eSim Semester Long Internship Autumn 2025

on

IHP OpenPDK Integration in eSim

Submitted by

Akshay Ajit Rukade
Haripriyan R

Under the guidance of

Prof. Prabhu Ramachandran
Principal Investigator
Department of Aerospace Engineering
Indian Institute of Technology Bombay

February 12, 2026

Acknowledgement

I would like to express my deepest gratitude to Prof. Prabhu Ramachandran for providing me the opportunity to be a part of the FOSSEE internship program and for supporting open-source engineering tool development at IIT Bombay. His guidance and vision have encouraged students and researchers to actively contribute to the open-source ecosystem.

I would also like to acknowledge Prof. Kannan M. Moudgalya for his foundational role in establishing and nurturing the FOSSEE initiative. His contributions toward open-source education and the creation of the FOSSEE fellowship framework have directly shaped the platform through which this internship was undertaken.

My sincere appreciation extends to my mentor, **Sumanto Kar**, for his continual support, technical guidance, and encouragement throughout the duration of this project. His insights and feedback played a key role in refining ideas, overcoming challenges, and ensuring timely completion of the tasks assigned to me.

I would also like to thank my internal mentors, **Mr. Varad Patil**, **Ms. Shanthi Priya K**, and **Mr. Aditya M**, for their valuable guidance, coordination, and technical inputs during the internship. Their mentorship contributed significantly to the clarity, progress, and successful execution of the work.

This internship has been an enriching learning experience, allowing me to work closely with open-source EDA tools, develop IC subcircuits in eSim, and gain exposure to real-world circuit modeling and simulation workflows. The knowledge acquired during this period will undoubtedly support my future academic and professional pursuits.

I would also like to thank the entire FOSSEE team for their coordination, assistance, and timely interactions at various stages of this work. Their collective efforts ensured smooth workflow, resource accessibility, and effective project execution.

Haripriyan R
Akshay Ajit Rukade

Abstract

eSim is an open-source Electronic Design Automation (EDA) tool developed by the FOSSEE team at IIT Bombay for circuit design, simulation, analysis, and PCB design. This project focuses on the integration of the **IHP Open PDK (SG13G2)** — a high-performance 0.13 μm BiCMOS process — into eSim.

To support this PDK, the underlying SPICE engine, Ngspice, was upgraded to version **45.2**, and support for the **Open Source Device Interface (OSDI)** was enabled. OSDI allows the fast and portable loading of Verilog-A compact models compiled via the **OpenVAF** compiler. Significant modifications were made to the eSim Python codebase, particularly in the netlist processing (`Processing.py`), netlist conversion (`Convert.py`), device model GUI (`DeviceModel.py`), and the main converter (`KicadtoNgspice.py`). The eSim installation scripts were updated with IHP PDK install/uninstall logic, a new KiCad symbol library (`eSim_IHP.kicad_sym`) was created with ∼25 IHP device symbols, and the KiCad symbol library table was updated accordingly. On the NGHDL side, `install-nghdl.sh` was modified to compile Ngspice with `--enable-osdi` and `--enable-openmp` flags along with the required build dependencies, and `model_generation.py` was fixed to enforce Unix line endings and correct VHDL compilation order.

This report presents a comprehensive comparison between the original `eSim-2.5` codebase and the modified `eSim-2.5` codebase, documenting every change made so that future interns can pick up the work exactly where we left off. The contributions were submitted via the following Pull Requests:

- **eSim PR #453**: https://github.com/FOSSEE/eSim/pull/453
- **eSim PR #454**: https://github.com/FOSSEE/eSim/pull/454
- **NGHDL PR #96**: https://github.com/FOSSEE/nghdl/pull/96
- **NGHDL PR #97**: https://github.com/FOSSEE/nghdl/pull/97

Contents

Chapter 1

Introduction to eSim

1.1 Overview

eSim (previously known as Oscad/FreeEDA) is a free and open-source EDA tool created by the FOSSEE (Free/Libre and Open Source Software for Education) team at IIT Bombay. It is used for circuit design, simulation, analysis, and PCB design. eSim integrates several powerful open-source tools — KiCad, Ngspice, GHDL, and Verilator — into a unified graphical user interface built with Python and PyQt5.

It serves as an alternative to proprietary software such as OrCAD, PSpice, LTspice, and Proteus, making advanced EDA capabilities accessible to students, educators, and professionals worldwide without licensing costs.

1.2 History and Motivation

The primary motivation behind eSim is to provide a robust, open-source alternative for electronic circuit design. Proprietary tools are often expensive and restrict access to the underlying algorithms and models. The FOSSEE project, funded by the National Mission on Education through ICT (NMEICT), Ministry of Education, Government of India, aims to promote the use of FOSS tools in education.

eSim was originally developed as "Oscad" and later renamed. It has been continuously improved through contributions from FOSSEE interns and the open-source community. Key milestones include:

- Integration of KiCad for schematic capture and PCB layout.
- Integration of Ngspice as the simulation engine.
- NGHDL for mixed-signal simulation using GHDL/Verilator.
- SKY130 PDK integration for open silicon design.
- **IHP SG13G2 PDK integration (this project)**.

1.3 Architecture

eSim integrates several powerful open-source tools into a unified GUI:

- **KiCad**: Used for schematic capture and PCB layout. Users draw circuit diagrams using a library of components. KiCad generates a netlist (`.cir` file) that is passed to the converter.
- **Ngspice**: The simulation engine. It is a mixed-level/mixed-signal circuit simulator based on Berkeley's SPICE3f5, XSPICE, and CIDER. It performs Transient, AC, DC, Noise, and other analyses.
- **GHDL / Verilator**: VHDL and Verilog simulators used for mixed-signal simulations involving digital components through the NGHDL subsystem.
- **Python Frontend**: The glue layer. The `src/` directory contains Python scripts that orchestrate the workflow: reading the KiCad netlist, presenting a GUI for model parameters, converting the netlist to Ngspice format, invoking the simulator, and plotting results.

1.4 The KiCad-to-Ngspice Pipeline

The most critical subsystem for this project is the **KiCad-to-Ngspice converter** pipeline, located in `src/kicadtoNgspice/`. The pipeline follows these steps:

1. **Read Netlist**: `Processing.py` reads the `.cir` file generated by KiCad.
2. **Pre-process**: Parameters are replaced, the netlist is lowercased, and continuation lines (starting with `+`) are merged.
3. **Separate Info**: Lines starting with `.` are separated as *options* (e.g., `.end`, `.param`), and the rest is *schematicInfo*.
4. **Source Identification**: Voltage/current sources (v, i) are identified and their parameters (sine, pulse, dc, etc.) are parsed.
5. **IC Conversion**: Components starting with `u` or `U` are matched against XML model files in `library/modelParamXML/`.
6. **GUI Presentation**: `KicadtoNgspice.py` presents a window with tabs for Analysis, Source Details, Model, Device Model, and Subcircuit. Users fill in parameters.
7. **Conversion**: `Convert.py` generates the final `.cir.out` file with all `.model`, `.include`, and `.lib` directives.
8. **Simulation**: Ngspice executes the `.cir.out` file and generates results.

Understanding this pipeline is **critical** for future interns, as all PDK integrations (SKY130, IHP, and future PDKs) hook into steps 4–7.

1.5 Importance of PDK Support

A Process Design Kit (PDK) is a set of files used to model a fabrication process. For eSim to be viable for IC design, it must support standard PDKs. Initially, eSim supported only generic SPICE models. The SKY130 PDK integration (by previous interns) was the first step toward supporting foundry-level PDKs. The IHP SG13G2 integration (this project) is the second, and it introduces the additional complexity of OSDI-based compact models.

1.6 Scope of This Project

1. Upgrade Ngspice to version 45.2 with `--enable-osdi` support.
2. Install OpenVAF and compile IHP Verilog-A models to `.osdi` format.
3. Integrate the IHP Open PDK into eSim's installation and runtime workflow.
4. Modify eSim's Python source to correctly detect, parse, and simulate IHP components.
5. Create installation automation scripts and documentation.
6. Provide a working example circuit (`IHP_Inverter`) for verification.

Chapter 2

eSim Installation and Setup

2.1 System Requirements

Development and testing were conducted on **Ubuntu Linux** (versions 22.04 and 24.04).

- **OS**: Ubuntu 22.04 LTS or 24.04 LTS recommended.
- **RAM**: Minimum 4 GB (8 GB recommended for Ngspice compilation).
- **Storage**: At least 10 GB of free space for eSim, Ngspice source, PDK files, and compiled OSDI models.
- **Internet**: Required for downloading dependencies and cloning repositories.

2.2 Standard eSim Installation

The main entry point is `install-eSim.sh`, which detects the Ubuntu version and calls the appropriate versioned script.

```
$ git clone https://github.com/FOSSEE/eSim.git
$ cd eSim-2.5
$ chmod +x install-eSim.sh
$ ./install-eSim.sh --install
```

Listing 2.1: Standard Installation

The script performs these actions:

1. Detects Ubuntu version (22.04, 23.04, or 24.04).
2. Calls the corresponding script in `install-eSim-scripts/` (e.g., `install-eSim-24.04.sh`).
3. Installs system dependencies: `python3`, `python3-pyqt5`, `python3-pip`, `libngspice0`, `kicad`, etc.
4. Installs Python packages: `matplotlib`, `numpy`, `hdlparse`, etc.
5. Configures ngspice and sets up environment variables.

2.3 Directory Structure of eSim

```
eSim-2.5/
+-- src/                        # Python source code
|   +-- frontEnd/               # Main GUI
|   +-- kicadtoNgspice/         # Netlist converter (KEY)
|   |   +-- Convert.py          # Final netlist generation
|   |   +-- Processing.py       # Netlist pre-processing
|   |   +-- DeviceModel.py      # Device library GUI
|   |   +-- KicadtoNgspice.py   # Main converter window
|   |   +-- SubcircuitTab.py    # Subcircuit handling
|   |   +-- TrackWidget.py      # Widget state tracker
|   +-- modelEditor/            # Model editing tools
|   +-- converter/              # LTSpice/other converters
+-- library/                    # Component libraries
|   +-- kicad_libs/             # KiCad symbols/footprints
|   +-- modelParamXML/          # XML model definitions
|   +-- deviceModelLibrary/     # .lib SPICE model files
+-- Examples/                   # Example projects
+-- install-eSim.sh             # Main installer
+-- install-eSim-scripts/       # OS-specific install scripts
+-- ihp/                        # IHP PDK installer (NEW)
+-- nghdl.zip                   # NGHDL package
```

Listing 2.2: eSim Directory Structure

2.4 Installation Scripts Modified

Three installation scripts were modified in `eSim-2.5`:

- `install-eSim-22.04.sh`
- `install-eSim-23.04.sh`
- `install-eSim-24.04.sh`

These scripts were updated to include the Ngspice 45.2 compilation from source with the `--enable-osdi` flag, rather than relying on the potentially outdated system package.

Chapter 3

System Upgrades: Ngspice and OSDI

3.1 Why Ngspice 45.2?

eSim was originally packaged with an older version of Ngspice (version 30 or above), which was installed via the Bundled version with NGHDL. While this version was sufficient for basic SPICE simulation with generic device models, it **did not include OSDI support**. The IHP SG13G2 PDK relies on compact device models written in Verilog-A (such as the PSP103 MOSFET model) that are compiled into `.osdi` shared libraries. Loading these models at runtime requires the **Open Source Device Interface (OSDI)**, which was introduced in **Ngspice version 43**.

Since the system-packaged Ngspice lacked this capability, it became necessary to compile Ngspice from source with the `--enable-osdi` configure flag. Version 45.2 was chosen as the target because it includes:

- Full OSDI support (available from version 43 onwards) for dynamically loading compiled Verilog-A models at runtime.
- Improved convergence algorithms for complex transistor models such as PSP and BSIM.
- Better PSP model support, which is the primary MOSFET model used by the IHP PDK.
- Bug fixes relevant to subcircuit instantiation and corner library loading.

3.2 What is OSDI?

OSDI (Open Source Device Interface) is a game-changer for open-source simulation. Traditionally, adding a new device model (e.g., a specific HBT from IHP) to Ngspice required:

1. Writing the model in C code.
2. Modifying Ngspice's source.

3. Recompiling the entire simulator.

With OSDI, device models written in **Verilog-A** can be compiled into a shared library (`.osdi` file) using a compiler like OpenVAF. Ngspice can then load these models at runtime:

```
set osdi_enabled
osdi /path/to/psp103.osdi
osdi /path/to/r3_cmc.osdi
```

Listing 3.1: OSDI Loading in Ngspice spinit

3.3 Compiling Ngspice from Source

```
# Download source
wget https://sourceforge.net/projects/ngspice/files/\
  ng-spice-rework/45/ngspice-45.tar.gz
tar -xzvf ngspice-45.tar.gz
cd ngspice-45

# Configure with OSDI enabled
    ./configure \
        --enable-osdi \
        --enable-xspice \
        --enable-openmp \
        --disable-debug \
        --prefix=$HOME/$nghdl/install_dir \
        --exec-prefix=$HOME/$nghdl/install_dir

# Compile and install
make -j$(nproc)
sudo make install

# Verify
ngspice --version   # Should show 45
```

Listing 3.2: Ngspice 45.2 Compilation with OSDI

3.4 OpenVAF Compiler

OpenVAF is a Verilog-A compiler that generates OSDI binaries. It is required to compile the IHP PDK's compact models.

```
wget https://openva.fra1.cdn.digitaloceanspaces.com/\
  openvaf_23_5_0_linux_amd64.tar.gz
tar -xzhf openvaf_23_5_0_linux_amd64.tar.gz
sudo mv openvaf /usr/bin/
openvaf --version
```

Listing 3.3: Installing OpenVAF

3.5 OSDI Files Required by IHP PDK

After compilation, the following `.osdi` files are generated and must be loaded by Ngspice:

Table 3.1: OSDI Files Required by IHP SG13G2 PDK

OSDI File	**Purpose**
`psp103.osdi`	PSP MOS transistor model
`psp103_nqs.osdi`	PSP Non-Quasi-Static model
`r3_cmc.osdi`	CMC resistor model
`mosvar.osdi`	MOS varactor model

Chapter 4

IHP Open PDK Integration

4.1 Overview of IHP SG13G2

The IHP SG13G2 is a high-performance 0.13 μm SiGe BiCMOS technology provided by **IHP – Leibniz Institute for High Performance Microelectronics** (Frankfurt/Oder, Germany). Key features:

- 0.13 μm CMOS with high-speed SiGe HBTs
- f_T/f_{max} up to 300/500 GHz
- High-quality passives (MIM capacitors, precision resistors)
- Fully open-source under permissive license
- Available at: https://github.com/IHP-GmbH/IHP-Open-PDK

4.2 The IHP Installation Script

We developed `ihp/ihp-install-script.sh` (a new file, 337 lines) to automate the PDK setup.

4.2.1 Key Functions

1. **check_ngspice_osdi()**: Verifies that Ngspice has OSDI support by checking the output of `ngspice -v` for keywords "osdi" or "45".
2. **install_dependencies()**: Installs git, cmake, build-essential, klayout, python3-pip, and pip packages (pandas, h5py).
3. **install_ihp_pdk()**: Clones the IHP-Open-PDK repository to `$PDK_ROOT`.
4. **install_openvaf()**: Downloads and installs the OpenVAF compiler.
5. **compile_osdi_models()**: Runs the PDK's `openvaf-compile-va.sh` to compile all Verilog-A models to `.osdi` format.

6. **configure_spiceinit()**: Modifies the NGHDL `spinit` file to enable OSDI and load IHP models.

4.2.2 Spinit Configuration

This is the most critical step. The script appends the following to the NGHDL spinit file:

```
* ========== IHP-Open-PDK OSDI Models ==========
if $?osdi_enabled
  osdi /path/to/ihp-sg13g2/libs.tech/ngspice/osdi/psp103.osdi
  osdi /path/to/ihp-sg13g2/libs.tech/ngspice/osdi/r3_cmc.osdi
  osdi /path/to/ihp-sg13g2/libs.tech/ngspice/osdi/mosvar.osdi
end
* ========== End IHP-Open-PDK ==========
```

Listing 4.1: Spinit Modifications for IHP OSDI

4.3 IHP PDK Library File Structure

Future interns should be familiar with the library structure used by the IHP PDK:

Table 4.1: IHP Corner Library Files and Default Corners

Corner File	Device Type	Available Corners
`cornerMOSlv.lib`	Low-V MOSFET	mos_tt, mos_ff, mos_ss, mos_sf, mos_fs
`cornerMOShv.lib`	High-V MOSFET	mos_tt, mos_ff, mos_ss
`cornerRES.lib`	Resistors	res_typ, res_bcs, res_wcs
`cornerCAP.lib`	Capacitors	cap_typ, cap_bcs, cap_wcs
`cornerDIO.lib`	Diodes	dio_tt, dio_bcs, dio_wcs
`cornerHBT.lib`	HBT Bipolar	hbt_typ, hbt_bcs, hbt_wcs

```
$PDK_ROOT/ihp-sg13g2/libs.tech/ngspice/
+-- models/
|   +-- cornerMOSlv.lib       # .lib "..." mos_tt
|   +-- cornerMOShv.lib       # .lib "..." mos_tt
|   +-- cornerRES.lib         # .lib "..." res_typ
|   +-- cornerCAP.lib         # .lib "..." cap_typ
|   +-- cornerDIO.lib         # .lib "..." dio_tt
|   +-- cornerHBT.lib         # .lib "..." hbt_typ
|   +-- sg13g2_moslv_mod.lib  # (included by corner files)
|   +-- sg13g2_moshv_mod.lib  # (included by corner files)
+-- osdi/
    +-- psp103.osdi
    +-- psp103_nqs.osdi
    +-- r3_cmc.osdi
    +-- mosvar.osdi
```

Listing 4.2: IHP PDK Models Directory Structure

Chapter 5

Comprehensive Code Comparison

This chapter documents **every single change** identified between `eSim-2.5` (original) and `eSim-2.5` (modified).

Architecture at a Glance

The changes follow the data flow below. Modifications were made at each numbered stage:

```
KiCad Schematic (.kicad_sch)
    |
    v
[1] Processing.py     -- skip IHP from source detection
    |
    v
[2] DeviceModel.py    -- IHP GUI panel (path/corner/params)
    |
    v
[3] KicadtoNgspice.py -- QComboBox widget value extraction
    |
    v
[4] Convert.py        -- .lib include + ihp -> xihp prefix
    |
    v
Ngspice Netlist (.cir.out)
```

5.1 Summary of All Changes

The changes span two repositories: **eSim** (PRs #453 and #454) and **NGHDL** (PRs #96 and #97).

Table 5.1: Complete Change Summary

Category	Count
Modified Files (eSim)	8
Modified Files (NGHDL)	2
New Files (eSim)	2
Binary Files Updated	1
New Directories	2
Total Changes	**15**

5.2 Modified Files – Complete List

5.2.1 eSim Repository (PRs #453 and #454)

Table 5.2: Modified Files – eSim

#	PR	File Path
1	#453	`Ubuntu/install-eSim-scripts/install-eSim-22.04.sh`
2	#453	`Ubuntu/install-eSim-scripts/install-eSim-23.04.sh`
3	#453	`Ubuntu/install-eSim-scripts/install-eSim-24.04.sh`
4	#454	`library/kicadLibrary/template/sym-lib-table`
5	#454	`src/kicadtoNgspice/Convert.py`
6	#454	`src/kicadtoNgspice/DeviceModel.py`
7	#454	`src/kicadtoNgspice/KicadtoNgspice.py`
8	#454	`src/kicadtoNgspice/Processing.py`

5.2.2 NGHDL Repository (PRs #96 and #97)

Table 5.3: Modified Files – NGHDL

#	PR	File Path
1	#96	`src/model_generation.py`
2	#97	`Ubuntu/install-nghdl.sh`

5.3 New Files and Directories

Table 5.4: New Files Added (from PRs)

#	PR	Path
1	eSim #454	`ihp/ihp-install-script.sh` (337 lines – IHP PDK installer)
2	eSim #454	`library/kicadLibrary/eSim-symbols/eSim_IHP.kicad_sym` (327 lines – 25 IHP symbols)

Table 5.5: Binary Files Updated

#	PR	Path
1	NGHDL #96	`nghdl-simulator-source.tar.xz` (updated Ngspice source archive)

Table 5.6: New Directories

#	Path
1	`ihp/` (IHP PDK installation scripts)
2	`Examples/IHP_Inverter/` (IHP example circuit)

Chapter 6

Detailed File-by-File Analysis

This chapter provides the exact code changes for each modified file. Each section follows the same format:

1. **Location** – file path (with the repository name for NGHDL files).
2. **Change** – summary of lines added/modified.
3. **Purpose** – why this change was necessary.
4. **Code** – before/after snippets you can `diff` against.
5. **For future interns** – what to watch out for when extending this code.

6.1 Processing.py

Location: `src/kicadtoNgspice/Processing.py`
Change: 1 line modified (line 141–142).
Purpose: Prevent IHP components (whose references start with `ihp`) from being processed as voltage or current sources.

6.1.1 The Problem

In the original code, the `insertSpecialSourceParam` function checks if a component name starts with `v` or `i` to identify voltage/current sources. However, the IHP integration introduces components like `ihpmode1` (which starts with `i`) that are **not** current sources. Without this fix, the code attempts to access `words[3]` for these components, causing an `IndexError`.

6.1.2 The Fix

```
# ORIGINAL:
if compName[0] == 'v' or compName[0] == 'i':
```

Listing 6.1: Processing.py – Original (line 141)

```
# MODIFIED: Skip IHP components that start with 'ihp'
if (compName[0] == 'v' or compName[0] == 'i') \
        and not compName.startswith('ihp'):
```

Listing 6.2: Processing.py – Modified (lines 141–142)

For future interns: If you add a new PDK with component prefixes starting with `v` or `i`, you will need to add similar exclusion logic here.

6.2 DeviceModel.py

Location: `src/kicadtoNgspice/DeviceModel.py`
Change: Major addition — ˜330 new lines.
Purpose: Add a complete GUI panel for IHP device library management.

6.2.1 Detection Logic (lines 71–91)

The `__init__` method was modified to detect IHP components **before** SKY130 or general libraries:

```
# Check for IHP SG13G2 PDK first
has_ihp = False
for line in schematicInfo:
    words = line.split()
    if len(words) >= 2:
        if words[0].startswith('ihp'):
            has_ihp = True
            break
        if 'sg13_' in words[-1].lower():
            has_ihp = True
            break

if has_ihp:
    self.eSim_ihp(schematicInfo)
elif "sky130" in " ".join(schematicInfo):
    self.eSim_sky130(schematicInfo)
else:
    self.eSim_general_libs(schematicInfo)
```

Listing 6.3: DeviceModel.py – IHP Detection (new code)

6.2.2 Model-to-Corner Mapping (lines 275–319)

A comprehensive dictionary maps every known IHP model to its corner library file and corner type:

```
self.ihp_model_to_corner = {
    # MOS Low-Voltage
    'sg13_lv_nmos': ('cornerMOSlv.lib', 'mos'),
    'sg13_lv_pmos': ('cornerMOSlv.lib', 'mos'),
    # MOS High-Voltage
    'sg13_hv_nmos': ('cornerMOShv.lib', 'mos'),
    'sg13_hv_pmos': ('cornerMOShv.lib', 'mos'),
    # Resistors
    'rppd': ('cornerRES.lib', 'res'),
    'rhigh': ('cornerRES.lib', 'res'),
    # Capacitors
    'cap_cmim': ('cornerCAP.lib', 'cap'),
    # Diodes
    'dantenna': ('cornerDIO.lib', 'dio'),
    # HBT (Bipolar)
    'npn13g2': ('cornerHBT.lib', 'hbt'),
    # ... (19 models total)
}
```

Listing 6.4: DeviceModel.py – IHP Model Mapping

For future interns: If new IHP device models are added to the PDK, you must add them to this dictionary.

6.2.3 The eSim_ihp() Method (lines 266–494)

This is the largest single addition. It creates a PyQt5 GUI panel for each IHP device with:

- **Library Path**: Text field with "Add" and "Default" buttons. Default auto-fills from `$PDK_ROOT`.
- **Corner Selection**: A `QComboBox` dropdown with corner options (e.g., mos_tt, mos_ff, mos_ss).
- **Parameters**: Text field for device parameters (e.g., `W=1u L=130n nf=1`).

The tracking format stored in `deviceModelTrack` is: `lib_path:corner:params`.

6.2.4 Helper Methods

- `trackDefaultIHPDeviceLib()` (lines 496–517): Sets the default PDK library path.
- `trackIHPDeviceLibrary()` (lines 519–543): Opens a file browser for custom library selection.
- `ihpCornerChanged()` (lines 545–556): Handles corner dropdown changes.
- `ihpParamChanged()` (lines 558–568): Handles parameter text changes.

6.3 Convert.py

Location: `src/kicadtoNgspice/Convert.py`
Change: Significant additions in `addDeviceLibrary()`.
Purpose: Handle IHP device libraries during netlist conversion.

6.3.1 IHP Device Detection

The `addDeviceLibrary` method was extended with detection logic:

```
ihp_known_devices = ['rppd', 'rhigh', 'rsil', 'ptap1',
    'ntap1', 'dantenna', 'dpantenna', 'isolbox',
    'pnpmpa', 'nmoscl_2', 'nmoscl_4', 'cparasitic',
    'dpwdnw', 'ddnwpsub']

is_ihp_device = (
    eachline[0:3] == 'ihp' or
    'sg13' in model_name or
    'npn13' in model_name or
    model_name.startswith('cap_') or
    model_name in ihp_known_devices
)
```

Listing 6.5: Convert.py – IHP Detection Logic

6.3.2 Library Path Parsing

For IHP devices, the library path stored in `deviceModelTrack` uses the format `path:corner:params`:

```
if is_ihp_device:
    ihp_parts = completeLibPath.split(':')
    ihp_lib_path = ihp_parts[0]
    ihp_corner = ihp_parts[1] if len(ihp_parts) > 1 \
                 else "mos_tt"
    ihp_params = ihp_parts[2] if len(ihp_parts) > 2 \
                 else ""

    # Add .lib statement
    lib_include = '.lib "' + ihp_lib_path + '" ' \
                  + ihp_corner
    if lib_include not in includeLine:
        includeLine.append(lib_include)
```

Listing 6.6: Convert.py – Library Parsing

6.3.3 Prefix Conversion

Ngspice requires subcircuit instances to begin with `x` or `X`. IHP components use the `ihp` prefix, so they are converted to `xihp`:

```
# Convert: ihp1 -> xihp1
words[0] = words[0].replace('ihp', 'xihp')
if ihp_params:
    words.append(ihp_params)
deviceLine[index] = words
```

Listing 6.7: Convert.py – Prefix Conversion

6.3.4 Legacy ihpmode Support

For backward compatibility, the code also handles `ihpmode` marker lines. These are legacy components that serve as mode indicators:

```
if eachline[0:7] == 'ihpmode':
    # Parse ihpmode: path:corner
    # Add all .lib includes for the PDK
    # Comment out the ihpmode line
    schematicInfo[index] = "* " + eachline
```

Listing 6.8: Convert.py – Legacy ihpmode Handling

6.4 KicadtoNgspice.py

Location: `src/kicadtoNgspice/KicadtoNgspice.py`
Change: Modified the `callConvert()` method.
Purpose: Handle the new `QComboBox` widgets introduced by the IHP DeviceModel panel. The original code assumed all `entry_var` widgets were `QLineEdit` (which have a `.text()` method). IHP corner selection uses `QComboBox` (which requires `.currentText()`).

6.4.1 The Fix

```
# ORIGINAL:
ET.SubElement(attr_var, "field").text = \
    str(obj_devicemodel.entry_var[it].text())

# MODIFIED: Handle both QComboBox and QLineEdit
widget = obj_devicemodel.entry_var[it]
if hasattr(widget, 'currentText'):
    widget_text = str(widget.currentText())
else:
    widget_text = str(widget.text())
ET.SubElement(attr_var, "field").text = widget_text
```

Listing 6.9: KicadtoNgspice.py – QComboBox Handling (callConvert)

For future interns: If you add new widget types (e.g., `QSpinBox`) to the DeviceModel panel for a future PDK, you will need to add their value-extraction method here as well.

6.5 KiCad Symbol Library: eSim_IHP.kicad_sym

Location: `library/kicadLibrary/eSim-symbols/eSim_IHP.kicad_sym`
Change: New file (327 lines).
Purpose: Provide KiCad schematic symbols for all IHP SG13G2 PDK components so users can place them in circuit schematics.

6.5.1 Symbols Defined

The library contains approximately 25 symbols covering all IHP device categories:

Table 6.1: IHP KiCad Symbols in eSim_IHP.kicad_sym

Category	Symbols	Description
MOSFET (LV)	`sg13_lv_nmos`, `sg13_lv_pmos`	Low-voltage NMOS/PMOS (4-pin: D, G, S, B)
MOSFET (HV)	`sg13_hv_nmos`, `sg13_hv_pmos`	High-voltage NMOS/PMOS
HBT (4-pin)	`npn13G2`, `npn13G2l`, `npn13G2v`	SiGe HBTs (C, B, E, BN)
HBT (5-pin)	`npn13G2_5t`, `npn13G2l_5t`, `npn13G2v_5t`	HBTs with thermal pin (C, B, E, BN, T)
PNP	`pnpMPA`	PNP bipolar (C, B, E)
Capacitors	`cap_cmim`, `cap_rfcmim`, `cparasitic`	MIM and parasitic capacitors
Resistors	`rsil`, `rhigh`, `rppd`, `Rparasitic`	Silicided, high-R, and parasitic resistors
Taps	`ptap1`, `ntap1`	P-well and N-well substrate taps
Diodes	`dantenna`, `dpantenna`, `ddnwpsub`, `dpwdnw`	Antenna and well diodes
Other	`isolbox`, `nmoscl_2`, `nmoscl_4`, `bondpad`	Isolation, clamp, and pad cells

All symbols use the `ihp` reference prefix and include `Sim.Device = SUBCKT` and `Sim.Pins` properties for Ngspice simulation mapping.

6.6 KiCad Symbol Library Table Update

Location: `library/kicadLibrary/template/sym-lib-table`
Change: 1 line added.
Purpose: Register the new `eSim_IHP` symbol library so KiCad can discover and display IHP components.

```
(lib (name "eSim_IHP")(type "KiCad")
  (uri "${KICAD6_SYMBOL_DIR}/eSim_IHP.kicad_sym")
  (options "")(descr "IHP SG13G2 Open PDK Symbols"))
```

Listing 6.10: sym-lib-table – New Entry

6.7 NGHDL: model_generation.py

Location: `src/model_generation.py` (NGHDL repository)
Change: 2 modifications in `createServerScript()` and `createSockScript()`.
Purpose: Fix two issues that caused failures on WSL/Linux.

6.7.1 Fix 1: Unix Line Endings

Shell scripts generated on Windows used CRLF line endings, which caused `/bin/bash^M: bad interpreter` errors. The fix forces Unix-style LF line endings:

```
# ORIGINAL:
start_server = open('start_server.sh', 'w')
sock_pkg_create = open('sock_pkg_create.sh', 'w')

# MODIFIED: Force Unix line endings (LF)
start_server = open('start_server.sh', 'w', newline='\n')
sock_pkg_create = open('sock_pkg_create.sh', 'w', newline='\n')
```

Listing 6.11: model_generation.py – Unix Line Endings Fix

6.7.2 Fix 2: VHDL Compilation Order

The original code used `ghdl -i *.vhdl` which imports all VHDL files without resolving dependencies, causing "`sock_pkg has not been analyzed`" errors. The fix explicitly compiles dependency files first:

```
# ORIGINAL:
start_server.write("ghdl -i *.vhdl &&\n")

# MODIFIED: Compile in correct dependency order
start_server.write("ghdl -a sock_pkg.vhdl &&\n")
start_server.write("ghdl -a Utility_Package.vhdl &&\n")
start_server.write("ghdl -a Vhpi_Package.vhdl &&\n")
start_server.write("ghdl -a *.vhdl &&\n")
```

Listing 6.12: model_generation.py – VHDL Compilation Order Fix

For future interns: If you add new VHDL package files to NGHDL, they must be added to the explicit compilation order in this function.

6.8 NGHDL: install-nghdl.sh

Location: `Ubuntu/install-nghdl.sh` (NGHDL repository)
Change: 2 modifications.
Purpose: Enable Ngspice OSDI support in the NGHDL build.

6.8.1 OSDI Build Dependencies

A new dependency block was added to `installDependency()` to install the build tools required for compiling Ngspice from source with OSDI:

```
sudo apt install -y \
    build-essential autoconf automake libtool \
    flex bison libreadline-dev libncurses5-dev \
    libxaw7 libxaw7-dev libx11-dev
```

Listing 6.13: install-nghdl.sh – New OSDI Dependencies

6.8.2 Configure Flags for OSDI

The Ngspice `./configure` invocation was updated from a single-line call to include OSDI and OpenMP flags:

```
# ORIGINAL:
../configure --enable-xspice --disable-debug \
    --prefix=$HOME/$nghdl/install_dir/ \
    --exec-prefix=$HOME/$nghdl/install_dir/

# MODIFIED:
../configure \
    --enable-osdi \
    --enable-xspice \
    --enable-openmp \
    --disable-debug \
    --prefix=$HOME/$nghdl/install_dir \
    --exec-prefix=$HOME/$nghdl/install_dir
```

Listing 6.14: install-nghdl.sh – Updated Configure Flags

Table 6.2: Ngspice Configure Flags – Quick Reference

Flag	Purpose
`--enable-osdi`	Load `.osdi` compact device models (**required** for IHP PDK)
`--enable-xspice`	Enable XSPICE code-model extensions (used by NGHDL)
`--enable-openmp`	Parallel simulation threads for multi-core speedup
`--disable-debug`	Strip debug symbols for smaller binary

Chapter 7

Example Circuit and Testing

7.1 IHP Inverter Example

A working CMOS inverter example using IHP SG13G2 components was created at `Examples/IHP_Inverter/`

```
* Simple CMOS Inverter with IHP SG13G2 PDK
* No ihpmode marker needed - detection via sg13

* PMOS transistor (pull-up)
ihp1  out in vdd vdd sg13_lv_pmos

* NMOS transistor (pull-down)
ihp2  out in gnd gnd sg13_lv_nmos

* Power supply
v1  vdd gnd dc

* Input pulse
v2  in gnd pulse

* Plot markers
U1  out plot_v1
U2  in plot_v1
U3  vdd plot_v1

.end
```

Listing 7.1: IHP_Inverter.cir – Test Circuit

7.1.1 Expected Converted Output

After running the converter, the `.cir.out` file should contain:

```
.lib "/path/to/cornerMOSlv.lib" mos_tt
xihp1 out in vdd vdd sg13_lv_pmos W=2u L=130n nf=1
xihp2 out in gnd gnd sg13_lv_nmos W=1u L=130n nf=1
v1 vdd gnd 1.2
v2 in gnd pulse(0 1.2 0 10p 10p 50n 100n)
```

Listing 7.2: Expected .cir.out Output

7.1.2 Component Naming Convention

Table 7.1: IHP Component Naming Convention

Designator	Type	Example
`ihpmode1`	IHP mode marker (legacy)	`ihpmode1 gnd sg13_lv_nmos`
`ihp1`, `ihp2`	IHP transistors	`ihp1 out in vdd vdd sg13_lv_pmos`
`v1`, `v2`	Voltage sources	`v1 vdd gnd dc`
`U1`, `U2`	Plot markers	`U1 out plot_v1`

7.2 Testing Instructions

1. Open eSim.
2. Navigate to `Examples/IHP_Inverter/`.
3. Click **KiCad to Ngspice** converter.
4. Verify the IHP mode panel appears (with "IHP Device" group boxes).
5. Set PDK path or click "Default".
6. Select corner: `mos_tt`.
7. Enter parameters: `W=1u L=130n nf=1` for both devices.
8. Click **Convert**.
9. Check the `.cir.out` file for correct output.
10. Run the simulation and verify waveform output.

7.2.1 Verification Checklist

Use this checklist to confirm a successful setup:

[ ] `ngspice --version` outputs `45.2` (or later)

[ ] `ls $PDK_ROOT/IHP-Open-PDK-*/` lists the IHP model files

[ ] `.spiceinit` contains `osdi` load commands

[ ] KiCad **Symbol Library Manager** shows `eSim_IHP`

[ ] IHP Inverter simulation produces an output waveform

Chapter 8

Guidance for Future Interns

8.1 Where We Left Off

Phase 1 of the IHP integration is complete. The following has been verified working:

- IHP detection triggers correctly in the converter pipeline.
- `.lib` includes are generated for all corner files.
- `.spiceinit` includes OSDI loading.
- `ihp1` → `xihp1` prefix conversion works.
- Simulation produces a raw data file.

8.2 What Remains (Phase 2+)

The following tasks are recommended for future work:

1. **Model XML Files**: Add XML model definitions in `library/modelParamXML/` for IHP devices so they can be discovered automatically.
2. **KLayout/Magic Integration**: Add Design Rule Checking (DRC) and Layout vs Schematic (LVS) support.
3. **Standard Cell Library**: Integrate IHP's standard cell library for digital design.
4. **Automated Corner Sweep**: Add functionality to sweep all process corners automatically and generate comparison plots.
5. **UI Enhancement**: Add a dedicated IHP PDK menu in eSim's main window.
6. **More PDKs**: The OSDI framework can be extended to support GF180MCU and other PDKs.

8.3 Key Files to Study

If you are a future intern, here is the recommended reading order:

1. `src/kicadtoNgspice/Processing.py` – Understand netlist pre-processing.
2. `src/kicadtoNgspice/DeviceModel.py` – Understand the GUI layer (study `eSim_sky130` first, then `eSim_ihp`).
3. `src/kicadtoNgspice/Convert.py` – Understand the final conversion step.
4. `src/kicadtoNgspice/KicadtoNgspice.py` – Understand QComboBox/QLineEdit widget handling.
5. `ihp/ihp-install-script.sh` – Understand the installation automation.
6. `Examples/IHP_Inverter/` – Run the example to verify your setup.

8.3.1 Quick `grep` Commands

To quickly locate all IHP-related code in the eSim codebase:

```
# All IHP detection logic
grep -rn 'ihp\|sg13\|IHP' src/kicadtoNgspice/

# Corner mapping dictionary
grep -n 'ihp_model_to_corner' src/kicadtoNgspice/DeviceModel.py

# KiCad symbol definitions
grep -c 'symbol' library/kicadLibrary/eSim-symbols/eSim_IHP.kicad_sym

# OSDI configure flags (NGHDL)
grep -n 'enable-osdi\|enable-openmp' Ubuntu/install-nghdl.sh
```

Listing 8.1: Finding IHP Code in the Source Tree

8.4 Common Pitfalls

- **OSDI version mismatch**: Ensure Ngspice is compiled with the same OSDI version expected by the `.osdi` files.
- **Component prefix confusion**: `ihpmode` starts with `i`, which can be confused with a current source. Always check for `ihp` prefix before treating a component as a source.
- **Corner naming**: Corner names are PDK-specific. IHP uses `mos_tt` (not just `tt` like SKY130). Make sure to use the correct corner name in the `.lib` directive.
- **Path issues**: Paths must be absolute in `.lib` directives. Relative paths will fail.
- **NGHDL line endings**: If you generate shell scripts on Windows (e.g., via `model_generation.py`), always use `newline='\n'` in the `open()` call to force Unix (LF) line endings. CRLF will cause "bad interpreter" errors when the script runs on Linux/WSL.

- **VHDL compilation order**: GHDL requires packages to be compiled before files that depend on them. If you add new VHDL package files, add explicit `ghdl -a <package>.vhdl` lines before the wildcard `ghdl -a *.vhdl` in `model_generation.py`.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

The integration of the IHP Open PDK (SG13G2) into eSim was successfully completed in Phase 1, spanning modifications across both the eSim and NGHDL repositories. By upgrading Ngspice to version 45.2 with OSDI support (via `--enable-osdi` and `--enable-openmp` flags), installing the OpenVAF compiler, and making targeted modifications to key Python source files and installation scripts, we enabled eSim to simulate circuits designed with IHP's 130 nm BiCMOS technology.

Across the 4 Pull Requests, a total of 10 source files were modified, 2 new files were created, and 2 new directories were added.

The project successfully demonstrated:

1. Automated Ngspice 45.2 installation with OSDI (both standalone and via NGHDL).
2. Automated IHP PDK installation and OSDI model compilation.
3. Correct netlist generation for IHP components.
4. A complete KiCad symbol library (`eSim_IHP.kicad_sym`) with ∼25 IHP device symbols.
5. A working CMOS inverter example using IHP SG13G2 devices.
6. NGHDL fixes for cross-platform compatibility (Unix line endings, VHDL compilation order).

9.2 Future Work

- Model XML file generation for automatic component discovery.
- DRC/LVS integration via KLayout.
- Support for additional PDKs (GF180MCU).
- Automated process corner sweep and comparison plots.

- Standard cell library integration for IHP digital design.