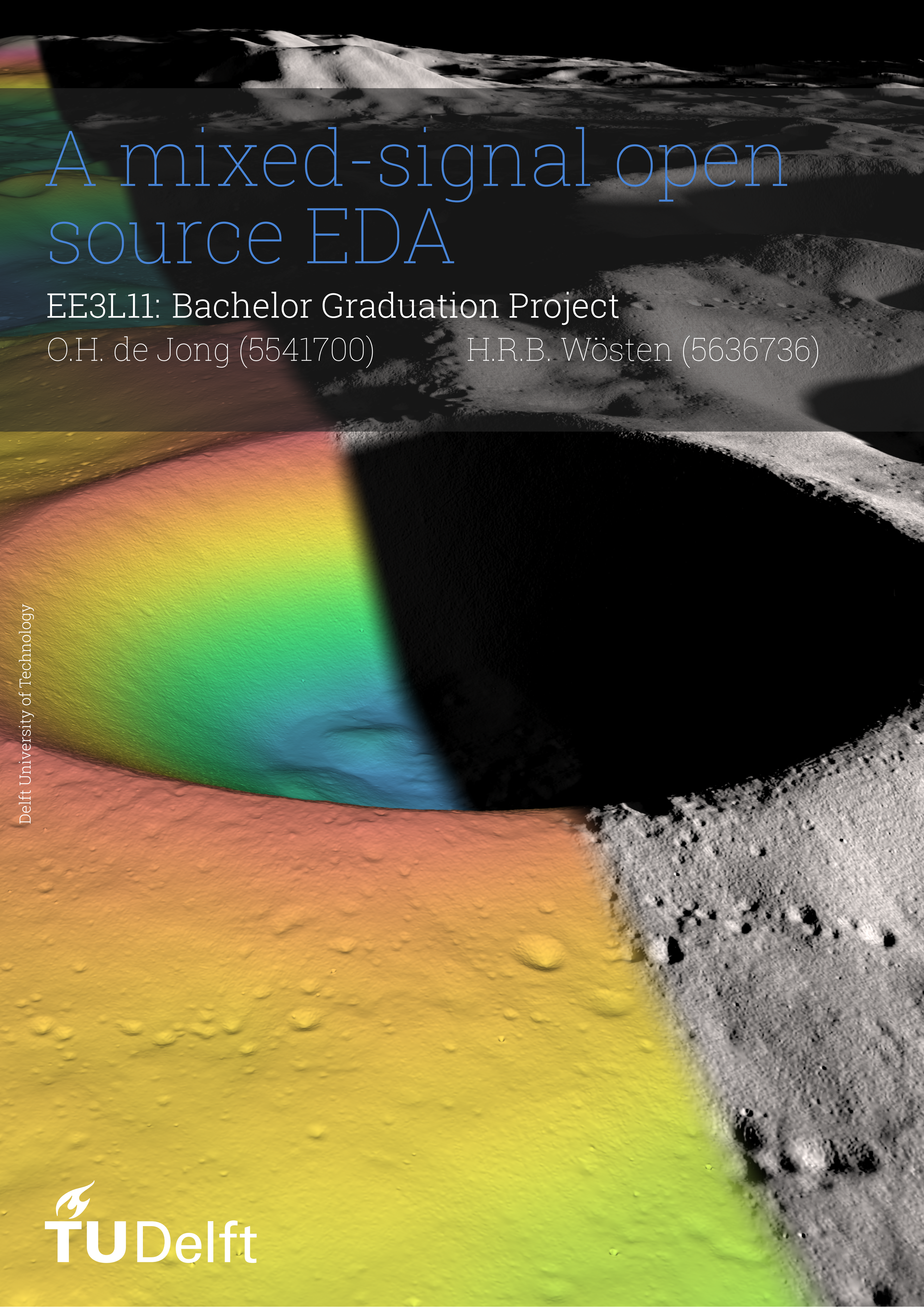


A mixed-signal open source EDA

EE3L11: Bachelor Graduation Project

O.H. de Jong (5541700) H.R.B. Wösten (5636736)

Delft University of Technology

TU Delft

A mixed-signal open source EDA

by

O.H.de Jong, H.R.B.Wösten

The two authors equally contributed to this work.

Supervisor: F. Sebastiano
Project Duration: May, 2025 - June, 2025
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover: A visualization of Shackleton crater.
Image Credit: NASA’s Scientific Visualization Studio
Source: https://svs.gsfc.nasa.gov/4716

TU Delft

Abstract

This thesis presents the development of an open-source electronic design automation (EDA) toolchain for mixed-signal integrated circuit (IC) design using the Skywater 130nm PDK. The toolchain integrates digital, analog, and mixed-signal flows using OpenLane, Xschem, Ngspice, and ALIGN, among others. It automates key steps such as schematic simulation, layout generation, and verification while remaining accessible and modular. A major contribution is the demonstration of cryogenic simulation support using extracted parameters from transistor measurements at 4K, showcasing the potential for space and quantum applications. The work also includes Docker-based deployment to ensure reproducibility and ease of use across platforms.

Preface

This thesis was written as part of our bachelor's degree in Electrical Engineering at Delft University of Technology (TU Delft). It reflects our joint work on developing an open-source electronic design automation (EDA) toolchain for mixed-signal integrated circuit (IC) design. The toolchain integrates both digital and analog design flows and explores cryogenic simulation capabilities for use in extreme environments.

This project is part of a larger initiative, An Open-Source Platform to Develop Wireless Nodes for Lunar Exploration, with our subgroup specifically focused on the development of an open-source mixed-signal EDA toolchain. Our work contributes to the overall effort to create accessible and flexible design tools for use in the challenging environments of space exploration.

The motivation for this project arose from the limitations of commercial EDA tools, which are often inaccessible due to high costs and restrictive licensing. Our aim was to create an open, modular, and partially automated toolchain that supports innovation, education, and rapid prototyping. Throughout the project, we worked with tools such as OpenLane, Magic, Xschem, Ngspice, and ALIGN, and implemented cryogenic transistor modeling using Python-based parameter extraction.

We would like to thank our supervisor, Fabio Sebastiano, for the valuable guidance and support during this project.

Olaf de Jong
Ruben Wösten

Delft, June 2025

Contents

1
Introduction

1.1. Motivation and context

Space agencies, such as NASA and ESA, along with private companies, are focused on achieving permanent human presence on the Moon[1]. While the current longest lunar mission lasts 74 hours, key challenges must be addressed before extending human stays, such as safety, optimal landing locations, and resource availability.

The Moon's harsh environment—extreme temperature variations, lunar dust, unfiltered cosmic radiation, and long nights (about 14 Earth days) poses significant risks. Of these, radiation remains the least understood yet one of the most dangerous threat. Without an atmosphere or magnetic field, the Moon's surface is directly exposed to solar particle events and galactic cosmic rays, which can harm both equipment and astronauts[2]. Mapping radiation and gathering more data is crucial to ensure safety.

The lunar south pole, especially locations like Shackleton crater, is considered the most optimal site for long-term human survival. Certain peaks and crater rims receive almost constant sunlight (up to 80-90 %)[3], while some craters remain permanently shadowed, potentially holding frozen water. This water could be converted into essentials like water, oxygen, fuel, and life support for astronauts.

Integrated circuit (IC) design plays a crucial role in enabling the sensor nodes to operate reliably under extreme lunar conditions. Custom ICs allow for ultra-low power consumption, high integration density, and radiation tolerance—essential for long-term deployment in space with strict size and energy constraints.

1.2. Purpose of project

Rapid innovation in integrated circuit (IC) design has led to an increasing demand for accessible, flexible and cost-effective design tools [4]. In recent history, the semiconductor design tool market has been dominated by proprietary electronic design automation (EDA) software. These tools often require expensive licenses and typically involve signing non-disclosure agreements (NDAs) [5]. While this may be manageable on a small scale, it becomes problematic when collaboration involves hundreds of individuals across different organizations, as legal restrictions can significantly hinder coordination. Additionally, licenses usually grant access only to the executable software, not the source code, which limits flexibility and can stifle innovation.

In response, the development of open-source EDA tools has accelerated. These tools are free to use and provide opportunities for researchers, educators, and start-ups to collaborate on IC design projects, encouraging further innovation. Currently, only a few open-source EDA tools support a complete flow for digital IC design. For analog and mixed-signal design, the situation is more complex. Although tools exist for individual stages, the design process still requires significant manual effort, making it less accessible and more prone to errors. This research aims to bridge that gap by developing an EDA toolchain that enables automated design for digital, analog, and mixed-signal ICs.

1.3. Background Knowledge

IC design has different stages. There are different stages for analog, digital, and mixed signal circuits. Mixed signal designs require both the digital and analog signal process flow. Figure 1.1 (on the next page) shows the complete flows for analog, digital, and mixed signal designs.

The design process starts by designing the individual blocks. These are analog, digital, or mixed-signal components. For the analog input, either a schematic or both the flattened and hierarchical netlists of the block are required. For a digital component, a Verilog file is needed. For a mixed signal block for simulations on the functionality of the design a python file is needed and for the layout a schematic is needed.

For the layout of digital components, the Verilog file can be put in the toolchain Openlane. This runs the complete digital design flow. In this flow is, the verilog file needs to be compiled in a compiler. After the code is compiled, it needs to be synthesized into gate-level netlist. The netlist is then placed and routed using a place-and-route tool. Lastly there is the timing analysis to check whether the design doesn't violate any timing constraints.

The analog flow starts with creating a schematic for the analog blocks. Then, this schematic is simulated to check the functionality. If the simulation results are correct, the schematic is converted into a transistor-level layout. This layout is then put through DRC, PEX, and LVS checks. PEX (Parasitic Extraction) extracts parasitic resistances and capacitances from the layout, which are then used in post-layout simulation to verify if the circuit still meets its functional requirements.

When all the individual blocks are designed then the complete layout can be made. This is a difficult objective to achieve automatically. Therefore IC designer usually do this by hand. With time in consideration this is left manual. When this step is finished the layout can be simulated and the last checks can be performed. After the layout simulation is complete and shows the functionality of the chip working then the layout is ready for sign off.

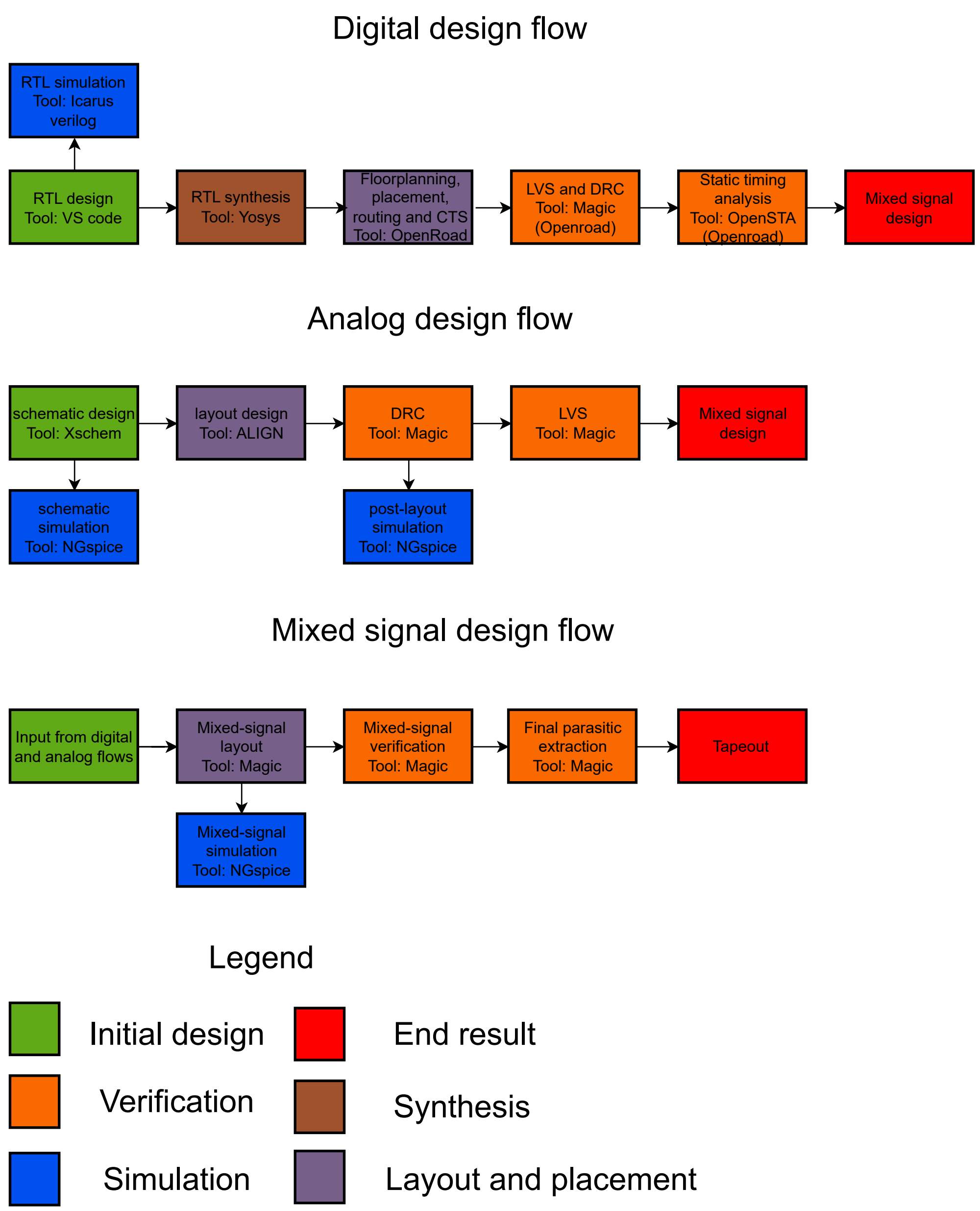


Figure 1.1: Analog, digital and mixed signal design flows

1.4. State-of-the-art analysis

According to a Reuters report, Synopsys, Cadence[6], and Siemens EDA[7] dominate the electronic design automation (EDA) market, controlling over 70 percent of the market share in China, particularly in the design of integrated circuits (ICs) [8]. These tools support the full design process for analog, digital, and mixed-signal circuits. However, they are very expensive and often not available to students and researchers because they require paid licenses and strict legal agreements [9].

1.4.1. Open-source EDA Tools

In recent years, more and more open-source EDA tools have been developed to try to replace parts of the expensive commercial tools[9]. Some well-known examples are:

- **OpenLane**: A digital design toolchain based on the Sky130 PDK. It provides a full RTL-to-GDSII digital flow, including synthesis (Yosys), placement (RePlAce), and routing (TritonRoute)[10].
- **Magic**: A tool for layout design and DRC checks, widely used in both academic and open-source contexts[11].
- **Xschem and Ngspice**: Used for schematic entry and simulation in analog design flows[12][13].
- **Netgen**: For layout vs. schematic (LVS) verification[14].
- **ALIGN**: A more recent tool that attempts to automate analog layout generation using constraint-based templates.

Despite these developments, few tools support a complete mixed-signal design flow. While digital tools are becoming robust and more automated, analog and especially mixed-signal designs still lack integration and automation. Analog design still requires a lot of manual work and depends on experience and guesswork[15].

1.4.2. Limitations in Mixed-Signal Flows

Mixed-signal design presents unique challenges:

- **Interface Management**: Digital and analog parts work in different ways and use different design methods, which makes connecting them together correctly more difficult[16].
- **Simulation Complexity**: Co-simulation of digital and analog blocks typically requires dedicated mixed-signal simulators (e.g., Cadence AMS Designer), which have no viable open-source counterpart that supports complex designs.
- **Lack of Unified Tools**: No single open-source framework currently manages schematic capture, simulation, layout, and verification across digital and analog in an integrated way[16].

1.4.3. Need for Integration

Right now, the open-source EDA tools are improving, but they are still separate pieces and don't work well together for mixed-signal design[17]. Because of this, the design process is less efficient, more likely to have mistakes, and harder for beginners to use. This thesis aims to solve this problem by creating a modular, open-source EDA toolchain for mixed-signal IC design that brings together digital, analog, and mixed-signal steps into one easy-to-use and semi-automated flow.

1.4.4. Mixed-signal simulations

Verilog-AMS (Analog and Mixed-Signal) is a hardware description language that extends Verilog to include analog behavior and is widely used in commercial EDA tools such as Cadence Spectre and Synopsys VCS AMS. These platforms support co-simulation of analog and digital blocks within a single simulation kernel, offering tight integration, accurate signal synchronization, and robust convergence algorithms [18], [19]. However, in the open-source domain, support for Verilog-AMS remains limited. Tools like Ngspice and Icarus Verilog lack a shared simulation kernel for mixed-signal operation and cannot natively interpret Verilog-AMS constructs. This gap exists primarily because Verilog-AMS is a complex standard that is difficult to implement without significant development resources, and its widespread use is often entangled with proprietary models and licensing restrictions [20]. As a result, most open-source workflows rely on a modular or co-simulation approach that connects separate digital and analog simulators using external scripts or intermediate data formats.

1.5. Problem Analysis

1.5.1. Scoping analysis

Needs: There is a growing need for an open, accessible, and automated mixed-signal IC design flow. Researchers and students lack the resources to use commercial tools, yet require robust design platforms to prototype and verify their ideas. This need is especially pressing in emerging fields like cryogenic electronics, where standard EDA tools and models are not readily available or validated at low temperatures.

Objectives:

- To develop an open-source toolchain that supports the complete mixed-signal design flow.
- To simplify the co-design and co-simulation of analog and digital components.
- To lower the entry barrier for new designers and educational institutions.
- To enable integration of cryogenic models into the open-source PDK (sky130).

Success Criteria:

- Integration of digital (OpenLane), analog (Xschem, Ngspice), and mixed-signal simulation steps.
- Usability: The toolchain must be usable by someone with only basic knowledge of EDA tools.
- Automation: Significant reduction in manual design steps, especially for layout integration and simulation.

1.5.2. Bounding Analysis

Constraints:

- Dependence on the maturity and limitations of existing open-source tools (e.g., ALIGN is still under development).
- Lack of validated SPICE models for cryogenic operation. Only primitive behavioral models are available.
- Tool compatibility and data exchange formats across different stages.

Parameters:

- Design size and complexity (small to medium-scale mixed-signal blocks).
- Target PDK (Sky130).
- Performance trade-offs between automation and flexibility.

Variables:

- Design time.
- Accuracy of simulation results.
- Number of manual interventions required.

This project is focused on improving the design situation by creating a more unified workflow rather than reinventing existing tools. It builds on and connects current capabilities in digital and analog flows, aiming to reduce user errors and design time while broadening accessibility.

1.6. Thesis Synopsis

This thesis presents the development of an open-source toolchain for the design and simulation of digital, analog, and mixed-signal integrated circuits (ICs), with an emphasis on automation, modularity, and accessibility. Aimed at supporting the Sky130 process design kit (PDK), the toolchain integrates a range of open-source tools including Xschem, Ngspice, Magic, Netgen, and OpenLane—into a unified, flexible workflow that spans schematic capture, simulation, layout, and verification.

The project focuses on four main objectives. The first is to automate the analog design flow, enabling users to transition from schematics to verified layouts with minimal manual intervention. The second objective is to implement a co-simulation environment that verifies the functionality of complete chips, bridging the gap between analog and digital domains. Third, a device-level simulation using a cryogenic model from the PDK was performed to explore the feasibility of supporting low-temperature circuit design. Lastly, the entire environment is encapsulated in a Docker container to ensure ease of installation, portability, and reproducibility across systems.

Overall, the thesis contributes a practical framework that lowers the barrier to entry for IC design using open-source tools, supporting the growth of the open hardware ecosystem.

2

Programme of Requirements

The goal of this project is to develop an open-source, modular toolchain that supports the complete design flow of mixed-signal integrated circuits (ICs). The system should integrate existing open-source tools for digital, analog, and mixed-signal design, while automating key steps to improve usability, reduce manual intervention, and lower the learning curve. Additionally, the project explores the integration of cryogenic PDK models and ensures system portability through Docker-based deployment. The requirements are categorized as follows:

2.0.1. Functional Requirements

These requirements define the essential operations that the toolchain must perform to support mixed-signal IC design.

[A.1] The system must support schematic-based analog design and simulate it using Ngspice.

[A.2] The system must support the digital RTL-to-GDSII flow using OpenLane and allow digital simulation using Icarus Verilog.

[A.3] The system must support co-simulation of analog and digital blocks for functional verification.

[A.4] The toolchain must allow for layout-versus-schematic (LVS) and design rule checking (DRC).

[A.5] The toolchain must support netlist extraction (PEX) and post-layout simulation of analog blocks.

[A.6] The entire toolchain must be containerized using Docker, enabling consistent execution across different operating systems (Linux, Windows, macOS) with minimal setup.

[A.7] The toolchain must support integrated placement and routing of analog and digital blocks, enabling mixed-signal physical design.

2.0.2. System Requirements

These requirements address usability, extensibility, and technical design decisions to ensure the toolchain works effectively in practical settings.

[B.1] The toolchain must work entirely with open-source software.

[B.4] The workflow should minimize manual intervention, particularly in layout generation and simulation.

[B.5] The toolchain must be compatible with the Sky130 process design kit (PDK).

[B.6] The system must include a Dockerfile that encapsulates all dependencies and enables consistent deployment across systems.

[B.7] The SkyWater130 PDK must be modified to support cryogenic operation.

3 Docker

To simplify the installation and setup process of the various EDA tools used in this project, a Docker-based environment was created using Docker Compose. In recent years, the use of Docker containers has become increasingly common in electronic design automation (EDA) environments, allowing for easier deployment and consistent toolchain setups across different platforms [21]. This approach enables a consistent and portable setup that can be easily deployed on any system supporting Docker. The goal was to implement each part of the toolchain analog design, digital design using OpenLane, layout generation using ALIGN, and Verilog simulation using Icarus into separate containers while sharing a common working directory and PDK installation to allow data exchange between tools.

The Docker Compose configuration defines four main services: align, analog, openlane, and icarus. Each service runs in its own container using a purpose-built Docker image. These containers share a mounted volume named shared, which includes both the design workspace and the local PDK files. This structure ensures that all tools operate on the same set of data, eliminating the need to copy files between containers.

3.1. Analog Container

The analog container is designed for analog IC development and includes tools such as Ngspice, Magic, and Xschem. These tools are built from source within the container to ensure compatibility and access to the latest features. The container is configured with GUI support through X11 forwarding, allowing tools like Magic and Xschem to be launched with their graphical interfaces from within the container. All commands can also be executed through the terminal, supporting automation workflows. The DISPLAY environment variable is set to enable GUI access on the host system. The specific Docker file for creating the analog container can be see in Appendix A.1.2.

3.2. Digital Design Container

The openlane container is based on the official OpenLane image but is extended to include Python, Volare, and a configured Sky130A PDK environment. Volare is used to manage PDK versions. The container mounts the shared volume, allowing digital design files to be placed and processed with minimal manual setup. OpenLane runs entirely from the command line, making it suitable for scripting and automated workflows. The specific Docker file for creating the openlane container can be see in Appendix A.1.3.

3.3. Layout Generation Container

The align container is responsible for automated layout generation of analog blocks using the ALIGN framework. It uses a custom-built image that includes ALIGN pre-installed and configured to work with the Sky130 PDK. ALIGN requires a flattened netlist, which is generated earlier using Xschem. The results are stored in the shared workspace, and the align container mounts the shared volume and operates from the '/workspace' directory. The specific Docker file for creating the openlane container can be see in Appendix A.1.4.

3.4. Verilog Simulation Container

The icarus container is used for Verilog simulation and is based on the Icarus Verilog image. It shares the same workspace as the other containers, ensuring that simulation files are easily accessible and processed within the same directory structure. Icarus Verilog is an open-source simulator, and this container allows simulation of digital designs in the toolchain. For the icarus container we have taken a publicly available Docker image.

3.5. Conclusion

By structuring the environment in this way, each part of the toolchain can run independently, while still contributing to the same overall design process. The use of Docker Compose allows the entire setup to be launched with a single command, dramatically reducing installation time and the potential for configuration errors. This is especially useful in collaborative environments or for onboarding new users, as the setup is fully encapsulated and version-controlled.

In summary, the Docker Compose-based setup enables reproducible, isolated, and automated IC design workflows for analog, digital, and mixed-signal circuits. Each container is tailored to a specific stage of the design process but shares a unified file structure and PDK environment, providing both flexibility and consistency across the toolchain.

4

Digital design

The digital design flow is fully automated with OpenLane, an open-source toolchain that handles the entire process from RTL (Register Transfer Level) to GDS (Graphic Data System) layout. OpenLane integrates tools for synthesis, placement, routing, and timing closure, providing a complete solution for digital chip design.

Although OpenLane uses Verilator for digital simulation, we opted to include Icarus Verilog (or Iverilog) for mixed-signal simulation. This was primarily because our mixed-signal testbenches are written in Verilog, and Icarus Verilog integrates more seamlessly with the analog simulation tools, facilitating easier co-simulation of digital and analog blocks.

To validate OpenLane's effectiveness, we tested it by running a design from the hardware subgroup through the entire flow [22]. We decided to put the radio component they designed in verilog trough the design flow[23]. The processed design was successfully synthesized, placed, routed, and laid out. The result can be seen in the layout image below.

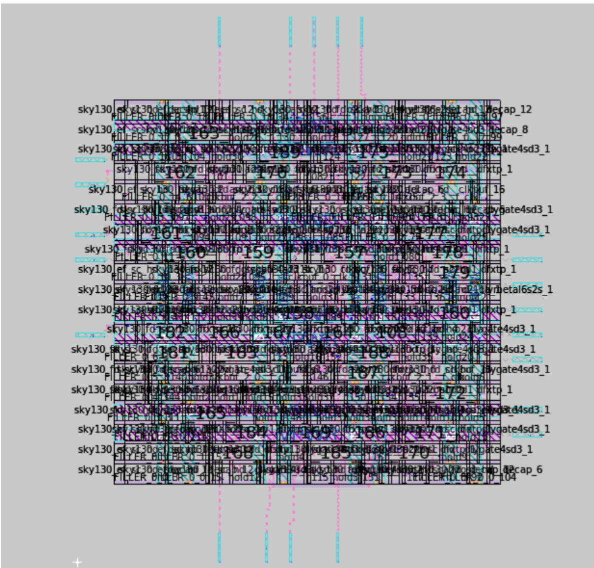


Figure 4.1: Layout of the design processed through the OpenLane flow.

This test confirmed the efficiency of OpenLane in real-world applications, demonstrating its value in streamlining the digital design process.

5

Analog design

5.1. Tool Selection Criteria and Rationale

To automate the analog design flow, open-source tools need to be identified for each stage. These are documented in table 5.1

Table 5.1: Selected open-source tools for analog design flow

Design Stage	Tool 1	Tool 2	Tool 3
Schematic Entry	Xschem	Electric VLSI	-
Netlist Generation	Xschem	Electric VLSI	-
Analog Simulation	Ngspice	Xyce	Gnucap
Layout Design (Manual)	Magic	Electric VLSI	KLayout
Layout Design (Automatic)	ALIGN	Laygo	KLayout + Python
DRC, LVS, and PEX Checks	Magic	Netgen	-
Post-Layout Simulation	Ngspice	Electric VLSI	Gnucap

The following criteria are considered to choose the best tool:

- Is the tool controllable via terminal commands?
- Does the tool integrate easily with the next stage?
- Does the tool offer good user support and have a user-friendly interface?

With these criteria in mind, the following tools are chosen:

Schematic Entry: Xschem is selected because it provides a user-friendly GUI for schematic design while also supporting batch mode for command-line control, enabling easy automation. Unlike Electric VLSI, Xschem offers better integration with downstream tools through its ability to generate both hierarchical and flattened netlists, which are essential for simulation and layout generation.

Netlist Generation: Xschem again is preferred because it can generate the netlist formats required by both simulation tools and layout engines. Electric VLSI offers netlist generation as well, but with less flexibility in netlist formatting, making Xschem more suitable for integration with ALIGN and Ngspice.

Analog Simulation: Ngspice is chosen for its seamless integration with Xschem and its command-line controllability, which is crucial for automated workflows and co-simulation scenarios. While tools like Xyce and Gnucap are capable simulators, Ngspice's wider community support and straightforward compatibility with Xschem make it the preferred choice.

Layout Design (Manual): Magic is selected due to its mature set of features for manual layout editing and its command-line driven DRC and LVS checks. Electric VLSI provides manual layout capabilities but lacks some of the automation and verification features offered by Magic. KLayout is also a strong contender, particularly for layout visualization, but Magic's integrated environment and scripting capabilities make it more suitable for manual layout work in this flow.

Layout Design (Automatic): ALIGN is chosen for automatic layout generation of analog blocks because it is specifically designed for this purpose and works best with flattened netlists from Xschem. Laygo and KLayout combined with Python scripts are alternatives, but ALIGN's ongoing development and focus on analog layout automation give it a clear edge for this flow.

DRC, LVS, and PEX Checks: Magic and Netgen are selected because both can be controlled via terminal commands and integrate well with Magic layouts. These tools are widely used in open-source flows and offer reliable verification of layout correctness. No viable alternatives in the list combine ease of integration and command-line control as effectively.

Post-Layout Simulation: Ngspice is again preferred for post-layout simulation due to its compatibility with the netlists generated by Xschem and its command-line interface, allowing easy automation. Electric VLSI and Gnucap also provide simulation capabilities but with less seamless integration in this workflow.

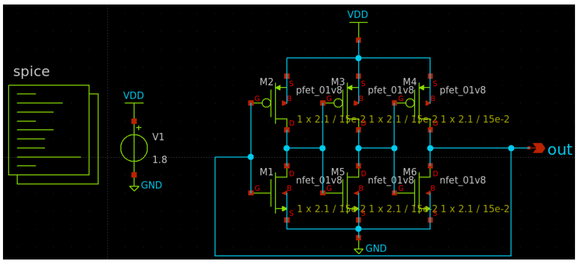
5.2. Validation: Ring Oscillator

To demonstrate the complete functionality of the open-source analog design flow, a simple ring oscillator was designed and implemented. The initial schematic was created using Xschem, as shown in Fig.5.1a. To verify the correctness of the circuit, a simulation was performed in NGspice. The resulting waveform can be seen in Fig.5.1c.

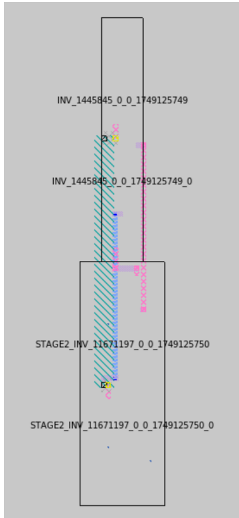
Next, a flattened netlist was generated from Xschem. However, this netlist cannot be used directly in ALIGN, as ALIGN requires a more abstracted or simplified netlist. Therefore, the netlist must first be manually simplified. After simplification, ALIGN is used to generate the layout, ensuring that it is DRC-clean. The resulting layout can be visualized using Magic, as shown in Fig. 5.1b.

Once the layout is complete, a Layout vs. Schematic (LVS) check is performed using NETGEN to verify that the layout matches the original schematic. Following LVS verification, parasitic extraction is carried out in Magic. The extracted parasitic information is then used for post-layout simulation in NGspice. The results of this simulation are shown in Fig. 5.1d.

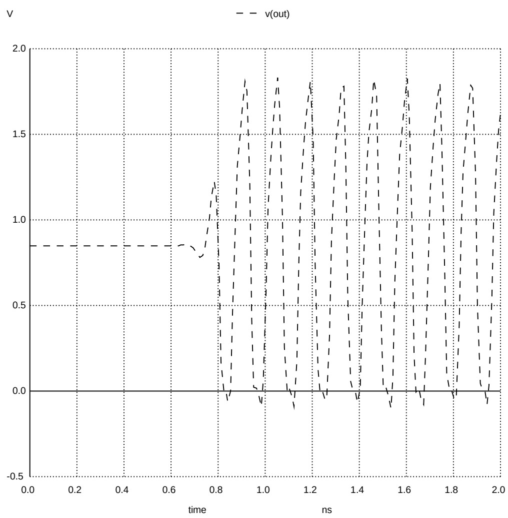
As illustrated by the close agreement between the pre-layout and post-layout simulation results, the design flow has been successfully executed, validating the correctness and reliability of the open-source analog design process.



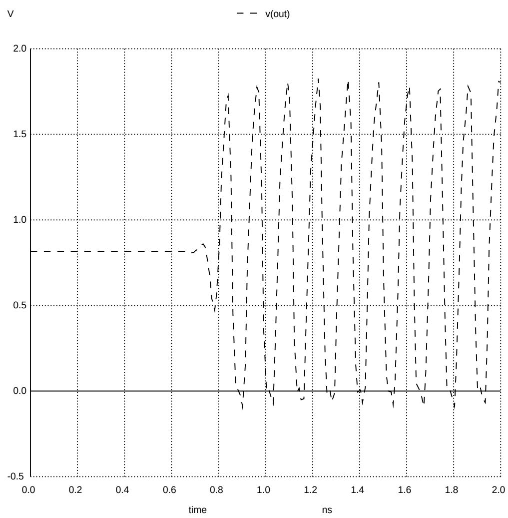
(a) Schematic in `Xschem`



(b) Layout in `Magic`



(c) Simulation before layout



(d) Post-layout simulation

Figure 5.1: Overview of the analog design flow using a ring oscillator: schematic and layout (top), and simulation results before and after layout (bottom).

6

Mixed-Signal Simulation

6.1. Methodology

Our mixed-signal co-simulator is built as an event-driven, modular approach. This design choice plays to the strengths of each specialized simulation tool. The simulator simulates digital blocks using a dedicated digital logic simulator (like Verilator or Icarus Verilog) and analog blocks with an analog circuit simulator (such as Ngspice). This ensures each part of the design gets the most accurate and efficient simulation possible, avoiding the trade-offs often found in single, all encompassing mixed-signal environments.

6.1.1. Mixed-signal components

One of the biggest hurdles in mixed-signal simulation is properly modeling mixed-signal components that sit at the boundary between the digital and analog worlds. Traditionally, commercial tools use Verilog-AMS for this. It's a standard language that lets you describe both analog and digital behaviors within a single model. However, finding open-source Verilog-AMS compilers and simulators that are powerful and fully featured is quite difficult. This limitation meant that this standard approach couldn't be used, so a different approach was needed to handle these mixed-signal components.

Our solution was to model these mixed-signal components using Python. This choice gives us incredible flexibility within an open-source framework:

- **Quick Development:** Python lets us build and change conversion algorithms very quickly without having to redesign the complete circuit.
- **Tailored Behavior:** We can easily implement specific, even non-standard, behaviors or highly detailed models without being constrained by a hardware description language.
- **The "Glue" Logic:** Python acts as the central "glue" that processes and translates signals between the discrete digital domain and the continuous analog domain.

The advantage about this Python modeling approach is that it gives the user fine-grained control over how abstract (or detailed) they want these mixed-signal components to be simulated:

- **High-Fidelity Simulation (Timing is Key):** If you need super precise timing and analog characteristics (like an Analog-to-Digital Converter's (ADC) settling time or a Digital-to-Analog Converter's (DAC) output impedance), you can create a detailed analog schematic of the mixed-signal component and simulate it directly in Ngspice. In this case, the Python code is mostly just a wrapper. It takes the analog voltage data, applies a comparator-like function or an ADC model to determine the digital state (0 or 1), or it takes digital bits and converts them into an analog voltage (like generating a piecewise linear (PWL) voltage source for Ngspice).
- **Behavioral/Idealized Simulation (Speed is Key):** When timing details aren't critical for overall system testing, you can model the mixed-signal component as an ideal, simplified component directly in Python. For instance, an ideal ADC might instantly convert a voltage to a digital code,

or an ideal DAC might produce a perfect stair-step voltage from a digital input without any delays or non-idealities. This drastically cuts down simulation time, speeding up design iterations.

Simulation preparation

The co-simulator begins with individual Verilog files for digital blocks and SPICE source files for analog blocks. Additionally, there's a top-level Xschem schematic where each block is represented by a symbol with input and output pins. A separate parameter file specifies global input signals and overall simulation time settings. The simulator starts by parsing the top-level netlist generated by Xschem. This netlist is broken down, and all the details about each individual block are stored in a Python block dictionary. This dictionary is like a detailed dossier for every component, containing:

- **Instance Name:** The unique ID for the block (e.g., X1 or X2).
- **Block Type:** Whether it's 'analog', 'digital', or 'mixed-signal', determining which simulator handles it.
- **Source File:** The actual .spice, .v, or .py file that defines the block's behavior.
- **Input/Output Signals:** A list of all pins connecting in and out of the block.
- **Nets:** A list of top-level wires (nets) connected to the block.
- **Pin-to-Net Mapping:** A precise map showing which internal pin connects to which top-level net.

At the same time, a separate net dictionary is built to manage the web of connections (signals) between blocks. It tracks signal flow and enables efficient data transfer between simulation steps. For each top-level net, it tracks:

- **Driver Block:** The block sending the signal onto this net.
- **Driver Signal:** The specific internal pin in the driver block connected to this net.
- **Receiver Blocks:** All blocks listening to this net.
- **Receiver Signals:** The specific internal pins in those receiver blocks connected to this net.
- **Current Value(s):** The latest simulated value(s) of the signal on the net crucial for setting initial conditions for the next simulation.

Using both a block-centric and a net-centric dictionary was a conscious design choice. The block dictionary gives an "inside-out" view, showing a component's internals helpful when generating its subcircuit. The net dictionary gives an "outside-in" view, focusing on signal connectivity across the system essential for managing data between simulation steps.

After parsing, the simulation files are prepared. Analog blocks need continuous input signals. A decision was made: all analog blocks that are interconnected (where one's output feeds another's input) are combined into a single Ngspice .spice file. This preserves analog integration and continuous behavior over time. Digital blocks get separate Verilog testbenches, as their event-driven nature allows independent simulation. The generated simulation files include placeholders for parameters like step size, total simulation time, and initial conditions. This blueprint-style setup allows file reuse across many simulation iterations. Before each simulation run, the parameter file is read, and all placeholders are populated with the correct values for the current iteration.

6.1.2. Simulation Iteration and the Event Manager

Each simulation iteration begins with both the digital and analog simulations running concurrently:

- **Analog Simulation:** The combined Ngspice .spice file for analog blocks is executed. Output data for each net (e.g., V(net1), V(net2)) is saved to a plain text file using the `wrdata` command. This produces a clean, tabular, space-separated text file that's easy for Python to parse using standard libraries like `numpy` or `pandas`.
- **Digital Simulation:** The separate Verilog testbenches for digital blocks are run (using tools like Icarus Verilog). These testbenches use Verilog's `$fdisplay` or `$fwrite` commands to output digital signal data (0, 1, X, Z states) to a single text file. This file is easily parsed by Python and split into separate structures for each output signal. Again, simplicity and Python integration are prioritized over traditional waveform formats like VCD.

Once these simulations finish and output data is available, the waveforms are fed into the Python models of the mixed-signal components. This is where domain bridging happens. For digital-to-analog, signals are stored as strings for Ngspice's PWL function, allowing new voltage values to be inserted. For analog-to-digital, value changes are stored as pairs of timestamps and new values.

After each iteration, once mixed-signal components are updated, the event manager analyzes all signals(analog, digital, and mixed). In this co-simulation framework, event detection is applied exclusively to signals originating from digital and mixed-signal blocks. This is because they produce discrete events (e.g., logic-level changes), making them suitable for event-driven processing. Analog blocks, which generate continuous-time signals, do not produce discrete events. Since analog blocks are simulated as cascaded subsystems that eventually connect to mixed-signal interfaces (e.g., comparators or DACs), transitions are captured at the boundary. This avoids unnecessary checks within analog paths, simplifying simulation and preserving synchronization across domains.

Finally, the simulation files (both Ngspice and Verilog) are dynamically updated with the new net values. This iterative process, driven by the event manager, continues until the full simulation duration is complete or no further events are detected. This adaptive, event-driven method focuses resources on active parts of the circuit, stitching together many short simulations into one complete waveform for analysis and plotting. The final code of the simulator is found in this github repository: https://github.com/Rubenwosten/Project_OPEN_SOURCE_LUNAR

6.2. Testing with a Mixed-Signal Circuit

To test the simulator's functionality, a mixed-signal circuit was designed. The circuit is shown in Figure 6.1.

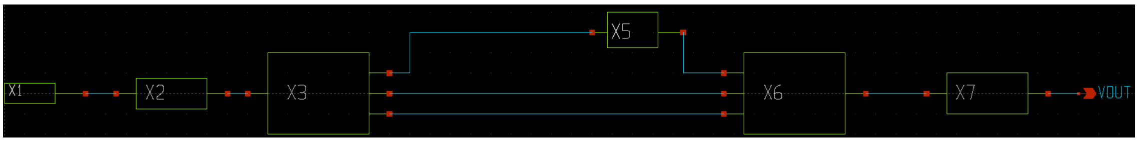


Figure 6.1: Top-level design of the mixed-signal IC with: X1 = V source, X2 = low-pass filter, X3 = ADC, X5 = inverter, X6 = DAC, X7 = voltage divider

The circuit begins with X1, a sine-wave voltage source. This passes through a low-pass filter (X2), which smooths the waveform. These blocks shows simultaneous simulation of two analog blocks. Because the analog signal is directly connected to X2, no signal integrity is lost.

The filtered signal reaches X3, an ADC that converts the waveform into a 3-bit digital signal. The LSB from the ADC passes through an inverter (X5), demonstrating digital block interaction (X3 to X5).

Meanwhile, the MSB and another bit from the ADC are routed directly to the DAC (X6). This demonstrates interaction between two mixed-signal components. Python is required to specify the signal type at output. The X5 output also acts as the LSB input for the DAC. The DAC converts the digital signal into a quantized, inverted, and delayed sine wave.

Finally, the analog output goes through a voltage divider (X7), producing Vout, half the DAC's output.

6.3. Simulation results

The circuit was simulated for 50 ns, using both 1 ns and 50 ps step times. Figure 6.2 and Figure 6.3 show the voltage divider output for 1 ns and 50 ps steps, respectively. Additional net signal plots are in Appendix B for both step sizes. The results show that a smaller step size provides higher accuracy. Both simulations were completed in 15 minutes.

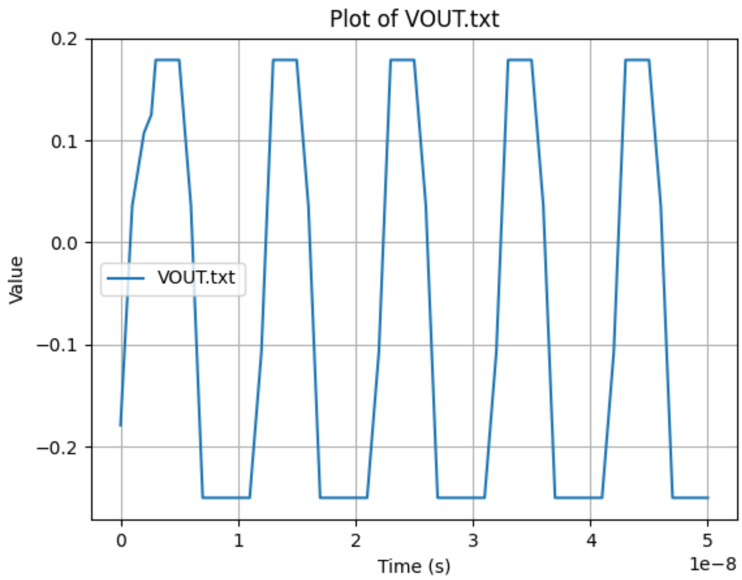


Figure 6.2: Vout after the voltage divider with a step of 1 ns.

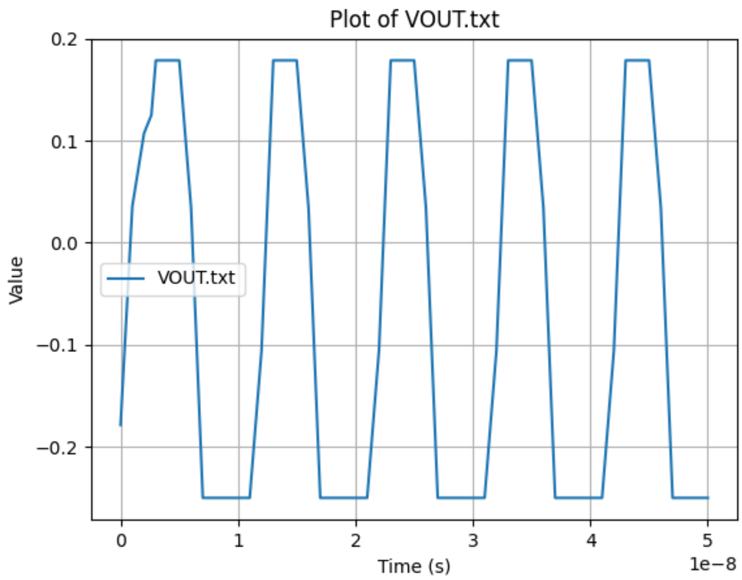


Figure 6.3: Vout after the voltage divider with a step of 50 ps.

7

Cryogenic PDK

To simulate analog circuits in extreme environments like the Moon, where temperatures can drop to around 20 K, standard room-temperature CMOS models are not accurate enough [24]. Cryogenic simulation is required to ensure that circuits still function reliably under these conditions. The goal of this part of the research was to prove that cryogenic simulation is possible using open-source tools.

As demonstrated by Akturk et al. [25], CMOS circuits fabricated using the Skywater 130nm process can operate at cryogenic temperatures (4K). However, full SPICE-compatible transistor models for cryogenic simulation have not been released. Instead, Akturk et al. provided a set of simulation primitives. These are Verilog-A behavioral models for NMOS and PMOS transistors, based on cryogenic measurements. These included measured gate voltage (V_G) sweep curves for the NFET 01V8 device under various combinations of source (V_S), drain (V_D), and body (V_B) voltages[26]. These curves are the basis of this work.

In this work, we manually extracted key electrical parameters from these primitives using a custom Python script (detailed in Appendix A.2) for a specific transistor geometry: width = 0.42µm and length = 0.15µm. The extracted parameters include threshold voltage (V_{th}), subthreshold slope, carrier mobility, and leakage current (I_{off}). Six different bias configurations were analyzed, each with distinct values of V_S, V_D, and V_B. The corresponding transfer characteristics are shown in Figure 7.1, and the extracted parameters are summarized in Table 7.1.

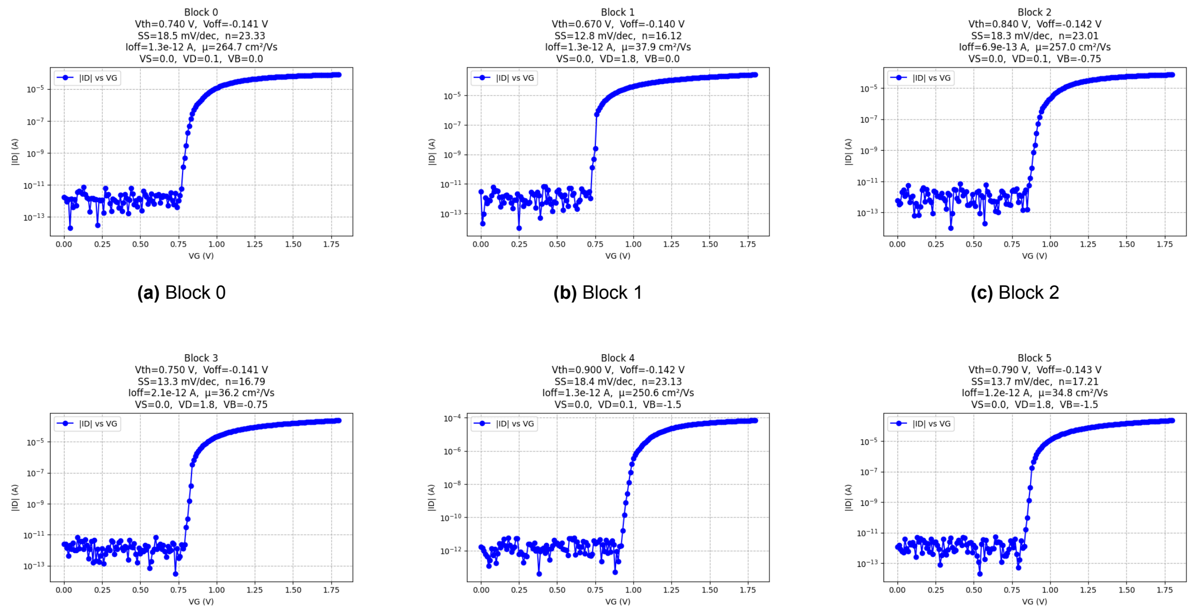
Compared to the standard 300K Skywater PDK models, the extracted 4K data show significant changes in key characteristics, particularly shifts in V_{th}, the nfactor and mobility. These values were then used to modify the NFET 01V8 model within the Skywater PDK to better reflect cryogenic behavior for the given device size.

Table 7.1: NFET 01V8 parameters at 4.0 K for different drain, source, and base voltages

Block	V_{th} (V)	V_{off} (V)	Slope (mV/dec)	n	I_{off} (A)	Mobility (cm^2/Vs)	V_S (V)	V_D (V)	V_B (V)
0	0.740	–0.141	18.52	23.33	1.33×10^{-12}	264.7	0.0	0.1	0.0
1	0.670	–0.140	12.80	16.12	1.30×10^{-12}	37.9	0.0	1.8	0.0
2	0.840	–0.142	18.27	23.01	6.85×10^{-13}	257.0	0.0	0.1	–0.75
3	0.750	–0.141	13.33	16.79	2.05×10^{-12}	36.2	0.0	1.8	–0.75
4	0.900	–0.142	18.37	23.13	1.34×10^{-12}	250.6	0.0	0.1	–1.5
5	0.790	–0.143	13.67	17.21	1.16×10^{-12}	34.8	0.0	1.8	–1.5

7.1. Results

To evaluate the impact of temperature on device behavior, we simulated the NFET in Xschem using NGspice with both the 4 K cryogenic parameters and the standard room temperature parameters. As Bohuslavskyi et al. demonstrate [[27]], the BSIM4 model becomes unreliable below roughly 77 K. Consequently, our NGspice simulations were performed at 77 K instead of 4 K to avoid the inaccuracies



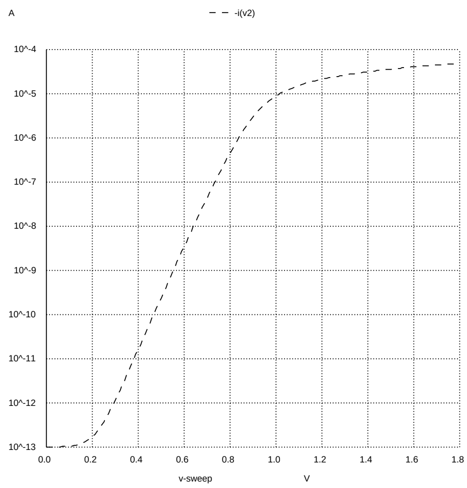
(a) Block 0 **(b)** Block 1 **(c)** Block 2

(d) Block 3 **(e)** Block 4 **(f)** Block 5

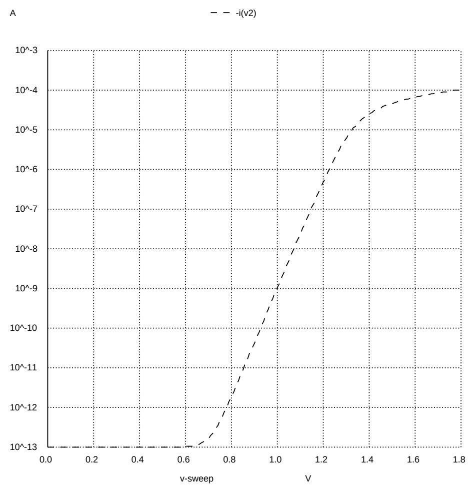
Figure 7.1: Transfer Characteristics for All Blocks

associated with the model's breakdown at deeper cryogenic temperatures. The results of these simulations are shown in Figure 7.2, where the transfer characteristics for both conditions are compared. The gate voltage (V_G) was swept from 0 to 1.8V with $V_S = 0$, $V_S = 0.1$, and $V_B = 0$ for both cases. As seen in the comparison, the NFET behaves very differently at 4K compared to room temperature. At cryogenic temperatures, the threshold voltage (V_{th}) shifts, the subthreshold slope becomes steeper, and the mobility decreases, which all contribute to a marked change in the transistor's behavior. These differences highlight the significant impact that temperature has on device performance and reinforce the need for cryogenic-specific modeling when designing for low-temperature environments. Although the 4 K curve resembles the initial dataset seen in 7.1 as block 0, the slope in the rising region is noticeably different. This discrepancy likely arises because the simulation was run at 77 K rather than 4 K and may also reflect that not all of the NFET's characterization parameters were fully incorporated.

The goal of this experiment was to prove that a full cryogenic PDK is possible by simulating a single device under cryogenic conditions. Akturk et al.[25] are aiming to implement their cryogenic models for CMOS devices into the open-source Skywater PDK. Once this integration is complete, these models can also be incorporated into our own EDA environment, enabling cryogenic simulation capabilities. This will make it easier to design and validate analog circuits for deep-space applications, such as lunar missions.



(a) Room Temperature



(b) 4K Temperature

Figure 7.2: Comparison of NFET with $V_\text{S} = 0$, $V_\text{D} = 0.1$, and $V_\text{B} = 0$ at Room Temperature and 4K, where V_G is swept from 0 to 1.8 V.

8
Discussion

8.1. Cryogenic PDK

This work only scratches the surface of cryogenic simulation by focusing on a single device at low temperatures. Only a few of the characteristics of this device were found so true cryogenic integration was not achieved, but the results do show that this integration is possible. Ongoing research aims to extend this capability by integrating cryogenic compact models for CMOS devices into the Skywater PDK [25]. Once implemented, this will enable cryogenic simulation support within open-source EDA tools. This will make it easier to design and validate analog circuits for deep-space applications, such as lunar missions, and also aligns with broader research efforts in cryogenic CMOS technology for quantum computing and space electronics [28].

8.2. Docker

Docker enabled a portable and reproducible design environment, simplifying tool setup across platforms. However, performance issues can arise when running multiple containers in parallel, especially on systems with limited resources. As shown by Đorđević et al. [29], resource contention increases with the number of active containers, potentially leading to slowdowns. This was occasionally observed during intensive tasks in our setup. Applying resource limits and staggered startup can help mitigate these effects while retaining the benefits of containerization.

8.3. Automatic IC design flow

: In this thesis, an automatic analog IC design flow was developed by integrating several open-source tools. Each tool plays a specific role in the design process: Xschem for schematic capture, Ngspice for simulation, Magic for layout editing, ALIGN for analog layout generation, and Netgen for layout-versus-schematic (LVS) verification. Python scripts coordinate the flow between these tools, automating much of the design, simulation, and verification process.

Despite these advancements, the current flow stops short of being a fully automatic IC design system. A critical missing component is an intelligent placement and routing engine that can take individual block layouts and arrange them into a complete chip-level layout. While ALIGN can automate the layout of analog blocks and OpenLane can do the same for digital blocks, placing and routing those individual blocks while managing spacing and routing constraints still requires significant manual effort. Furthermore, the transition from schematic capture in tools like xschem to layout generation with ALIGN is not fully automated, as it typically requires manual modification of the netlist to make it compatible with ALIGN's input format. It's also important to note that ALIGN itself is still under active development, with ongoing efforts to improve its capabilities and extend automation.

To close this gap, the integration of machine learning or AI-based algorithms is proposed for future research[30]. Such a system would learn optimal layout strategies from existing designs and generate placement and routing solutions that respect design rules, minimize area, and optimize signal quality.

Once a complete layout is assembled, final checks such as Design Rule Check (DRC) and LVS can be performed using Magic and Netgen, respectively. The final simulation of the complete transistor-level layout can then be conducted using Ngspice to validate the entire design.

8.4. Mixed-signal simulator

This mixed-signal co-simulator, while open-source, shares some fundamental ideas with powerful commercial tools like Cadence Spectre-AMS. However, it also has some key differences that lead to its main limitations and define areas for future research.

Common ideas:

- Co-Simulation Core: Both our simulator and Spectre-AMS share the basic idea of ”co-simulation.” This means they're both designed to simulate digital parts with one engine and analog parts with another, then link them up. It's all about using the best tool for each job.
- Event-Driven Approach: Just like our simulator, Spectre-AMS is also event-driven. It focuses on when significant things happen (like signal changes or clock edges) rather than just plodding through every tiny time step. This helps manage complex simulations efficiently.
- Domain Interfacing: Both systems need a way to pass information between the digital and analog worlds. They both translate digital logic states into analog voltages and vice-versa at the boundaries between mixed-signal components.

Key Differences:

- Mixed-Signal Component Modeling: Spectre uses Verilog-AMS, We use python for ideal behevior or spice+python to model the analog schematic with the conversion done by python.
- Integration Level: Offers a much tighter, often native, integration between its analog and digital solvers. The entire simulation often feels like one seamless process. Our Simulator: Achieves integration through file-based communication and Python glue code. This is a looser coupling.
- Spectre-AMS: Being a highly optimized commercial tool, it boasts advanced algorithms for convergence, error handling, and speed. It can handle very large and complex mixed-signal designs with high efficiency. Our Simulator: Relies on simpler, sequential execution of separate open-source tools. While the individual simulators (Ngspice, Verilator) are optimized, the overall ”stitching” process in Python adds overhead. This impacts performance of very large or complex designs.

Based on these differences, the primary limitations of our proposed simulator become clear:

- Overhead of File-Based Communication: The continuous writing and reading of text files, along with Python parsing and data manipulation, introduces significant overhead. This can slow down simulations.
- Less Robust Convergence: Commercial simulators have sophisticated algorithms to ensure convergence in tricky analog circuits and handle rapid digital-analog interactions. Our simpler, iterative approach might be more prone to convergence issues or require careful tuning of simulation segment lengths and initial conditions.
- No Native Verilog-AMS Support: The reliance on Python models for mixed-signal components, while flexible, means we can't directly leverage existing Verilog-AMS models or the benefits of its integrated language. This adds a translation layer and might limit the direct reusability of models from other platforms.
- Manual Integration and Debugging: The ”glue code” approach requires more manual effort in tracing a problem through multiple simulation outputs and Python scripts can be more challenging

Looking ahead, there are several exciting directions for future research and development to enhance this co-simulator:

- Improved Communication Interface: Explore more efficient inter-process communication (IPC) methods (e.g., pipes, sockets, shared memory) between Ngspice, the Verilog simulator, and

Python, to reduce the overhead of file I/O. This would make the ”stitching” process much faster and smoother.

- Advanced Event Management: Develop more sophisticated event detection algorithms in the event manager. This could include predictive event detection, more complex thresholding logic, and a hierarchical event processing system to better manage large designs.
- Graphical User Interface (GUI): Develop a basic GUI to simplify setup, visualization of simulation results (perhaps integrating with an open-source waveform viewer), and debugging. This would make the simulator much more user-friendly.
- Distributed Simulation: For very large designs, investigate the possibility of running analog and digital simulations on different processor cores, leveraging parallel computing to speed up the overall simulation time.
- Integration of Machine Learning/AI: Explore how Machine Learning (ML) or Artificial Intelligence (AI) techniques could be integrated. This could involve using ML for predictive timestepping in analog simulations, optimizing initial conditions for convergence, or even creating behavioral models of complex analog blocks from observed data to accelerate simulation runtime.
- Exploring Alternative Simulators and Languages: Investigate the feasibility of integrating other open-source analog (e.g., Xyce) or digital (e.g., GHDL for VHDL) simulators. Additionally, explore using different hardware description languages (like VHDL) and their associated open-source tools to broaden the simulator’s applicability and flexibility.

This project, while currently a robust functional demonstration, has potential for growth and further development within the open-source EDA space.

9

Conclusion

The implementation status of the project's initial requirements is summarized in Table 9.1. These requirements were defined at the outset of the project as functional and non-functional goals for the toolchain. Most of them were successfully implemented, confirming the technical feasibility and effectiveness of the open-source mixed-signal design flow. However, three requirements were only partially fulfilled:

- **[A.7] Integrated analog and digital placement and routing:** While the flow supports placement and routing of analog and digital blocks individually, full automation of their integration in a unified layout proved to be beyond the scope of this project. Currently, mixed-signal layout integration must be performed manually using Magic.
- **[B.4] Minimizing manual intervention:** The digital flow is fully automated through OpenLane. For the analog side, most steps can be automated; however, some manual effort is required to adapt Xschem-generated netlists, as ALIGN expects simplified netlist formats. Despite this limitation, the workflow achieves a high degree of automation.
- **[B.7] Cryogenic PDK adaptation:** Although full support for cryogenic operation in the SkyWater130 PDK is not yet available, initial work has been completed. Due to the limited availability of open-source cryogenic models, only one device model was implemented to demonstrate the viability of cryogenic integration within the toolchain.

This thesis presented the development of an open-source EDA toolchain for mixed-signal IC design, combining digital and analog workflows while also enabling preliminary support for cryogenic simulation. By integrating tools such as OpenLane, Magic, Xschem, Ngspice, and ALIGN, a modular and partially automated design flow was demonstrated. This approach significantly lowers the barrier to entry for researchers, educators, and students interested in IC design.

In addition, the thesis showed that cryogenic transistor behavior can be extracted and modeled using Python-based methods. This lays a foundation for future integration of cryogenic device models into the SkyWater PDK.

The tools and methodologies developed in this thesis will also contribute to chip design for space exploration, particularly for lunar missions. The extreme conditions on the Moon, including low temperatures and high radiation exposure, require specialized electronics. The ability to simulate and model cryogenic behavior in transistors and integrate these models into open-source EDA tools is a critical step toward developing robust, reliable chips for use in lunar environments. This could directly support the creation of the next generation of electronics needed for long-term human presence on the Moon, ensuring that systems can operate under the harsh conditions of space.

In general, the results confirm that open source tools can serve as a viable and practical alternative to commercial EDA's, especially in research and academic environments where transparency, accessibility, and customization are crucial.

ID	Requirement Description	Implemented?
A.1	The system must support schematic-based analog design and simulate it using Ngspice.	Yes
A.2	The system must support digital RTL-to-GDSII flow using OpenLane and allow digital simulation using Iverilog.	Yes
A.3	The system must support co-simulation of analog and digital blocks for functional verification.	Yes
A.4	The toolchain must allow for layout-versus-schematic (LVS) and design rule checking (DRC).	Yes
A.5	The toolchain must support netlist extraction (PEX) and post-layout simulation of analog blocks.	Yes
A.6	The entire toolchain must be containerized using Docker, enabling consistent execution across different operating systems (Linux, Windows, macOS) with minimal setup.	Yes
A.7	The toolchain must support integrated placement and routing of analog and digital blocks, enabling mixed-signal physical design.	Partly
B.1	The toolchain must work entirely with open-source software.	Yes
B.4	The workflow should minimize manual intervention, particularly in layout generation and simulation.	Partly
B.5	The toolchain must be compatible with the Sky130 process design kit (PDK).	Yes
B.6	The system must include a Dockerfile that encapsulates all dependencies and enables consistent deployment across systems.	Yes
B.7	The SkyWater130 PDK must be changed to support cryogenic operation.	Partly

Table 9.1: Programme of Requirements (PoR) Implementation Status

References

[1] NASA, *Artemis: Humans in space*, Accessed: 2025-06-16, 2025. [Online]. Available: https://www.nasa.gov/humans-in-space/artemis/.

[2] Author(s), "Radiation exposure on the moon: The moon lacks a global magnetic field and dense atmosphere, leaving its surface exposed to galactic cosmic rays and solar energetic particles," *PubMed*, Year, Accessed: 2025-06-16. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/.

[3] NASA, *Shackleton crater's illuminated rim and shadowed interior*, Accessed: [Insert Date], 2022. [Online]. Available: https://science.nasa.gov/resource/shackleton-craters-illuminated-rim-shadowed-interior/.

[4] H. Pretl, "Designing Analog/RF Chips Using Open PDKs and Open-Source Tools," Tech. Rep.

[5] J.-P. Kleinhans, "The eda chokepoint dilemma? openness, oligopolies, and china's ecosystem," Institute on Global Conflict and Cooperation, University of California, San Diego, Working Paper qt16d3b8z5, Dec. 2022, RePEc handle: RePEc:cdl:globco:qt16d3b8z5. [Online]. Available: https://escholarship.org/uc/item/16d3b8z5.pdf.

[6] C. D. Systems, *Cadence design tools*, Accessed: 2025-06-16, 2025. [Online]. Available: https://www.cadence.com.

[7] S. EDA, *Siemens eda tools*, Accessed: 2025-06-16, 2025. [Online]. Available: https://eda.sw.siemens.com.

[8] Reuters, "Synopsys halts china sales due to u.s. export restrictions, internal memo shows," *Reuters*, 2025, Accessed: 2025-06-10. [Online]. Available: https://www.reuters.com/world/china/synopsys-halts-china-sales-due-us-export-restrictions-internal-memo-shows-2025-05-30/?utm_source=chatgpt.com.

[9] C. Lück, D. Sánchez Lopera, S. Wenzek, and W. Ecker, "Industrial experience with open-source eda tools," in *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD '22)*, Accessed: 2025-06-10, 2022, p. 143. DOI: 10.1145/3551901.3557040. [Online]. Available: https://doi.org/10.1145/3551901.3557040.

[10] E. Corporation, *Openlane: Open-source rtl to gdsii flow for digital asic design*, https://github.com/The-OpenROAD-Project/OpenLane, Accessed: 2025-06-12, 2025.

[11] T. Edwards, *Magic vlsi layout tool*, http://opencircuitdesign.com/magic/, Accessed: 2025-06-12, 2025.

[12] S. Schippers, *Xschem: Schematic capture tool for analog design*, https://github.com/StefanSchippers/xschem, Accessed: 2025-06-12, 2025.

[13] H. Vogt *et al.*, *Ngspice: Open source spice simulator*, http://ngspice.sourceforge.net/, Accessed: 2025-06-12, 2025.

[14] T. Edwards, *Netgen: Lvs tool for layout vs schematic verification*, http://opencircuitdesign.com/netgen/, Accessed: 2025-06-12, 2025.

[15] K. Hofmann and J. Anders, *Analog 2020*. VDE Verlag, 2020, ISBN: 9783800753352.

[16] M. P. Systems, *Integration challenges in mixed-signal environments*, https://www.monolithicpower.com/en/learning/mpscholar/analog-vs-digital-control/practical-design-considerations/integration-challenges-in-mixed-signal-environments?srsltid=AfmBOooT3erQ7ZNlm61fLQ7VabSiJbSJ7K65nnrIE-qBYppNOXy5_yEH&utm_source=chatgpt.com, Accessed: 2025-06-10, 2025.

[17] Y. Liu, X. Zhang, F. Yang, and Z. Wang, "Iicpilot: An intelligent integrated circuit backend design framework using open eda," *arXiv preprint arXiv:2308.01857*, 2023, Accessed: 2025-06-10. [Online]. Available: https://arxiv.org/pdf/2308.01857.pdf.

[18] C. D. Systems, *Cadence spectre ams designer*, https://www.cadence.com, Accessed: 2025-06-16.

[19] S. Inc., *Vcs ams simulator*, https://www.synopsys.com, Accessed: 2025-06-16.

[20] F. Foundation, *Challenges in open-source eda*, https://fossi-foundation.org, Accessed: 2025-06-16.

[21] M. Sherwood and R. Michael, *Full-stack reproducibility for ai/ml with docker and kaskada*, Accessed: 2025-06-12, 2023. [Online]. Available: https://www.docker.com/blog/full-stack-reproducibility-for-ai-ml-with-docker-kaskada/.

[22] R. J. Bithray and D. Stavrov, “Designing hardware for a lunar wireless sensor network,” Delft University of Technology, Tech. Rep., Jun. 2025.

[23] R. J. Bihray and D. Stavrov, *Wirelesssensornodes*, Jun. 2025. [Online]. Available: https://github.com/rbithray/WirelessSensorNodes.

[24] B. Patra, R. M. Incandela, J. P. G. van Dijk, *et al.*, “Cryo-cmos circuits and systems for quantum computing applications,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 1, pp. 309–321, 2018. DOI: 10.1109/JSSC.2017.2737549.

[25] A. Akturk, A. Tripathi, and M. Saligane, “Cryogenic Modeling for Open-Source Process Design Kit Technology,” in *2023 IEEE BiCMOS and Compound Semiconductor Integrated Circuits and Technology Symposium, BCICTS 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 58–65, ISBN: 9798350307641. DOI: 10.1109/BCICTS54660.2023.10310944.

[26] A. Akturk, A. Li, M. Saligane, *et al.*, *Cmos sky130 primitives measured at cryogenic temperatures*, Accessed: 2025-06-13, 2023. DOI: 10.18434/mds2-2997. [Online]. Available: https://doi.org/10.18434/mds2-2997.

[27] H. Bohuslavskyi, A. G. M. Jansen, S. Barraud, *et al.*, “Cryogenic subthreshold swing saturation in fd-soi mosfets described with band broadening,” *IEEE Electron Device Letters*, vol. 40, no. 5, pp. 784–787, May 2019. DOI: 10.1109/LED.2019.2903111. arXiv: 1903.05409 [cond-mat.mes-hall].

[28] E. Charbon, F. Sebastiano, A. Vladimirescu, *et al.*, “Cryo-cmos for quantum computing,” in *Proceedings of the 62nd IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 343–346. DOI: 10.1109/IEDM.2016.7838410.

[29] E. Gamess and M. Parajuli, “Image-processing workloads and ddos attack resilience: Evaluating docker and podman containers on raspberry pi and odroid,” in *Proceedings of the 2024 ACM Southeast Conference (ACMSE ’24)*, 2024, pp. 138–147. DOI: 10.1145/3603287.3651219. [Online]. Available: https://doi.org/10.1145/3603287.3651219.

[30] A. Caviglia, G. Masera, and M. Martina, “Towards machine learning for placement and routing in chip design: A methodological overview,” *arXiv preprint arXiv:2202.00329*, 2022. [Online]. Available: https://arxiv.org/abs/2202.00329.

A

Appendix A

A.1. Docker images

A.1.1. Docker compose configuration

```
version: "3.8"

services:
  align:
    image: ohdejong/my-align-with-pdk:latest
    container_name: align
    volumes:
      - ./shared:/workspace
      - ./shared/pdks:/opt/pdk
    working_dir: /workspace
    tty: true

  analog:
    image: ohdejong/analog-docker:latest
    container_name: analog
    volumes:
      - ./shared:/workspace
      - ./shared/pdks:/opt/pdk
    environment:
      - DISPLAY=host.docker.internal:0.0
    working_dir: /workspace
    tty: true

  openlane:
    image: ohdejong/my-openlane-with-pdk:latest
    container_name: openlane
    volumes:
      - ./shared:/workspace
      - ./shared/pdks:/opt/pdk
    working_dir: /workspace
    tty: true

  icarus:
    image: nchandra75/iverilog:latest
    container_name: icarus
    volumes:
      - ./shared:/workspace
    working_dir: /workspace
    tty: true
```

Listing A.1: Docker Compose Configuration

A.1.2. Custom Analog Dockerfile

```
FROM ubuntu:22.04

ENV DEBIAN_FRONTEND=noninteractive

# ------------------------------
# Install system dependencies
# ------------------------------
RUN apt-get update && apt-get install -y \
    build-essential \
    gcc \
    g++ \
    python3 \
    python3-dev \
    python3-venv \
    python3-pip \
    cmake \
    ninja-build \
    libboost-all-dev \
    libx11-dev \
    libxext-dev \
    libxpm-dev \
    libxmu-dev \
    libmotif-dev \
    libfftw3-dev \
    bison \
    flex \
    pkg-config \
    libgtk2.0-dev \
    netgen \
    tcl-dev \
    tk-dev \
    libcairo2-dev \
    libxcb1-dev \
    libxrender-dev \
    libboost-python-dev \
    autoconf \
    automake \
    libtool \
    git \
    curl \
    wget \
    xterm \
    libxaw7-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

# ------------------------------
# Build and install ngspice with GUI support
# ------------------------------
RUN git clone https://github.com/ngspice/ngspice.git /opt/ngspice && \
    cd /opt/ngspice && \
    ./autogen.sh && \
    mkdir release && cd release && \
    ../configure --with-x --enable-xspice --disable-debug && \
    make -j$(nproc) && \
    make install

```

```
# ------------------------------
# Build and install Magic from source
# ------------------------------
RUN git clone https://github.com/RTimothyEdwards/magic.git /opt/magic && \
    cd /opt/magic && \
    ./configure && \
    make -j$(nproc) && \
    make install

# ------------------------------
# Build and install Xschem from source
# ------------------------------
RUN git clone https://github.com/StefanSchippers/xschem.git /opt/xschem && \
    cd /opt/xschem && \
    ./configure && \
    make -j$(nproc) && \
    make install

# ------------------------------
# Set environment variables
# ------------------------------
ENV PATH="/opt/xschem/bin:${PATH}"

# ------------------------------
# Set working directory
# ------------------------------
WORKDIR /work

CMD ["/bin/bash"]
```

Listing A.2: Dockerfile for Analog Container with Ngspice, Magic, and Xschem

A.1.3. Custom Openlane Dockerfile

```
# Use OpenLane's base Docker image
FROM efabless/openlane:latest

# Set the working directory to OpenLane
WORKDIR /openlane

# Set environment variable for PDK_ROOT
ENV PDK_ROOT=/openlane/pdks

# Set the PDK variant as Sky130A
ENV PDK=sky130A

# Install Python and Pip (Python is needed for OpenLane dependencies)
RUN python3 -m ensurepip --upgrade && \
    pip3 install --no-cache-dir -U pip setuptools

# Install Volare (for handling PDK builds)
RUN pip3 install volare

# Enable the specific Sky130 PDK version using its commit hash
RUN volare enable --pdk sky130 bdc9412b3e468c102d01b7cf6337be06ec6e9c9a

# Configure OpenLane to use Sky130A PDK
RUN ln -sf $PDK_ROOT/sky130/libs.tech/sky130_fd_sc_hd/openlane/config.tcl /
    openlane/config.pdk

# Expose OpenLane's working directory for design files
VOLUME ["/openlane/designs"]
```

```

# Set the default command to bash to interact with the container
CMD ["bash"]
```

Listing A.3: Dockerfile based on the OpenLane base image with Sky130A PDK and Volare

A.1.4. Custom ALIGN Dockerfile

```
# Start from the official Align Docker image
FROM darpaalign/align-public:latest AS env

# Set build arguments for UID and GID
ARG UID=0
ARG GID=0

# Create a group and user based on the UID and GID arguments
RUN if [ "$GID" -ne "0" ] ; then echo $GID && groupadd -g $GID -o align; else
    echo 1000 && groupadd -g 1000 -o align; fi
RUN if [ "$UID" -ne "0" ] ; then useradd -m -u $UID -g $GID -p align -o -s /bin/
    bash align; else useradd -m -u 1000 -g 1000 -p align -o -s /bin/bash align; fi

# Install wget (and any other dependencies like git or curl)
USER root
RUN apt-get update && apt-get install -y \
    git \
    make \
    && rm -rf /var/lib/apt/lists/*

# Change ownership of /work directory to the user
RUN chown -R align /work

# Set the working directory
USER align
WORKDIR /work

# Set the PDK root directory environment variable to /home/align/pdk
ENV PDK_ROOT=/home/align/pdk

# Create the /home/align/pdk directory and clone the ALIGN-pdk-sky130 repository
RUN mkdir -p $PDK_ROOT && \
    git clone https://github.com/ALIGN-analoglayout/ALIGN-pdk-sky130.git $PDK_ROOT

# Switch back to the align user
USER align
```

Listing A.4: Dockerfile for ALIGN container based on darpaalign image with user setup and PDK installation

A.2. NFET 01v8 Characterization Script

```
import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.ndimage import gaussian_filter1d

# MDM parser

def parse_mdm(file_path):
    with open(file_path, "r") as f:
        lines = f.readlines()

    db_blocks = []
```

```python
    current_vars, data = {}, []
    in_db = False

    for line in lines:
        line = line.strip()

        if line.startswith("BEGIN_DB"):
            in_db = True
            current_vars, data = {}, []
            continue

        if line.startswith("END_DB"):
            if data:
                headers = [h.upper() for h in data[0]]
                df = pd.DataFrame(data[1:], columns=headers)
                for k, v in current_vars.items():
                    df[k.upper()] = float(v)
                db_blocks.append(df)
            in_db = False
            continue

        if not in_db:
            continue

        if line.startswith("ICCAP_VAR"):
            _, var, val = line.split()
            current_vars[var.upper()] = val
        elif line.startswith("#"):
            headers = line.replace("#", "").split()
            data.append(headers)
        elif line:
            values = line.split()
            data.append(values)

    return db_blocks

# Parameter extraction

def extract_parameters(df, temperature=4.0):
    df = df.copy()
    df[["VG", "ID"]] = df[["VG", "ID"]].apply(
        pd.to_numeric, errors="coerce"
    )
    df = df.dropna().sort_values("VG")

    vg = df["VG"].values
    id_abs = np.abs(df["ID"].values)
    id_abs = gaussian_filter1d(id_abs, sigma=2)  # light smoothing
    log_id = np.log10(id_abs + 1e-30)            # avoid log(0)

    # Threshold voltage: max gm/ID criterion
    gm = np.gradient(id_abs, vg)
    gm_over_id = gm / id_abs
    idx_vth = np.argmax(gm_over_id)
    vth = vg[idx_vth]

    # Subthreshold slope (SS) in a 50 mV window above VTH
    slope_region = (vg >= vth) & (vg <= vth + 0.05)
    ss = np.nan
    if slope_region.sum() >= 3:
        m_ss, _ = np.polyfit(vg[slope_region], log_id[slope_region], 1)
```

```
        ss = 1.0 / abs(m_ss) * 1000.0  # mV/dec

    # Device physics constants
    q = 1.602e-19          # C
    k = 1.381e-23          # J/K
    T = temperature        # K
    Vt = k * T / q         # thermal voltage

    if not np.isnan(ss):
        n = (ss/1000.0) / (2.303 * Vt)
    else:
        n = np.nan

    # VOFF from linear fit of ln(ID) vs VG 150 mV below VTH
    fit_window = (vg < vth) & (vg > vth - 0.15)
    if fit_window.sum() >= 3:
        ln_id = np.log(id_abs[fit_window])
        m_fit, c_fit = np.polyfit(vg[fit_window], ln_id, 1)
        nVt = Vt * n

        vg0  = vg[fit_window][0]
        id0  = id_abs[fit_window][0]
        voff = vg0 - vth - nVt * (np.log(id0) - (m_fit * vg0 + c_fit))
    else:
        voff = np.nan

    # Ioff: current at VGS = 0 V (if present)
    idx_ioff = np.where(vg == min(vg))[0]
    ioff = id_abs[idx_ioff[0]] if idx_ioff.size else np.nan

    # Mobility estimate in linear region (VD != 0)
    vd = pd.to_numeric(df["VD"]).values[0] if "VD" in df.columns else 0
    mu = np.nan
    if vd != 0 and not np.isnan(vth):
        cox = 8.32e-3       # F/m^2
        l = 0.15e-6         # m
        w = 0.42e-6         # m
        vg_eff = vg - vth
        linear = vg_eff > 0
        id_lin = id_abs[linear]
        vg_eff_lin = vg_eff[linear]
        if id_lin.size:
            mu = np.mean(id_lin / (cox * vg_eff_lin * vd)) * l / w
            mu *= 1e4        # convert to cm^2/Vs

    return {
        "Vth (V)": vth,
        "Voff (V)": voff,
        "Subthreshold Slope (mV/dec)": ss,
        "Subthreshold Factor n": n,
        "Ioff (A)": ioff,
        "Mobility (cm^2/Vs)": mu,
    }

# Plot & report

def plot_transfer_characteristics(db_blocks, output_dir="plots", temperature=4.0):
    os.makedirs(output_dir, exist_ok=True)

    for i, df in enumerate(db_blocks):
        if not {"VG", "ID", "VD"}.issubset(df.columns):
```

```
            print(f"[Warning] Skipping block {i} (missing columns)")
            continue

        df[["VG", "ID", "VD"]] = df[["VG", "ID", "VD"]].apply(
            pd.to_numeric, errors="coerce"
        )
        df = df.dropna()
        params = extract_parameters(df, temperature=temperature)

        vs = df.get("VS", pd.Series(["?"])).iloc[0]
        vd = df.get("VD", pd.Series(["?"])).iloc[0]
        vb = df.get("VB", pd.Series(["?"])).iloc[0]

        plt.semilogy(df["VG"], df["ID"].abs(), "bo-", label="|ID| vs VG")
        plt.xlabel("VG (V)")
        plt.ylabel("|ID| (A)")

        title = (
            f"Block {i}: Vth={params['Vth (V)']:.3f} V, Voff={params['Voff (V)']:.3f} V,"
            f" SS={params['Subthreshold Slope (mV/dec)']:.1f} mV/dec, n={params['Subthreshold Factor n']:.2f},"
            f" Ioff={params['Ioff (A)']:.1e} A, µ={params['Mobility (cm^2/Vs)']:.1f} cm^2/Vs,"
            f" VS={vs}, VD={vd}, VB={vb}"
        )
        plt.title(title)
        plt.grid(True, which="both", linestyle="--")
        plt.legend()
        plt.tight_layout()
        plt.show()
        plt.close()

        print(f"Block {i} Parameters:")
        print(f"  VS: {vs}, VD: {vd}, VB: {vb}")
        for k, v in params.items():
            print(f"  {k}: {v:.3e}" if isinstance(v, float) else f"  {k}: {v}")

# Main

def main():
    file_path = (
        r"C:\Users\olafd\Documents\Studie\BAP\cryo\sky130_fd_pr__nfet_01v8"
        r"_w0p42u_l0p15u_m1(8392_11_12_IDVG)_4K.mdm"
    )
    db_blocks = parse_mdm(file_path)
    print(f"Parsed {len(db_blocks)} data blocks.")
    plot_transfer_characteristics(db_blocks, temperature=4.0)

if __name__ == "__main__":
    main()
```

A

Appendix B

A.1. Design rules for mixed signal simulator

The design conventions needed for run the co-simulator are:

- Every block instance should start with X.
- No IO-pin must be used. Make an output and input pin which is connected to it-self.
- Mixed signal conversion needs a python code to be attached to the symbol file using SYMATTR File .py.
- Same point as above but the with .v file
- Only squarewave functions can be used as an inputsignal or the user has to give an array with timestamp and value pairs.

A.2. Simulation results timestep 1 ns

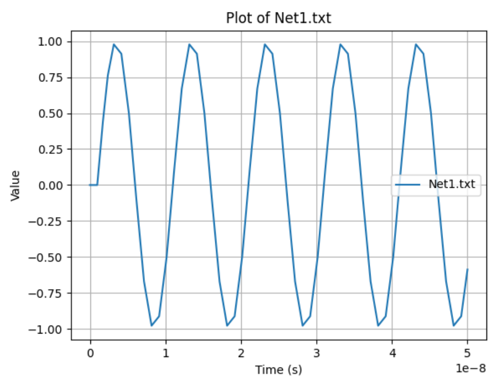


Figure A.1: Net 1 signal between voltage source and lowpass filter

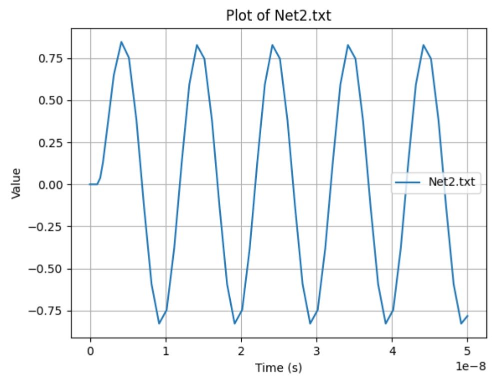


Figure A.2: Net 2 signal between lowpass filter and ADC

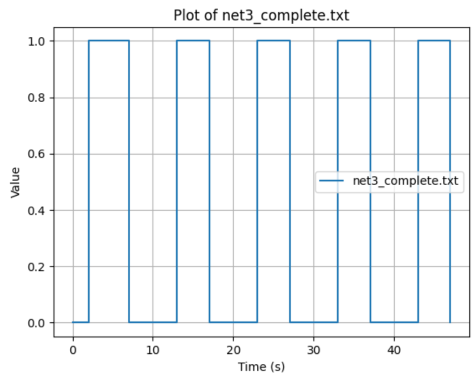


Figure A.3: Net 3 signal between ADC and inverter

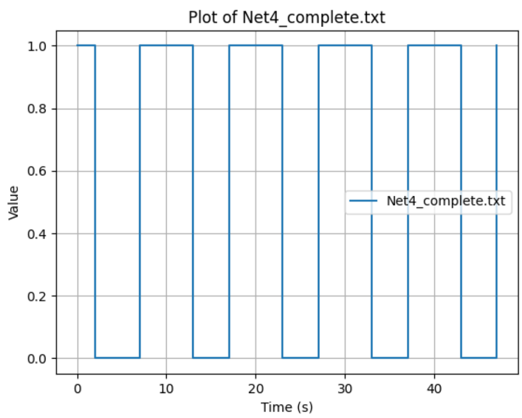


Figure A.4: Net 4 signal between inverter and DAC as LSB

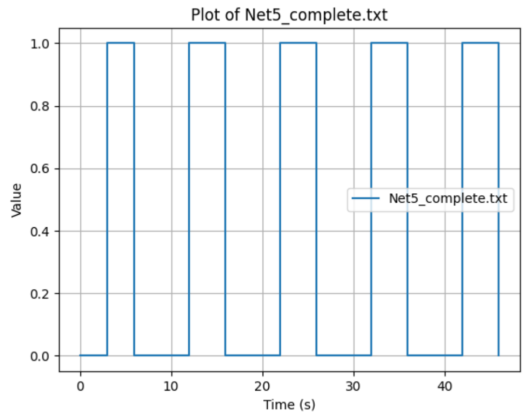


Figure A.5: Net 5 signal between ADC and DAC as middle bit

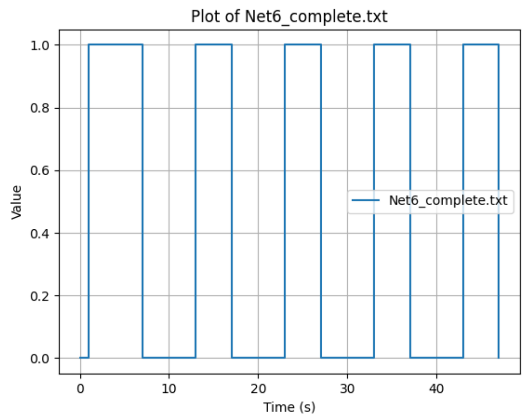


Figure A.6: Net 6 signal between ADC and DAC as middle bit

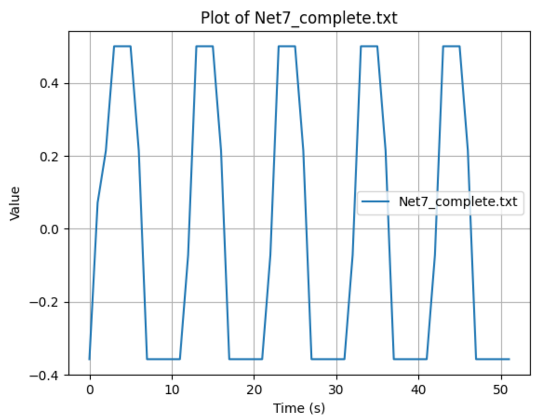


Figure A.7: Net 7 signal between DAC and voltage divider

A.3. Simulation results timestep 50 ps

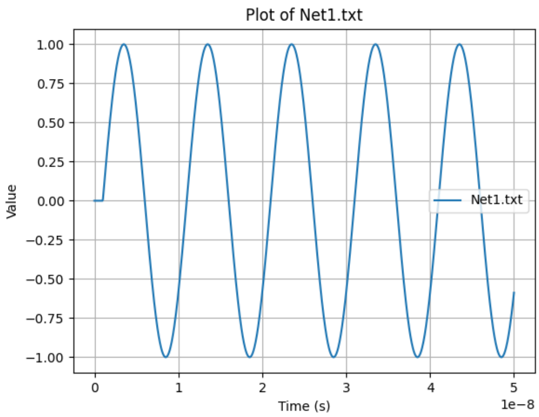


Figure A.8: Net 1 signal between voltage source and lowpass filte

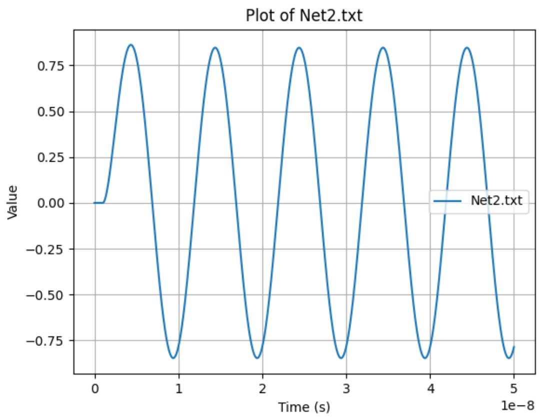


Figure A.9: Net 2 signal between lowpass filter and ADC

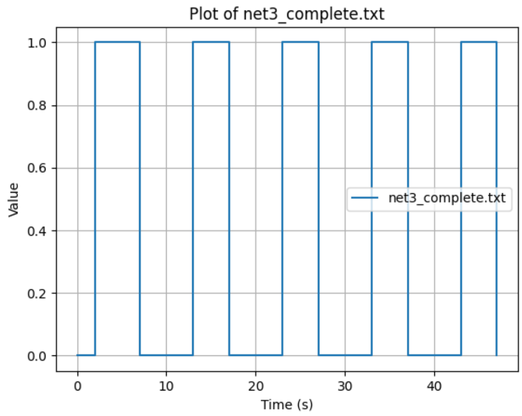


Figure A.10: Net 3 signal between ADC and inverter

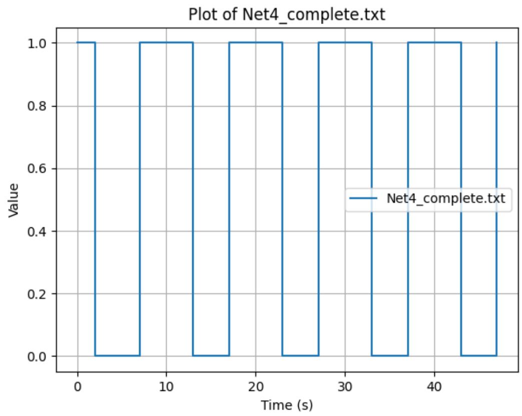


Figure A.11: Net 4 signal between inverter and DAC as LSB

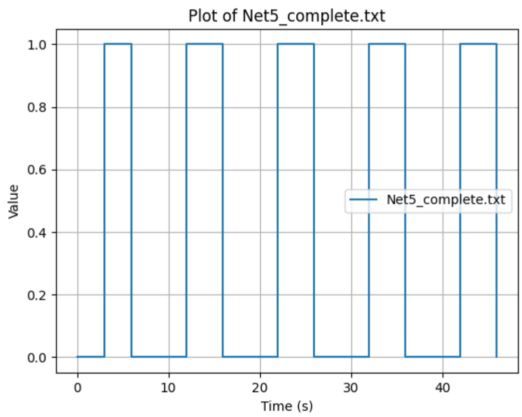


Figure A.12: Net 5 signal between ADC and DAC as middle bit

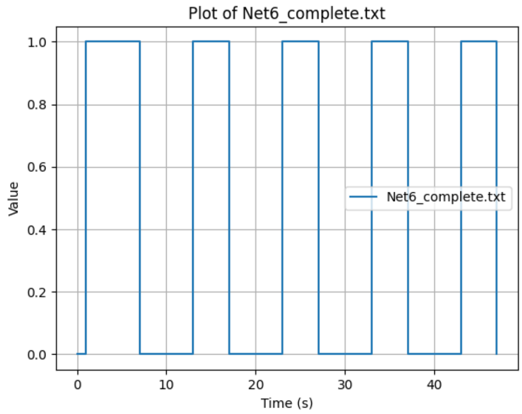


Figure A.13: Net 6 signal between ADC and DAC as middle bit

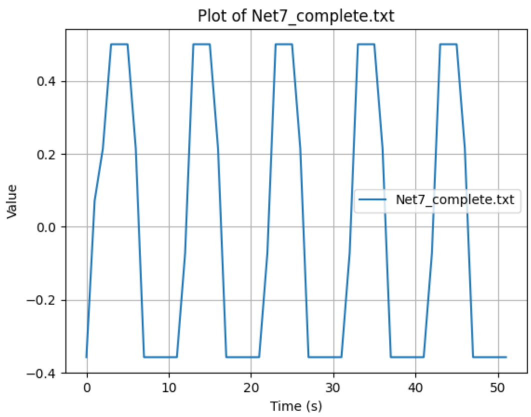


Figure A.14: Net 7 signal between DAC and voltage divider