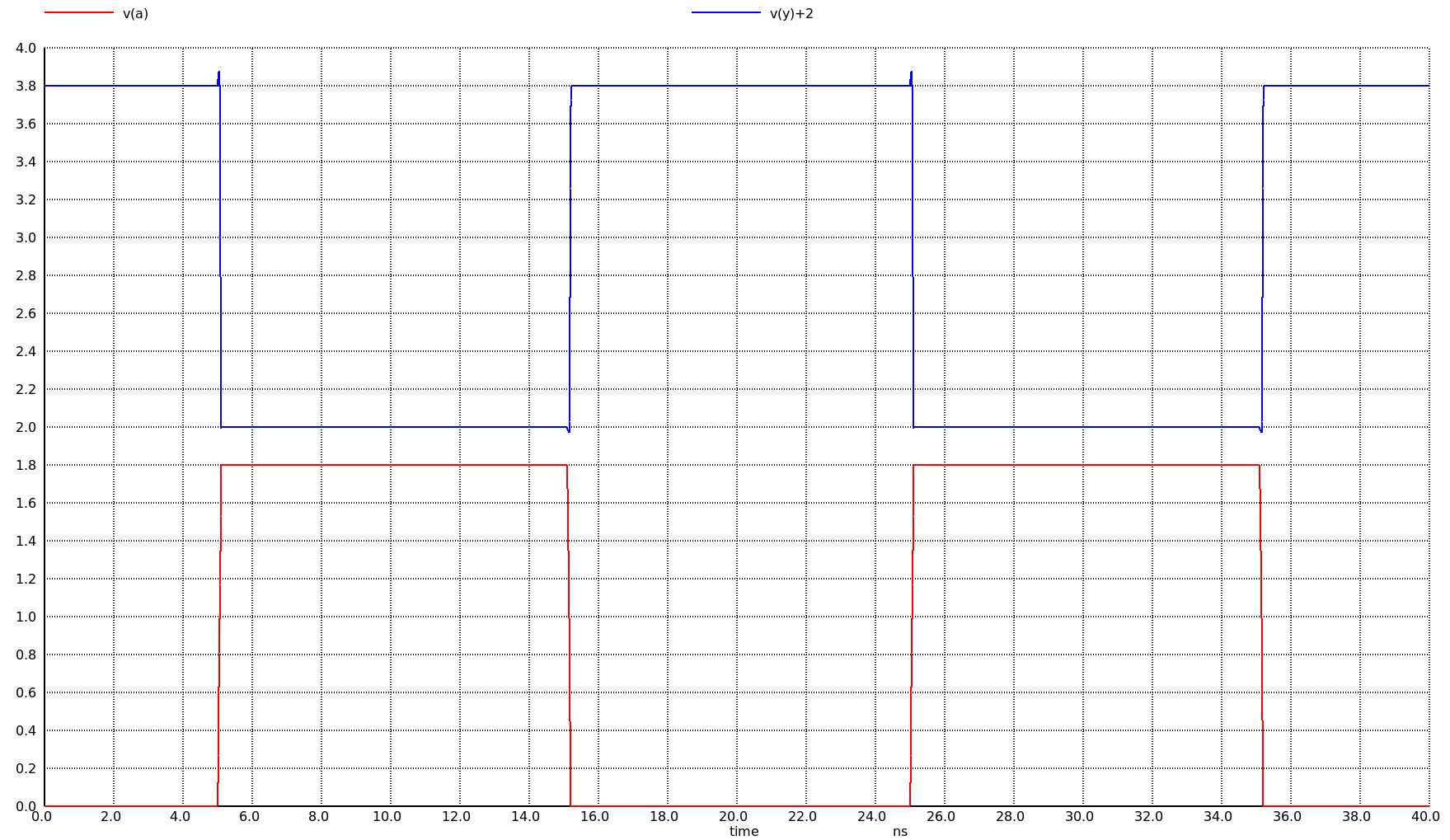


**Figure 22.** Spice netlist from the RC extraction for the inverter.



**Figure 23.** An image of the testbench used to test the netlist from the RC extraction phase.

Below are the results of the simulation. The inverter functioned as expected, and the input was successfully inverted at the output of the circuit. There were no significant setbacks in the post-layout simulation of the inverter, as the circuit was straightforward. However, this was not the same when the team tried to generate the post-layout simulation for the ring oscillator, which will be discussed in the ring oscillator section of the report.



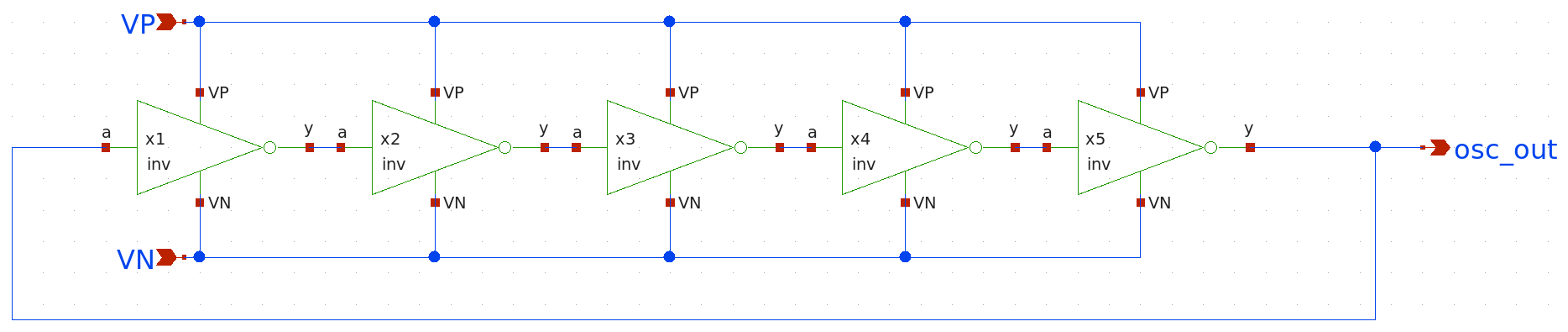
**Figure 24.** Post-layout inverter waveform, the input v(a) (red) and the output v(y)+2 (blue).

**Ring Oscillator**

A ring oscillator is a loop of an odd number of inverters connected in a chain. Looping the odd number of inverters together creates an oscillating output. The inverters created earlier in the project are used to implement a schematic (XSchem), simulation (Ngspice), layout (K-Layout), RC extraction (Netgen), and post-layout simulation (Ngspice) for a ring oscillator.

**Schematic (XSchem)**

Using the symbol created when making the inverter schematic, the ring oscillator schematic was created in Xschem. An additional symbol for the ring oscillator was created to allow for high-level testing and future circuit use in larger projects. This process had no significant setbacks and provided good experience in creating hierarchical schematics.

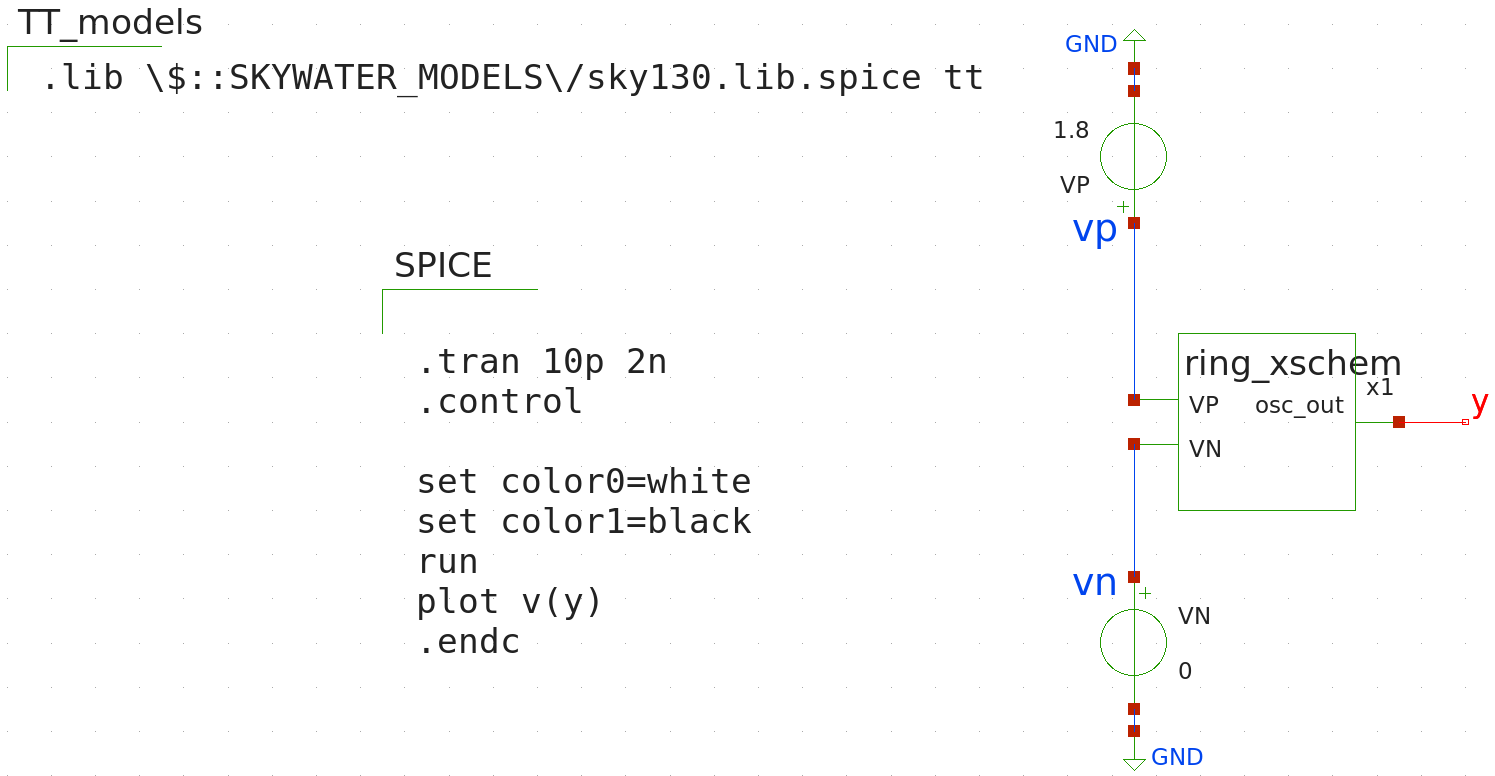
**Figure 25.** Ring oscillator schematic.

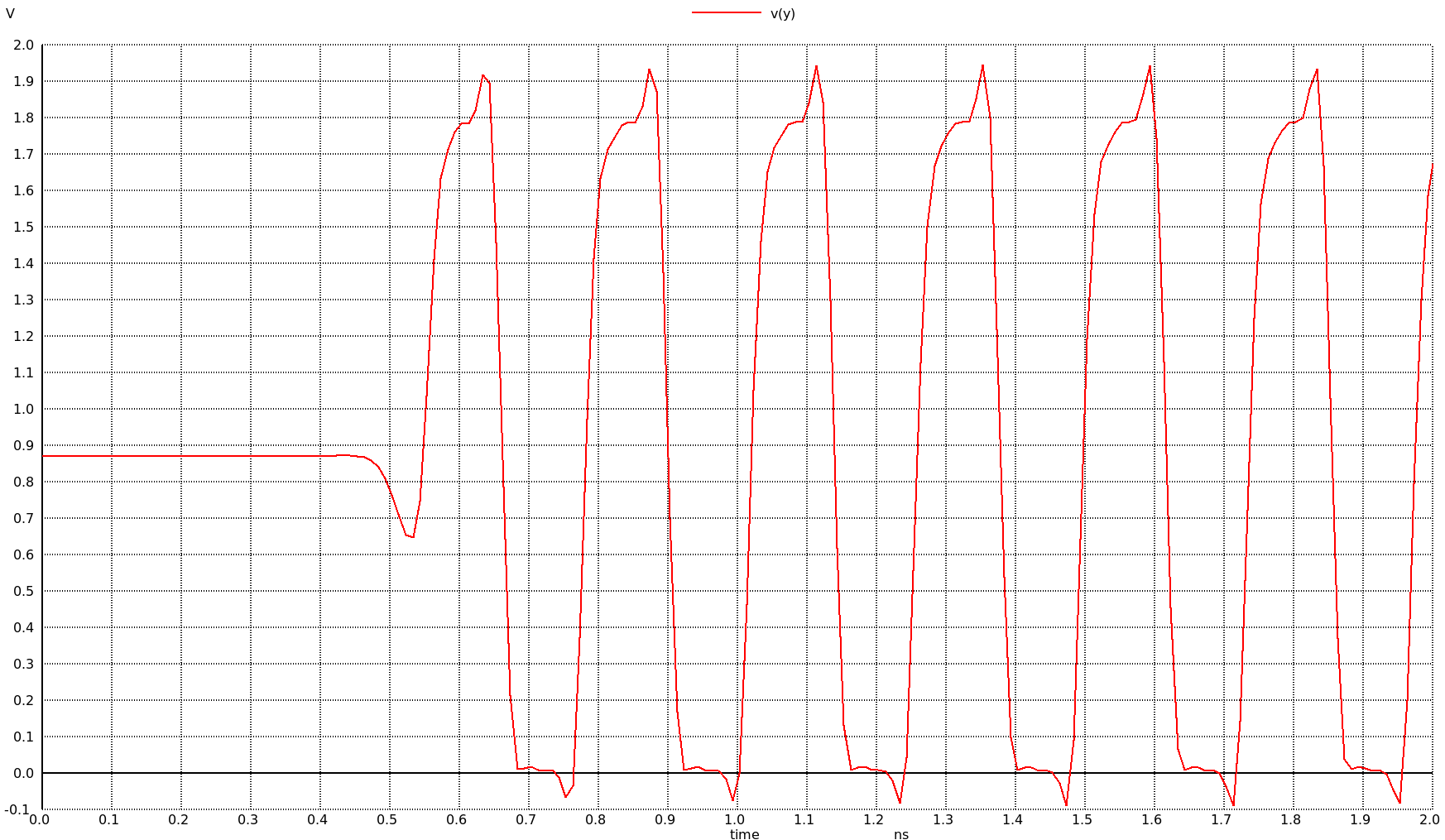


**Figure 26.** Ring oscillator symbol.

**Simulation (Ngspice)**

A simple testbench was generated in ngspice to test this circuit. By running transient analysis, the functionality of the ring oscillator was observed. The circuit operated as expected, confirming the design of our schematic. Below are the spice commands and output waveform of the ring oscillator.

**Figure 27.** Testbench simulation for ring oscillator.

**Figure 28.** Output waveform from testbench simulation of the ring oscillator.

**Layout (KLayout)**

The inverter cell from the previous section is used to build the ring oscillator. Figure 29 shows the additional cell added, called ring. This cell then declares five instances of the inverter cell. These inverters are spaced 1.85 micrometers away from each other because that is the width of the inverter cell.

A screenshot of a cell phone

Description automatically generated

**Figure 29.**

Figure 30 shows the completed ring oscillator layout in KLayout. The five inverter cells are placed next to each other. Then, a metal1 layer is placed from the rightmost inverter's output to the leftmost inverter's input. This connection completes the feedback loop.

A green and orange grid

Description automatically generated

**Figure 30.** Layout of the ring oscillator in KLayout.

**Design Rule Checking (Klayout)**

After the layout for the ring oscillator was completed, the team worked on rechecking the design rules. This check used the same script as the inverter, and the tests passed on the first try. This phase of the post-layout simulation was very straightforward, and the team did not encounter any problems. Figure 31 shows the completed DRC check.

A screenshot of a computer

Description automatically generated

**Figure** **31.** Ring-Oscillator passing all DRC tests.

**Layout Versus Schematic (Netgen)**

After researching LVS during the design of the inverter, the LVS for the ring oscillator was completed using the sky130A technology and netgen. As the team better understood the tools and netlist architecture at this point, the LVS for the ring oscillator did not take as long as the LVS.

The LVS was able to be completed with and without a hierarchical design. The same commands used in the inverter LVS were repeated here with the netlists extracted from the KLayout layout and the Xschem schematic. The circuits matched, and LVS was completed:

*netgen –batch lvs ring\_spice ring\_xschem.spice sky130A\_setup.tcl*

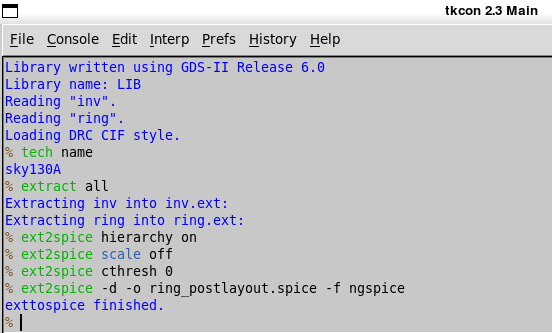


**Figure 32.** LVS result using the sky130A.

It is important to note a significant challenge in this process. When the netlist was initially generated from KLayout, the instances of the inverters were created as an entire subcircuit called 'ring.' When the LVS was run with the entire circuit being a subcircuit, which was done in Post-Layout Simulation to create a testbench, the LVS failed, and the layout had one less net than the schematic. To test our design, we remade the circuit in a non-hierarchical design, where the netlist was ten transistors and no subcircuits. This netlist passed LVS, leading us to the conclusion that removing the unnecessary subcircuits was needed. Therefore, we completed LVS using a hierarchical design after some manual review.

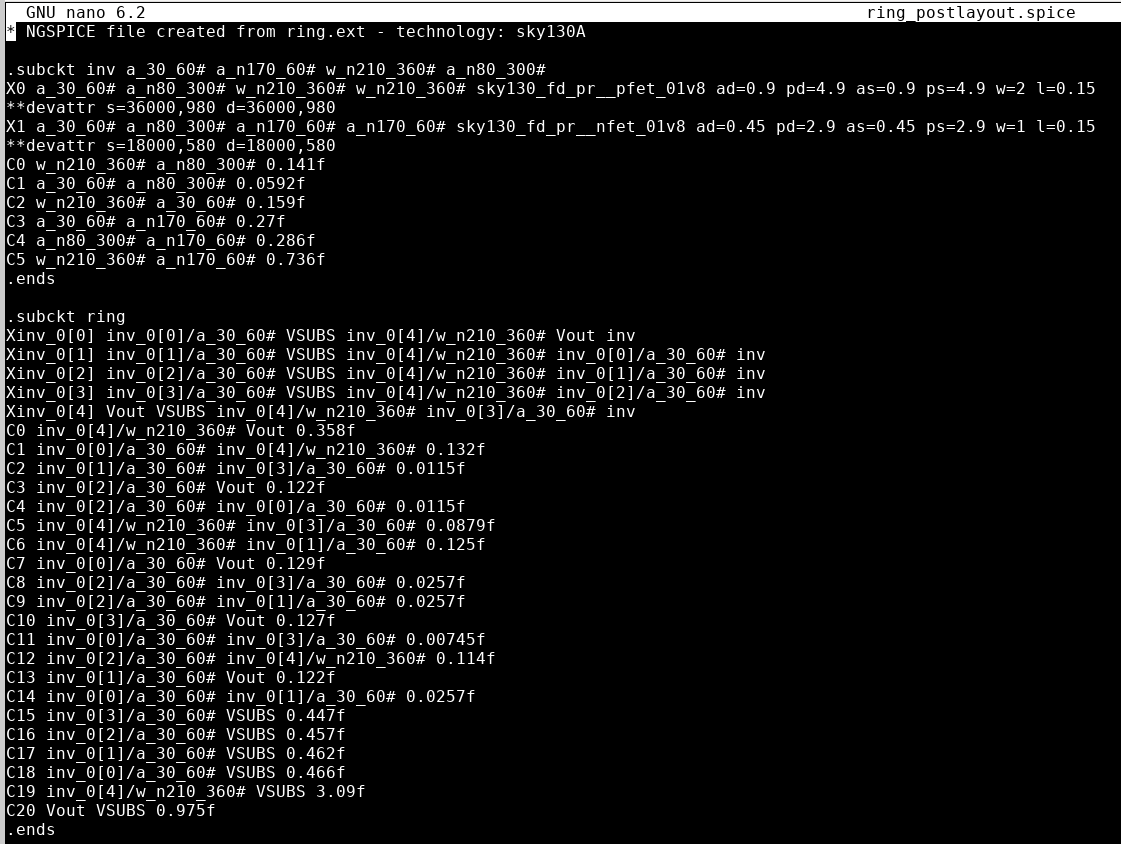
**RC Extraction (Magic)**

As described in the inverter RC Extraction, load the ring oscillator layout in magic/netgen and run these commands:



**Figure 33.** Commands to load layout in magic/netgen

The analysis generated the following file:



**Figure 34.** Results generated for RC Extraction

**Post-Layout Simulation (Ngspice)**

Once the parasitics had been generated for the ring oscillator, post-layout simulation in Ngspice was completed to test the functionality of our layout. However, this process had several setbacks. First, the netlist generated by the layout had very hard-to-understand net names, such as w\_n210\_360#. These names required the team to spend significant time mapping the netlist to the circuit to ensure our testbench connected the external signals correctly. Additionally, using a hierarchical design took time to learn, and ensuring the inverters were referenced and set up correctly in the netlist took much longer than expected. All challenges were resolved, and a testbench was created for the ring oscillator. Below are the outputs of the simulations, where the red signal is the system's output voltage. The simulation results with and without parasitics are included.

A computer screen with text and numbers

Description automatically generated

**Figure 35.** Spice testbench for the ring oscillator

A graph with red and blue lines

Description automatically generatedA graph of a graph

Description automatically generated with medium confidence

**Figure 35 (Left).** Ring-oscillator post-layout simulation using a hierarchal model

**Figure 36 (Right).** Ring-oscillator post-layout simulation using a hierarchal model after resistance and capacitances.

**Design Flow Conclusions**

All stages of the design process assigned to the team were completed. The main changes in execution came from which software was used rather than from being unable to finish any part. The sky130A technology allowed the design process to be completed and implemented into the schematic and layout with little setbacks compared to the sg13g2 technology. Xschem and Ngspice allowed for efficient schematic design and simulation, and practicing using these tools earlier in the year was very beneficial. KLayout is a good tool for layout and DRC for the sky130A but is not currently prepared for LVS. Significant testing is required before switching away from the tools that worked well, such as Netgen and Magic. Netgen was excellent at LVS and provided better feedback than KLayout, allowing for easier manual debugging. Magic proved to be the best RC Extraction tool, resulting in no significant problems when running this tool. Ngspice again allowed for efficient Post-Layout Simulation.

**Future Recommendations**

If this project were completed again, it would be vital to allow the sg13g2 tools to be developed more before use, specifically LVS. KLayout proved useful in layout design and DRC checking and could be used instead of Magic if preferred. However, Netgen and Magic proved to be the most efficient in LVS and RC Extraction and should be used again unless KLayout LVS is researched more. Overall, starting with the simple design of the inverter was very beneficial in allowing the team to practice using the tools. A short initial netlist allowed for easier debugging and led to less time focused on the schematic and more time on the more challenging parts of this project. The experience with the design flow allowed the team to complete the ring oscillator process more easily.