eSim Semester Long Internship Spring 2026

on

OpenROAD Integration with eSim

Submitted by

Rishabh Jain

Jaypee University of Engineering & Technology

Under the guidance of

Prof. Prabhu Ramachandran

Principal Investigator

Department of Aerospace Engineering

Indian Institute of Technology Bombay

May 30, 2026

Acknowledgement

I would like to express my sincere 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, Mr. 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 the successful completion of the assigned tasks.

I would also like to thank my internal mentors, Mr. Varad Patil and Ms. Shanthi Priya K, 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, integrate OpenROAD into eSim, and gain practical exposure to RTL-to-GDSII digital design flows. 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 and the open-source community 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.

Rishabh Jain

Abstract

eSim is a free and open-source Electronic Design Automation (EDA) tool developed by the FOSSEE team at IIT Bombay. It provides an integrated environment for circuit design, simulation, and analysis using tools such as KiCad, Ngspice, GHDL, and Verilator through a Python-based graphical interface.

This internship project focuses on the integration of **OpenROAD**, an open-source RTL-to-GDSII digital physical design flow, with eSim. The objective of the project was to enable users to directly perform backend physical design operations from the eSim environment, thereby extending eSim from circuit simulation to complete digital layout generation.

The implementation involved modifying the eSim frontend by updating `Application.py` to include an OpenROAD execution button within the graphical user interface. A backend automation module, `OpenROAD.py`, was developed to manage the OpenROAD Flow Scripts (ORFS) execution process. Additionally, a Python-based converter script, `netlist2rtl.py`, was implemented to automatically generate synthesizable Verilog code from eSim-generated `.cir.out` netlist files.

The integrated workflow was validated using two example circuits: **FullAdder** and **HalfAdder**. The generated Verilog files were successfully processed through the OpenROAD RTL-to-GDSII flow, producing backend design outputs including synthesized Verilog, DEF floorplan files, GDSII layout files, timing constraint files, and parasitic extraction reports.

The generated layouts and backend outputs were verified using the OpenROAD GUI and KLayout visualization tools. Multiple debugging and integration issues related to file handling, OpenROAD execution paths, DEF loading, RTL generation, and GUI execution were identified and resolved during the implementation process.

The complete implementation and project repositories are publicly available at:

- `https://github.com/FOSSEE/eSim-to-OpenROAD_Design_Flow_Plugin`
- `https://github.com/RISHABH12005/eSim-OpenROAD-Plugin`

Contents

List of Figures

Chapter 1

Introduction to eSim

1.1 Overview

eSim (previously known as Oscad/FreeEDA) is a free and open-source Electronic Design Automation (EDA) tool developed by the FOSSEE (Free/Libre and Open Source Software for Education) project at IIT Bombay. It provides an integrated environment for circuit design, simulation, analysis, and PCB design using open-source tools such as KiCad, Ngspice, GHDL, and Verilator through a unified Python-based graphical user interface built using Python and PyQt5.

eSim serves as an accessible alternative to proprietary EDA software such as OrCAD, PSpice, and LTspice. Its open-source architecture allows students, researchers, and developers to study the internal implementation of the tool, contribute improvements, and extend functionalities according to project requirements.

The platform is supported under the National Mission on Education through ICT (NMEICT), Ministry of Education, Government of India, with the objective of promoting open-source software usage in engineering education and research.

1.2 Open-Source Circuit Simulation

The primary simulation engine used in eSim is Ngspice, which supports DC, AC, transient, and noise analysis. KiCad is integrated for schematic capture, while GHDL and Verilator enable mixed-signal simulation support through the NGHDL subsystem.

The modular and open-source nature of eSim makes it highly suitable for educational projects, internship contributions, and research-oriented development. During this internship, the same architecture was extended to integrate OpenROAD for RTL-to-GDSII backend physical design flow support.

1.3 eSim Workspace and Project Structure

The eSim workspace organizes individual circuits as separate project directories. Each project typically contains simulation files, generated netlists, schematic files, and backend outputs required during circuit analysis and implementation.

Common project files include:

- `.cir` — KiCad-generated circuit netlist
- `.cir.out` — processed Ngspice-compatible netlist
- `.sch` — schematic design file
- `.sub` — subcircuit definitions

Figure 1.1 shows the eSim workspace and file manager interface used during the project implementation.

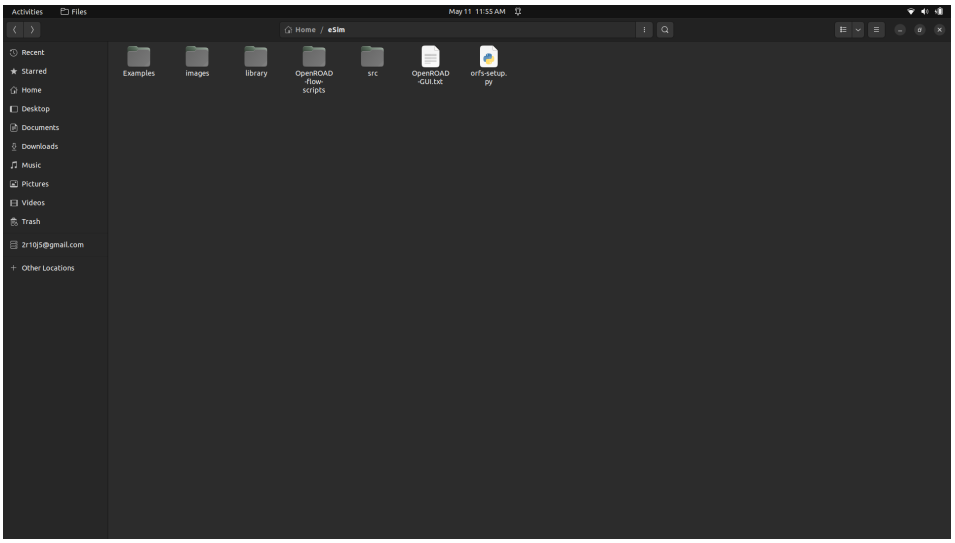

Figure 1.1. eSim workspace file manager showing project structure.

1.4 Project Organisation

The OpenROAD integration workflow was tested using example circuit projects such as **FullAdder** and **HalfAdder**. Each project directory contained generated netlists, synthesized Verilog files, backend physical design outputs, and generated layout files.

Figure 1.2 shows the FullAdder project structure inside the eSim project panel.

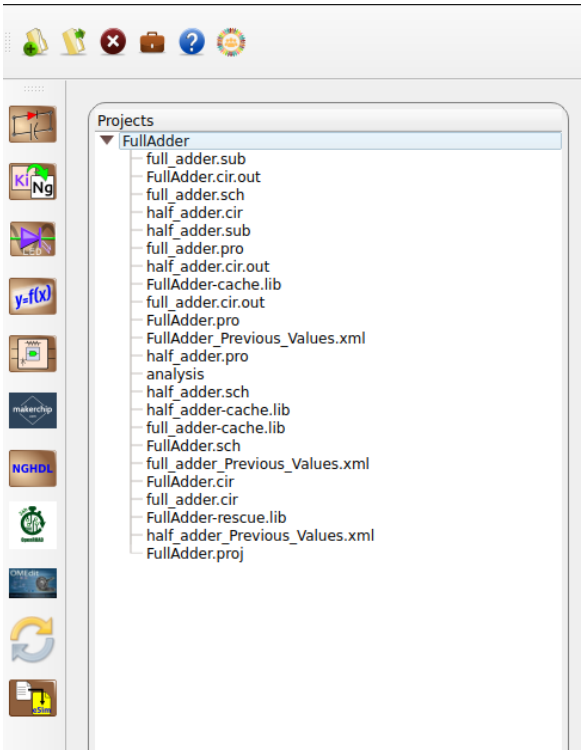

Figure 1.2. FullAdder project structure inside the eSim project panel.

1.5 Internship Objective

The primary objective of this internship was to integrate the **OpenROAD** digital physical design flow into eSim, enabling users to execute a complete RTL-to-GDSII workflow directly from the eSim graphical interface.

The major objectives of the project included:

- Understanding the eSim frontend and project workflow.
- Adding an OpenROAD execution button in the eSim graphical interface.
- Developing a backend Python module to execute OpenROAD Flow Scripts (ORFS).
- Implementing a netlist-to-RTL converter for automatic Verilog generation.
- Generating backend physical design outputs such as DEF and GDSII files.
- Testing the complete workflow using FullAdder and HalfAdder circuits.
- Visualizing generated layouts using OpenROAD GUI and KLayout.

1.6 Project Directory Structure

The integrated OpenROAD-eSim system follows the directory structure shown below:

```
eSim/
|-- src/
|   |-- frontEnd/
|   |   `-- Application.py
|   |
|   `-- maker/
|       |-- OpenROAD.py
|       `-- netlist2rtl.py
|
|-- Examples/
|   |-- FullAdder/
|   |   |-- FullAdder.cir.out
|   |   |-- FullAdder.v
|   |   |-- FullAdder.def
|   |   |-- FullAdder.gds
|   |   `-- logs/
|   |
|   `-- HalfAdder/
|       |-- HalfAdder.cir.out
|       |-- HalfAdder.v
|       |-- HalfAdder.def
|       |-- HalfAdder.gds
|       `-- logs/
|
`-- OpenROAD-flow-scripts/
    `-- flow/
        |-- designs/
        |   `-- sky130hd/
        |       |-- FullAdder/
        |       `-- HalfAdder/
        |
        `-- results/
            `-- sky130hd/
                |-- FullAdder/
                `-- HalfAdder/
```

Listing 1.1. Project directory structure used during integration.

The above structure was used throughout the internship for OpenROAD integration, backend automation, testing, and generated file management.

Chapter 2

eSim Installation and Setup

2.1 System Requirements

Development and testing for the OpenROAD integration were performed on **Ubuntu 22.04 LTS**. The following system configuration was used during implementation and testing:

- **Operating System:** Ubuntu 22.04 LTS
- **RAM:** Minimum 4 GB (8 GB recommended)
- **Storage:** At least 20 GB free disk space
- **Internet Connection:** Required for repository cloning and dependency installation
- **Python Version:** Python 3.8 or later with PyQt5 support

The installation environment also included OpenROAD-flow-scripts (ORFS), KLayout, Yosys, and additional Python dependencies required for backend automation.

2.2 Standard eSim Installation

The eSim source code was cloned from the official FOSSEE GitHub repository and installed using the provided installation script.

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

Listing 2.1. Standard eSim installation commands.

The installation script automatically installs the required packages including Python, PyQt5, Ngspice, KiCad, and related system dependencies.

2.3 Required Dependencies

Additional dependencies were installed to support OpenROAD integration and RTL-to-GDSII backend flow execution.

```
$ sudo apt update

$ sudo apt install -y build-essential git cmake
$ sudo apt install -y python3-dev python3-pip
$ sudo apt install -y klayout

$ pip3 install --user pyqt5
```

Listing 2.2. Additional dependencies for OpenROAD integration.

These dependencies were required for GUI execution, Python backend scripting, layout visualization, and ORFS workflow execution.

2.4 Workspace Setup and Project Creation

After installation, eSim was launched from the terminal or application menu. New projects were created using the *New Project* option available in the eSim graphical interface.

Each project generated its own directory containing schematic files, generated netlists, simulation outputs, and backend design files required during OpenROAD execution.

2.5 GUI Overview and OpenROAD Button

The eSim graphical interface provides project management, schematic editing, simulation control, and tool integration support. During this internship, an additional **OpenROAD** execution button was integrated into the main toolbar of the application.

The button was connected with the backend automation flow implemented in `OpenROAD.py`. When triggered, it automatically processed the selected project and initiated the OpenROAD RTL-to-GDSII flow.

Figure 2.1 shows the OpenROAD integration button added to the eSim GUI toolbar.

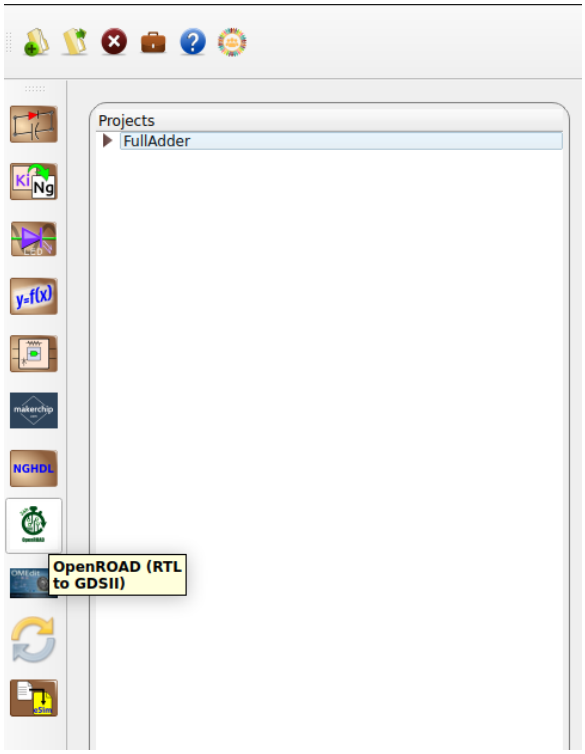

Figure 2.1. OpenROAD integration button added to the eSim GUI toolbar.

2.6 Installation Verification

After installation and integration, the setup was verified by launching eSim and confirming that:

- the graphical interface opened correctly,
- the OpenROAD button was visible in the toolbar,
- existing projects could be loaded successfully,
- generated `.cir.out` files were valid,
- OpenROAD Flow Scripts executed without errors.

Figure 2.2 shows the terminal output generated during project execution and ORFS backend flow processing.

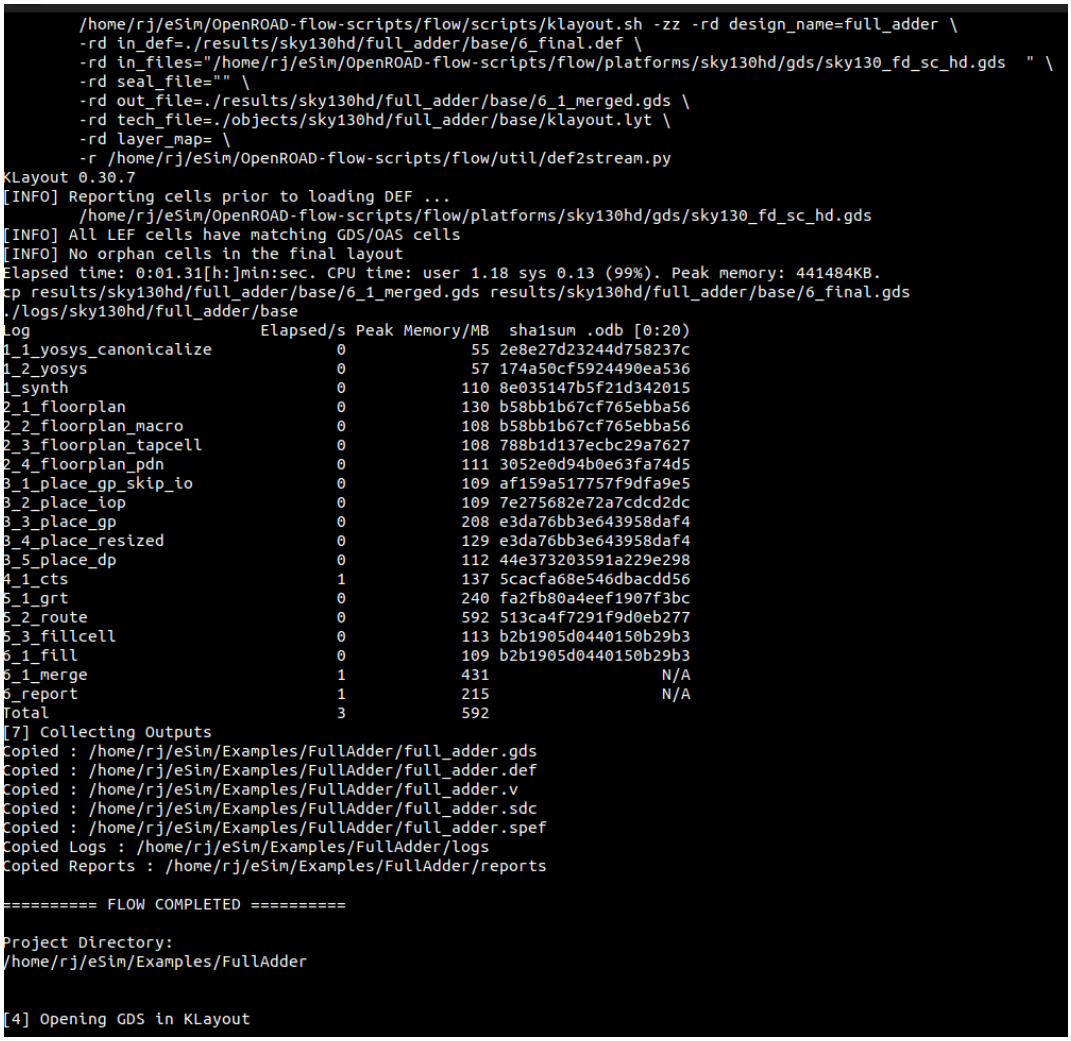

Figure 2.2. Terminal output showing the eSim project build and ORFS execution process.

Chapter 3

OpenROAD Project and Installation

3.1 OpenROAD Overview

OpenROAD is an open-source RTL-to-GDSII digital physical design flow developed under the DARPA OpenROAD project. It provides an automated backend implementation flow starting from synthesized RTL and ending with a manufacturable GDSII layout.

The OpenROAD ecosystem consists of multiple open-source tools integrated into a unified backend flow. Major components used during this internship include:

- **Yosys** — RTL synthesis tool
- **OpenROAD** — Physical design engine
- **OpenSTA** — Static timing analysis
- **TritonRoute** — Detailed routing engine
- **KLayout** — GDSII layout viewer

The backend implementation flow used during this project was based on the **sky130hd** standard cell library provided by the SkyWater 130 nm open-source PDK.

3.2 OpenROAD Flow Scripts (ORFS)

OpenROAD Flow Scripts (ORFS) provide an automated Makefile-based backend implementation flow for RTL-to-GDSII design generation.

The complete flow includes the following stages:

1. **Synthesis** — Verilog RTL is synthesized into a gate-level netlist using Yosys.
2. **Floorplanning** — Core area and die area are generated for physical implementation.

3. **Placement** — Standard cells are placed inside the core region.
4. **Clock Tree Synthesis (CTS)** — Clock distribution buffers are inserted.
5. **Routing** — Global routing and detailed routing are performed.
6. **GDSII Generation** — Final layout is exported as a `.gds` file.

The ORFS infrastructure was used as the primary backend flow engine during the OpenROAD integration with eSim.

3.3 OpenROAD GUI

The OpenROAD GUI provides a graphical environment for viewing floorplans, standard-cell placement, routing information, and final layout structures generated during backend implementation.

Figure 3.1 shows the OpenROAD GUI displaying the generated backend layout for the FullAdder design.

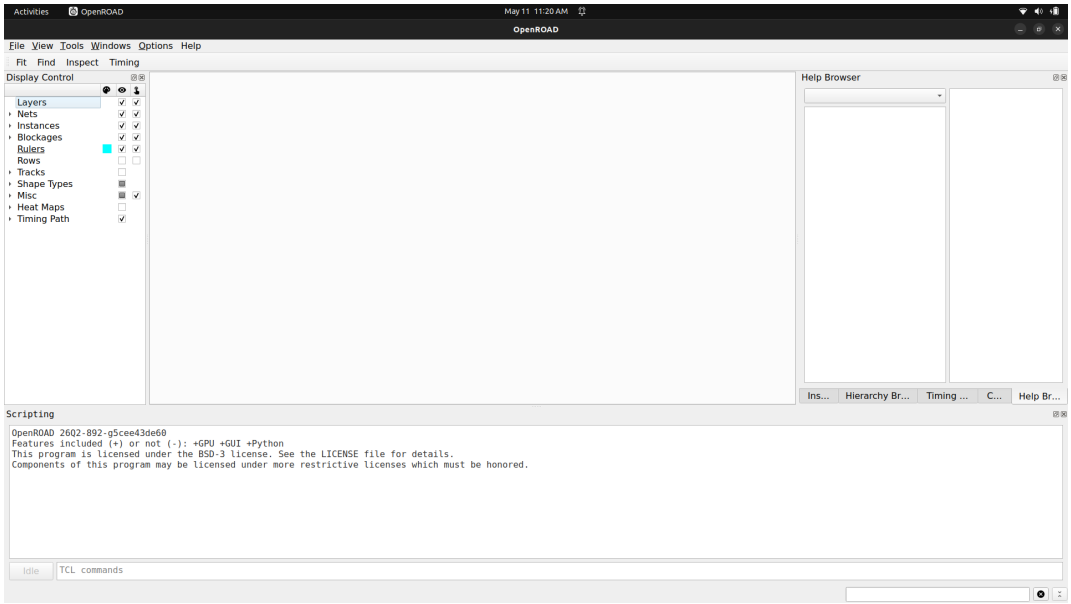

Figure 3.1. OpenROAD GUI showing the generated physical design layout.

3.4 OpenROAD Installation

OpenROAD Flow Scripts were installed by cloning the official repository and executing the provided setup scripts.

```
$ git clone --recursive \
https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts

$ cd OpenROAD-flow-scripts

$ sudo ./setup.sh
```

```

$ ./build_openroad.sh --local
```

Listing 3.1. OpenROAD Flow Scripts installation commands.

The installation process downloads required dependencies, builds OpenROAD from source, and installs the required backend tools locally inside the ORFS directory.

After compilation, the OpenROAD executable becomes available at:

```
OpenROAD-flow-scripts/tools/install/OpenROAD/bin/openroad
```

3.5 Yosys and KLayout Setup

Yosys was installed as part of the OpenROAD Flow Scripts build process. KLayout was installed separately for GDSII layout visualization and inspection.

```
$ sudo apt install -y klayout
```

Listing 3.2. KLayout installation command.

The generated layout files were verified using both OpenROAD GUI and KLayout during backend testing.

3.6 Environment Variables and Path Configuration

The backend integration module `OpenROAD.py` was configured to automatically locate the OpenROAD and Yosys binaries inside the ORFS installation directory.

The following configuration paths were used:

```
ORFS_PATH = "/home/rj/eSim/OpenROAD-flow-scripts"

OPENROAD_BIN = ORFS_PATH + \
"/tools/install/OpenROAD/bin/openroad"

YOSYS_BIN = ORFS_PATH + \
"/tools/install/yosys/bin/yosys"
```

Listing 3.3. Tool path configuration inside OpenROAD.py.

These paths were configured directly inside the backend module to avoid manual environment variable configuration during execution.

3.7 ORFS Directory Structure

The OpenROAD Flow Scripts infrastructure follows a structured design hierarchy for storing flow configuration files and generated backend outputs.

```
OpenROAD-flow-scripts/
‘-- flow/
    |-- designs/
```

```
|   `-- sky130hd/
|       |-- FullAdder/
|       |   |-- config.mk
|       |   `-- constraint.sdc
|       |
|       `-- HalfAdder/
|           |-- config.mk
|           `-- constraint.sdc
|
`-- results/
    `-- sky130hd/
        |-- FullAdder/
        |   |-- 6_final.gds
        |   |-- 6_final.def
        |   `-- ...
        |
        `-- HalfAdder/
            `-- ...
```

Listing 3.4. ORFS design directory structure.

The `designs/` directory stores configuration and constraint files, while the `results/` directory stores generated backend outputs such as DEF and GDSII layouts.

Figure 3.2 shows the terminal output generated during OpenROAD Flow Scripts execution.

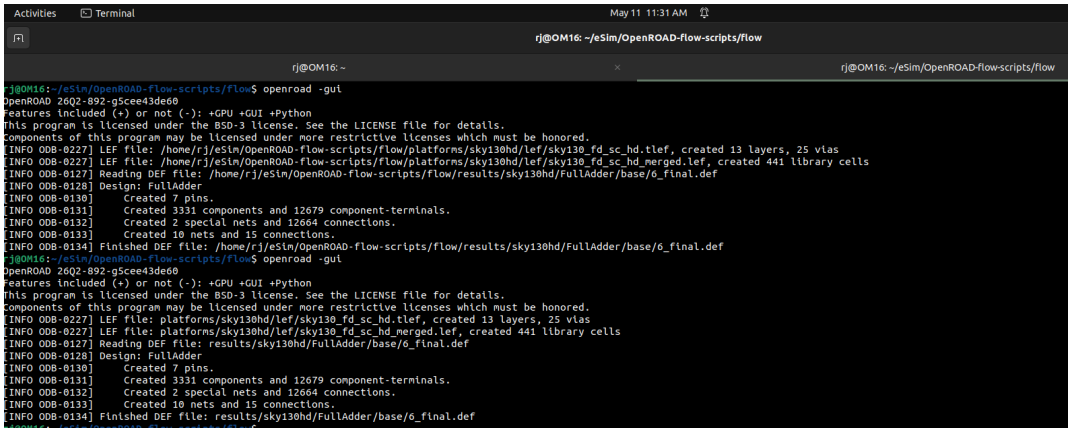

Figure 3.2. Terminal output during OpenROAD Flow Scripts execution.

Chapter 4

OpenROAD Integration with eSim

This chapter describes the complete implementation work carried out during the integration of OpenROAD with eSim. It explains the frontend modifications, backend automation modules, netlist conversion process, OpenROAD Flow Scripts execution, and the debugging steps performed during development.

4.1 Architecture Overview

The integration connects the eSim simulation environment with the OpenROAD RTL-to-GDSII backend flow. The overall workflow implemented during this internship is shown below:

1. The user selects a project inside eSim and clicks the **OpenROAD** button.
2. The button handler inside `Application.py` identifies the selected project and locates the generated `.cir.out` file.
3. The `netlist2rtl.py` module parses the `.cir.out` file and generates a synthesizable Verilog (`.v`) file.
4. The `OpenROAD.py` backend module automatically generates the required ORFS configuration files such as `config.mk` and `constraint.sdc`.
5. OpenROAD Flow Scripts execute synthesis, floorplanning, placement, CTS, routing, and GDSII generation.
6. The generated layout is visualized automatically using the OpenROAD GUI and KLayout.

4.2 Modification of Application.py

The eSim frontend is primarily controlled through the file `src/frontEnd/Application.py`. During this internship, a new OpenROAD execution button was added to the left-side toolbar of the graphical interface.

The following code snippet shows the implementation used for adding the OpenROAD button:

```
self.openroadAction = QtWidgets.QAction(
    QtGui.QIcon(os.path.join(
        self.resource_path,
        'openroad.png')),
    'OpenROAD',
    self
)

self.openroadAction.setStatusTip(
    'Run OpenROAD Flow'
)

self.openroadAction.triggered.connect(
    self.openOpenROAD
)

self.leftToolBar.addAction(
    self.openroadAction
)
```

Listing 4.1. Adding the OpenROAD button inside Application.py.

The button was connected with a backend execution handler that validates the selected project and launches the OpenROAD flow.

4.3 OpenROAD Button Handler

The handler function retrieves the currently selected project directory, validates the generated `.cir.out` file, and initializes the backend execution module.

```
def openOpenROAD(self):

    project_dir = self.obj_projectExplorer.curProject

    if not project_dir:
        QtWidgets.QMessageBox.warning(
            self,
            'No Project',
            'Please select a project.'
        )
        return

    project_name = os.path.basename(project_dir)

    cir_out = os.path.join(
        project_dir,
        project_name + '.cir.out'
    )

    if not os.path.isfile(cir_out):
```

```
        QtWidgets.QMessageBox.warning(
            self,
            'File Not Found',
            'Run KiCad-to-Ngspice conversion first.'
        )
        return

    from maker.OpenROAD import OpenROADFlow

    self.openroad_obj = OpenROADFlow(
        cir_out,
        project_dir
    )

    self.openroad_obj.run()
```

Listing 4.2. OpenROAD execution handler in Application.py.

Figure 4.1 shows the OpenROAD button integrated into the eSim GUI toolbar.

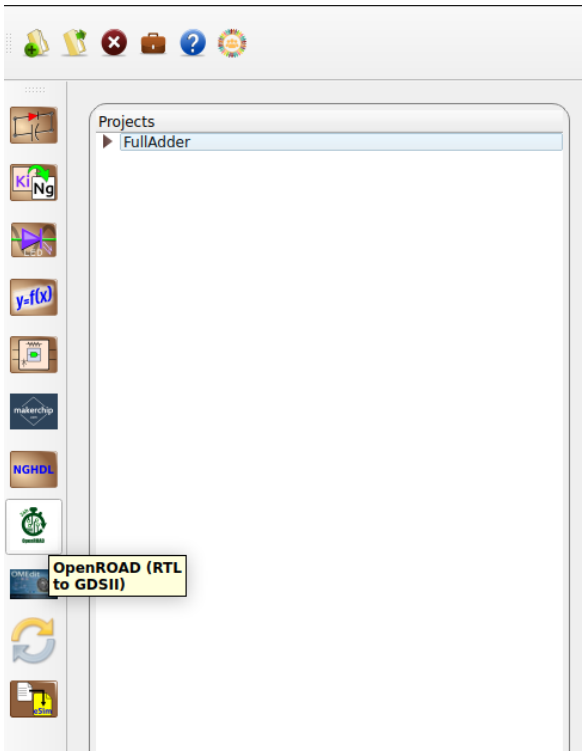

Figure 4.1. OpenROAD button integrated into the eSim GUI toolbar.

4.4 Implementation of netlist2rtl.py

The `netlist2rtl.py` module was developed to automatically convert eSim-generated `.cir.out` files into synthesizable Verilog RTL.

The parser reads the SPICE netlist line by line and extracts:

- module name,

- input ports,
- output ports,
- subcircuit instances,
- internal net connections.

4.4.1 Netlist Parsing

The following logic was used for parsing the `.cir.out` file:

```
def parse_cir_out(cir_out_path, project_dir):

    module_name = os.path.basename(project_dir)

    ports_in = []
    ports_out = []
    instances = []

    with open(cir_out_path, 'r') as f:

        for line in f:

            line = line.strip()

            if line.startswith('*') or not line:
                continue

            if line.lower().startswith('v'):

                parts = line.split()

                if len(parts) >= 3:

                    net = parts[1]

                    if net.lower() not in ('gnd', '0'):
                        ports_in.append(net)

            if line.lower().startswith('x'):
                instances.append(line)

    return (
        module_name,
        ports_in,
        ports_out,
        instances
    )
```

Listing 4.3. Core parsing logic in netlist2rtl.py.

4.4.2 Verilog Generation

After parsing, the extracted information is converted into a Verilog module.

```
def generate_verilog(
    module_name,
    ports_in,
    ports_out,
    instances,
    output_path
):

    lines = []

    all_ports = ports_in + ports_out

    lines.append(
        f'module {module_name} ('
    )

    lines.append(
        '    ' + ',\n    '.join(all_ports) + ');'
    )

    for p in ports_in:
        lines.append(f'input {p};')

    for p in ports_out:
        lines.append(f'output {p};')

    for inst in instances:
        lines.append(' ' + inst + ';')

    lines.append('endmodule')

    with open(output_path, 'w') as f:
        f.write('\n'.join(lines) + '\n')
```

Listing 4.4. Verilog generation in netlist2rtl.py.

4.5 Implementation of OpenROAD.py

The `OpenROAD.py` module serves as the primary backend controller for ORFS execution.

The module performs the following tasks:

- validates input files,
- generates ORFS configuration files,
- creates timing constraints,
- launches ORFS execution,

- verifies generated outputs,
- opens the generated layout in OpenROAD GUI.

4.5.1 config.mk Generation

The following configuration file was generated automatically for each project:

```
def generate_config_mk(
    design_name,
    verilog_path,
    output_dir
):

    config = f"""
export DESIGN_NICKNAME = {design_name}
export DESIGN_NAME     = {design_name}
export PLATFORM        = sky130hd

export VERILOG_FILES   = {verilog_path}

export SDC_FILE = \
./designs/$(PLATFORM)/\
$(DESIGN_NICKNAME)/constraint.sdc

export CORE_UTILIZATION  = 40
export CORE_ASPECT_RATIO = 1
export CORE_MARGIN       = 2
export PLACE_DENSITY     = 0.65
"""

    config_path = os.path.join(
        output_dir,
        'config.mk'
    )

    with open(config_path, 'w') as f:
        f.write(config.strip())

    return config_path
```

Listing 4.5. config.mk generation inside OpenROAD.py.

4.5.2 SDC File Generation

Timing constraints were automatically generated using the following logic:

```
def generate_sdc(
    design_name,
    ports_in,
    ports_out,
```

```
    output_path
):

    lines = [
        f'current_design {design_name}',
        'create_clock -name clk -period 10.0000'
    ]

    for p in ports_in:
        lines.append(
            f'set_input_delay 0.1000 '
            f'-add_delay [get_ports {{{p}}}]'
        )

    for p in ports_out:
        lines.append(
            f'set_output_delay 0.1000 '
            f'-add_delay [get_ports {{{p}}}]'
        )

    with open(output_path, 'w') as f:
        f.write('\n'.join(lines) + '\n')
```

Listing 4.6. Automatic SDC generation.

4.6 ORFS Execution

Once configuration files were generated successfully, OpenROAD Flow Scripts were executed using the Makefile-based backend flow.

```
def run_orfs(design_name, orfs_path):

    cmd = [
        'make',
        f'DESIGN_CONFIG=./designs/sky130hd/'
        f'{design_name}/config.mk'
    ]

    result = subprocess.run(
        cmd,
        cwd=os.path.join(orfs_path, 'flow'),
        text=True
    )

    if result.returncode != 0:
        raise RuntimeError(
            f'ORFS failed for {design_name}'
        )
```

Listing 4.7. ORFS execution from OpenROAD.py.

4.7 Validation of .cir.out Files

Before launching the OpenROAD flow, validation checks were added to verify that the generated `.cir.out` file existed and contained valid SPICE data.

```
def validate_cir_out(cir_out_path):

    if not os.path.isfile(cir_out_path):
        return False, 'File not found.'

    with open(cir_out_path, 'r') as f:
        content = f.read().lower()

    if '.end' not in content:
        return False, 'Invalid .cir.out file.'

    return True, 'OK'
```

Listing 4.8. Validation of .cir.out files.

4.8 Debugging and Fixes

Several issues were encountered during implementation and testing. These issues were identified and resolved during the integration process.

4.8.1 ORFS Path Errors

Incorrect project naming initially caused ORFS configuration paths to fail. The issue was resolved by sanitizing project names before directory creation.

```
import re

design_name = re.sub(
    r'[^A-Za-z0-9_]',
    '',
    project_name.replace(' ', '_')
)
```

Listing 4.9. Project name sanitization.

4.8.2 HalfAdder Verilog Generation Issue

The generated Verilog file for the HalfAdder project was initially empty because of case-sensitive module naming differences.

The issue was resolved by extracting the module name directly from the project directory instead of parsing it from the netlist.

```
module_name = os.path.basename(project_dir)
```

Listing 4.10. Module name extraction fix.

4.8.3 DEF Loading Error in OpenROAD GUI

Initially, the OpenROAD GUI failed to load the generated DEF file correctly because LEF technology files were not loaded.

The issue was resolved by launching the GUI using the generated `.odb` database file.

```
def launch_openroad_gui(
    orfs_path,
    design_name
):

    result_dir = os.path.join(
        orfs_path,
        'flow',
        'results',
        'sky130hd',
        design_name
    )

    odb_file = os.path.join(
        result_dir,
        '6_final.odb'
    )

    cmd = [
        os.path.join(
            orfs_path,
            'tools',
            'install',
            'OpenROAD',
            'bin',
            'openroad'
        ),
        '-gui',
        odb_file
    ]

    subprocess.Popen(cmd)
```

Listing 4.11. Corrected OpenROAD GUI launch.

4.9 FullAdder and HalfAdder Validation

The complete integration flow was tested using both **FullAdder** and **HalfAdder** circuits.

The integration successfully generated:

- synthesized Verilog files,
- DEF floorplan files,

- GDSII layout files,
- timing constraint files,
- parasitic extraction outputs.

Figure 4.2 shows both projects visible inside the eSim project panel.

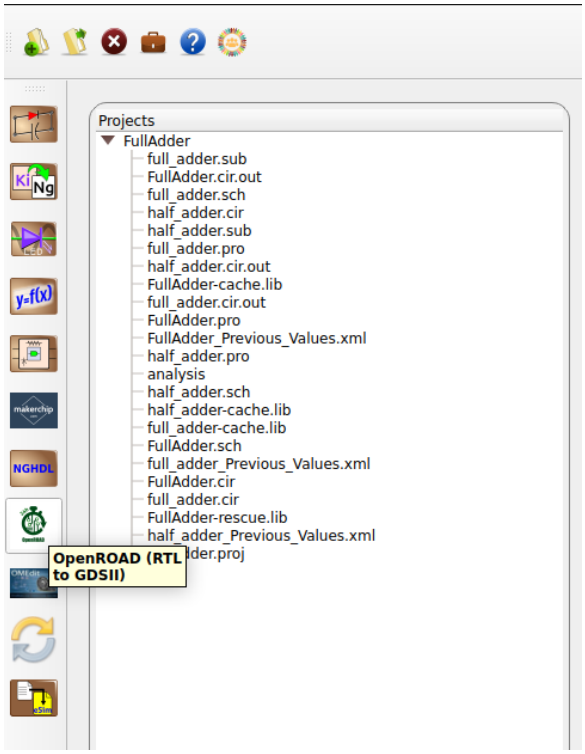

Figure 4.2. FullAdder and HalfAdder projects inside the eSim workspace.

Figure 4.3 shows the FullAdder project during OpenROAD flow execution.

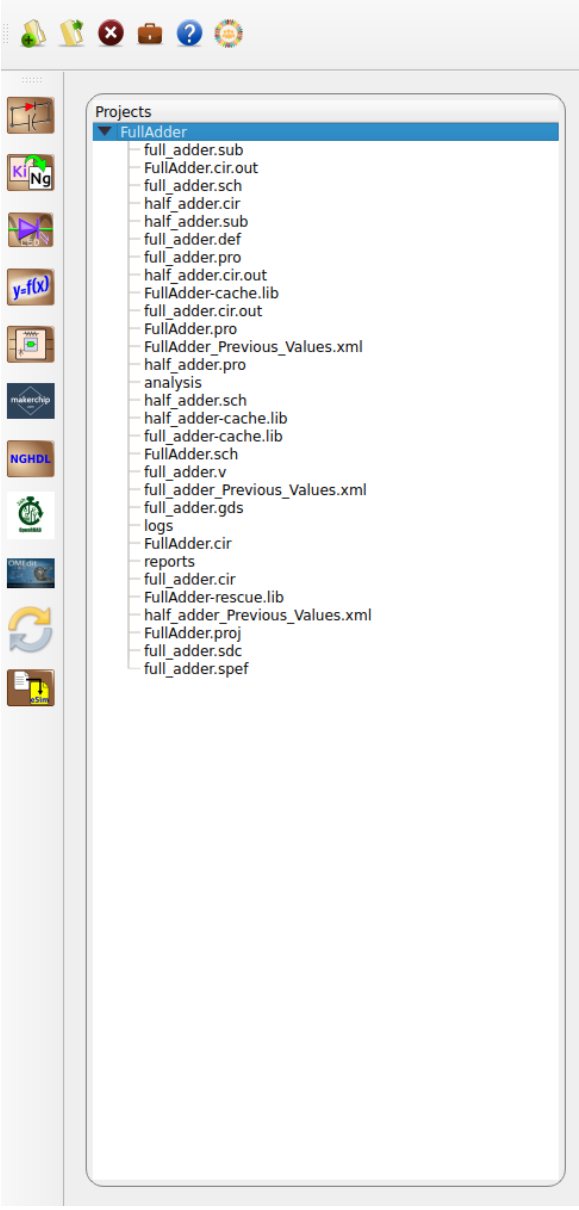

Figure 4.3. FullAdder project during OpenROAD execution.

Chapter 5

Example Circuit and Testing

5.1 Overview

Two example circuits were used to validate the complete OpenROAD integration workflow: **FullAdder** and **HalfAdder**. Both circuits were successfully processed through the complete RTL-to-GDSII backend flow and the generated outputs were verified at each implementation stage.

5.2 FullAdder Testing

5.2.1 Input: FullAdder.cir.out

The FullAdder project uses a `.cir.out` netlist generated from the KiCad schematic inside eSim. The circuit implements a 1-bit FullAdder with three input ports (`in1`, `in2`, and `cin`) and two output ports (`sum` and `cout`).

5.2.2 Generated Verilog

The `netlist2rtl.py` converter processed the generated `FullAdder.cir.out` file and automatically generated the following Verilog module:

```
module FullAdder (
    cin,
    cout,
    in1,
    in2,
    sum
);

input cin;
output cout;
input in1;
input in2;
output sum;

```

```
wire net1;
wire net2;
wire net3;
wire net4;
wire net5;

sky130_fd_sc_hd__fa_1 _0_ (
    .A(net2),
    .B(net3),
    .CIN(net1),
    .COUT(net4),
    .SUM(net5)
);

sky130_fd_sc_hd__clkdlybuf4s50_1 input1 (
    .A(cin),
    .X(net1)
);

sky130_fd_sc_hd__clkdlybuf4s50_1 input2 (
    .A(in1),
    .X(net2)
);

sky130_fd_sc_hd__clkdlybuf4s50_1 input3 (
    .A(in2),
    .X(net3)
);

sky130_fd_sc_hd__clkdlybuf4s50_1 output4 (
    .A(net4),
    .X(cout)
);

sky130_fd_sc_hd__clkdlybuf4s50_1 output5 (
    .A(net5),
    .X(sum)
);

endmodule
```

Listing 5.1. Generated FullAdder.v module.

5.2.3 Generated SDC File

The timing constraints file was automatically generated during ORFS setup.

```
current_design FullAdder

create_clock -name clk -period 10.0000

set_input_delay 0.1000 -add_delay \
```

```
[get_ports {cin}]

set_input_delay 0.1000 -add_delay \
[get_ports {in1}]

set_input_delay 0.1000 -add_delay \
[get_ports {in2}]

set_output_delay 0.1000 -add_delay \
[get_ports {cout}]

set_output_delay 0.1000 -add_delay \
[get_ports {sum}]
```

Listing 5.2. Auto-generated FullAdder constraint.sdc file.

5.2.4 Flow Completion

The OpenROAD Flow Scripts execution for the FullAdder circuit completed successfully through all backend stages including synthesis, floorplanning, placement, CTS, routing, and GDSII generation.

Figure 5.1 shows the terminal output after successful completion of the RTL-to-GDSII flow.

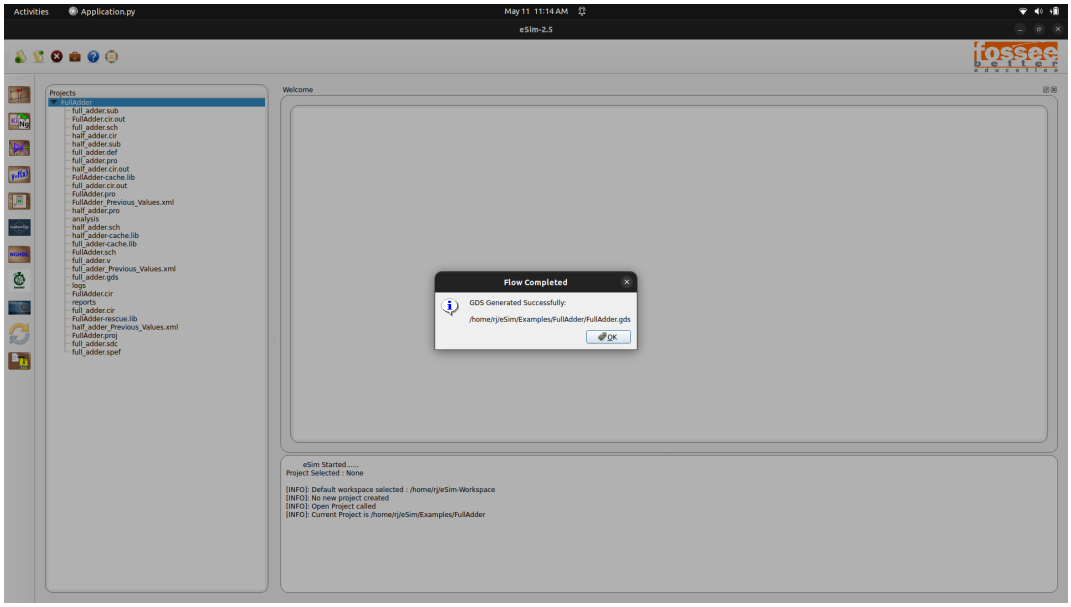

Figure 5.1. Successful RTL-to-GDSII flow completion for the FullAdder circuit.

5.2.5 Generated Output Files

After successful ORFS execution, multiple backend design files were generated automatically.

Table 5.1. Generated output files after ORFS execution.

File	Format	Description
`FullAdder.v`	Verilog	Generated RTL module
`FullAdder.def`	DEF	Placement and routing data
`FullAdder.gds`	GDSII	Final layout file
`FullAdder.sdc`	SDC	Timing constraints
`FullAdder.spef`	SPEF	Parasitic extraction output
`logs/`	Directory	Backend execution logs

5.3 OpenROAD GUI Visualization

After completion of the backend flow, the generated layout was opened automatically in the OpenROAD GUI using the generated `.odb` database file.

Figure 5.2 shows the FullAdder physical layout displayed in the OpenROAD GUI.

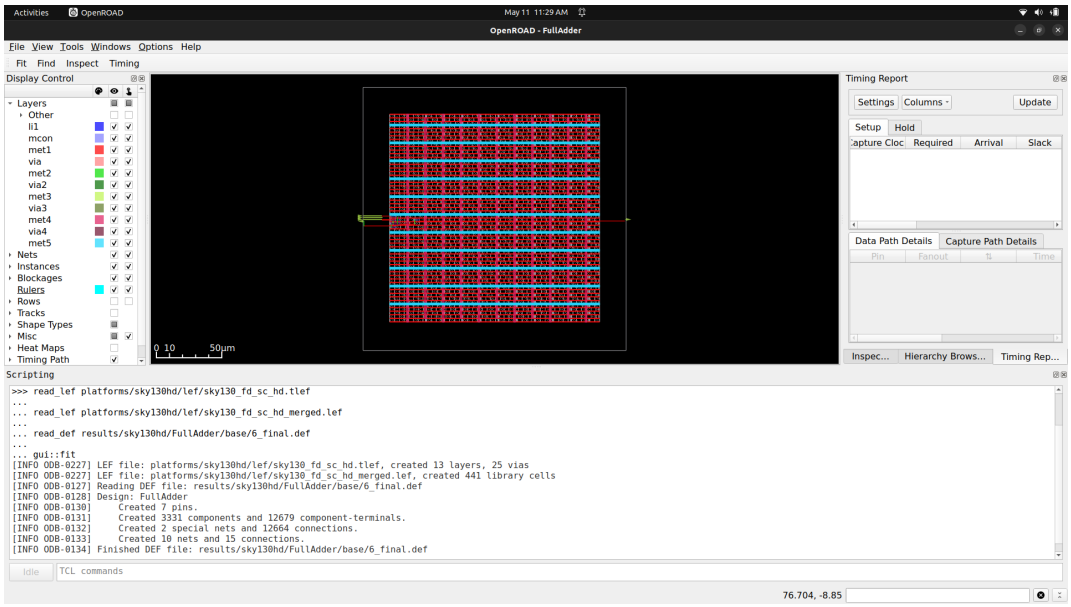

Figure 5.2. FullAdder layout visualized in the OpenROAD GUI.

5.4 KLayout Visualization

The generated GDSII layout file was also verified using KLayout.

Figure 5.3 shows the FullAdder layout loaded in KLayout for physical inspection.

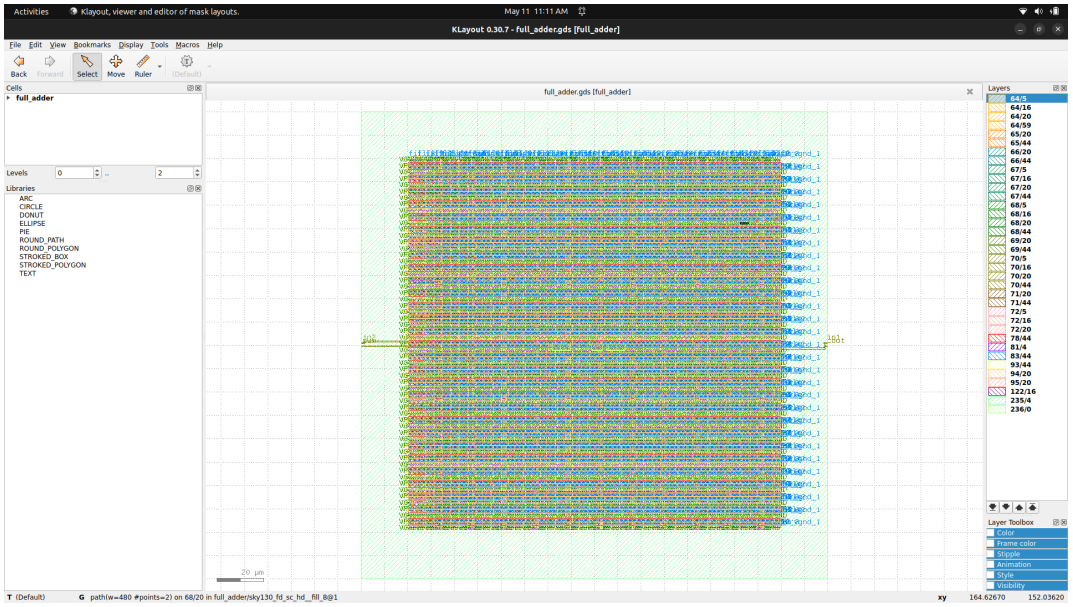

Figure 5.3. FullAdder GDS layout visualized in KLayout.

5.5 HalfAdder Testing

The HalfAdder circuit was also validated successfully using the same OpenROAD integration workflow. The generated `HalfAdder.cir.out` file was converted into Verilog using `netlist2rtl.py`, and the OpenROAD Flow Scripts execution completed successfully.

The generated backend outputs included:

- synthesized Verilog,
- DEF layout files,
- GDSII layout files,
- timing constraint files,
- ORFS execution logs.

Figure 5.4 shows both FullAdder and HalfAdder projects after successful flow completion.

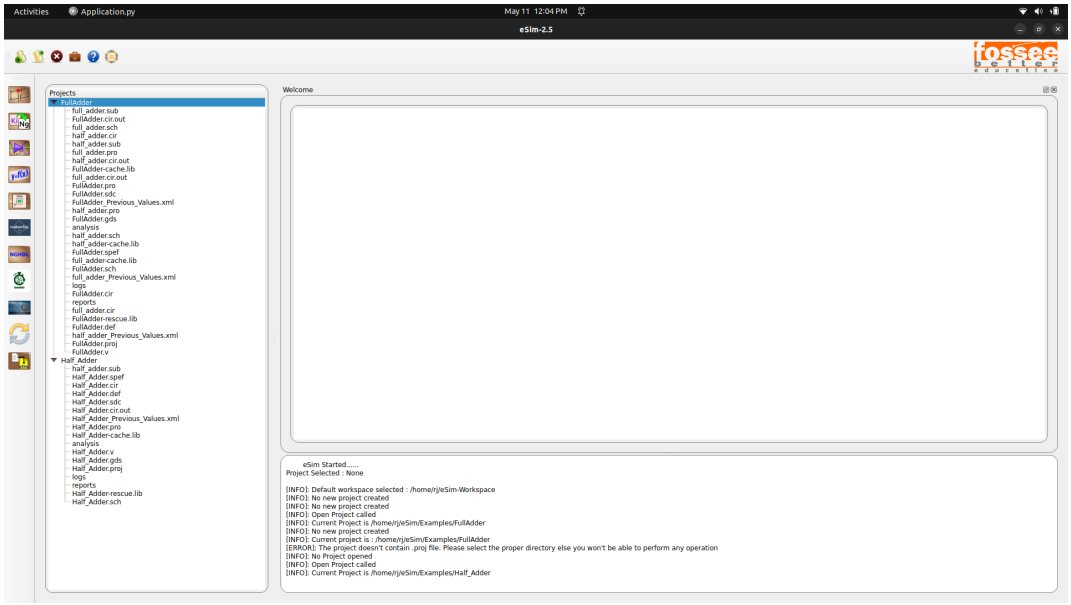

Figure 5.4. FullAdder and HalfAdder after successful RTL-to-GDSII flow execution.

5.6 Workflow Summary

The complete validated workflow implemented during the internship is summarized below:

1. Create and simulate the circuit in eSim using KiCad and Ngspice.
2. Run KiCad-to-Ngspice conversion to generate the `.cir.out` file.
3. Click the OpenROAD button integrated into the eSim toolbar.
4. Automatically generate Verilog RTL using `netlist2rtl.py`.
5. Automatically generate `config.mk` and `constraint.sdc`.
6. Execute the ORFS backend implementation flow.
7. Generate backend outputs including DEF and GDSII layouts.
8. Visualize the generated layouts using OpenROAD GUI and KLayout.

Chapter 6

Future Enhancement and Left Off

6.1 Current Status

The OpenROAD integration with eSim was successfully completed and validated using the **FullAdder** and **HalfAdder** example circuits. The implemented workflow successfully performs automatic Verilog generation, OpenROAD Flow Scripts execution, backend file generation, and layout visualization using both OpenROAD GUI and KLayout.

The RTL-to-GDSII flow was tested end-to-end and the generated backend outputs were verified successfully during implementation.

6.2 Completed Work

At the end of this internship, the following implementation tasks were completed successfully:

- OpenROAD execution button added to the eSim graphical interface.
- `netlist2rtl.py` implemented for automatic Verilog generation.
- `OpenROAD.py` backend module developed for ORFS execution.
- Automatic generation of `config.mk` and `constraint.sdc`.
- Successful RTL-to-GDSII flow execution for FullAdder and HalfAdder.
- Automatic OpenROAD GUI launch after flow completion.
- Successful GDSII visualization using KLayout.
- Multiple backend integration and execution issues resolved.

6.3 Support for Additional Circuit Types

The current implementation primarily supports combinational circuits such as FullAdder and HalfAdder. Future work can extend the parser to support:

- sequential circuits,
- flip-flop based designs,
- hierarchical netlists,
- larger digital modules.

Additional improvements may also be required for handling more complex SPICE netlists generated from larger projects.

6.4 Improved RTL Parsing

The current `netlist2rtl.py` module uses a simplified parsing approach for extracting ports and subcircuit instances from the generated `.cir.out` files.

Future improvements may include:

- improved port identification,
- better wire extraction,
- support for complex net connections,
- automatic detection of input and output signals.

A more robust parser would improve compatibility with larger and more complex circuit designs.

6.5 Improved GUI Integration

The current OpenROAD flow executes as a backend subprocess launched directly from eSim.

Future improvements may include:

- real-time progress display,
- flow execution logs inside the GUI,
- flow cancellation support,
- automatic error reporting,
- generated file summaries after execution.

These features would improve the overall usability of the integration.

6.6 Automatic File Management

Currently, generated backend outputs such as `.def`, `.gds`, and `.spef` files are stored inside the ORFS results directory.

Future improvements may include:

- automatic copying of generated files into the eSim project directory,
- automatic workspace refresh,
- organized output management for multiple projects.

This would simplify project management and improve integration consistency.

6.7 Enhanced Layout Visualization

The generated layouts are currently visualized using external OpenROAD GUI and KLayout applications.

Future improvements may include direct integration of layout visualization inside the eSim graphical interface, allowing users to inspect generated layouts without leaving the eSim environment.

Such integration would provide a more unified open-source RTL-to-GDSII workflow within a single application.

Chapter 7

Conclusion

7.1 Summary

This internship successfully implemented the integration of the **OpenROAD** digital physical design flow into the **eSim** open-source EDA environment. The completed workflow connects eSim circuit simulation with the OpenROAD RTL-to-GDSII backend implementation flow entirely within an open-source ecosystem.

The major implementation tasks completed during this internship include:

1. Integration of an OpenROAD execution button into the eSim graphical interface through modifications in `Application.py`.
2. Development of `netlist2rtl.py` for automatic Verilog generation from eSim-generated `.cir.out` netlists.
3. Development of `OpenROAD.py` for OpenROAD Flow Scripts (ORFS) execution and backend automation.
4. Automatic generation of `config.mk` and `constraint.sdc` files required for ORFS execution.
5. Successful RTL-to-GDSII flow validation for both FullAdder and HalfAdder circuits.
6. Generation of backend outputs including Verilog, DEF, GDSII, SDC, and SPEF files.
7. Successful layout visualization using OpenROAD GUI and KLayout.
8. Identification and resolution of multiple backend integration and execution issues during implementation.

7.2 Learning Outcomes

This internship provided practical exposure to:

- eSim frontend and backend architecture,
- Python and PyQt5 GUI development,
- OpenROAD Flow Scripts (ORFS),
- RTL synthesis using Yosys,
- physical design flow execution,
- GDSII layout visualization using KLayout,
- debugging and integration of open-source EDA tools.

The project also provided hands-on experience with backend automation, flow execution management, and open-source digital design workflows.

7.3 Final Remarks

The completed integration demonstrates that a complete open-source RTL-to-GDSII workflow can be executed directly from the eSim environment. The successful implementation of FullAdder and HalfAdder backend flows confirms that OpenROAD can be effectively integrated into eSim for digital physical design automation.

This integration extends the capabilities of eSim beyond circuit simulation by enabling backend physical design generation within a unified open-source environment.

Figure 7.1 shows the successful RTL-to-GDSII flow completion for the FullAdder circuit.

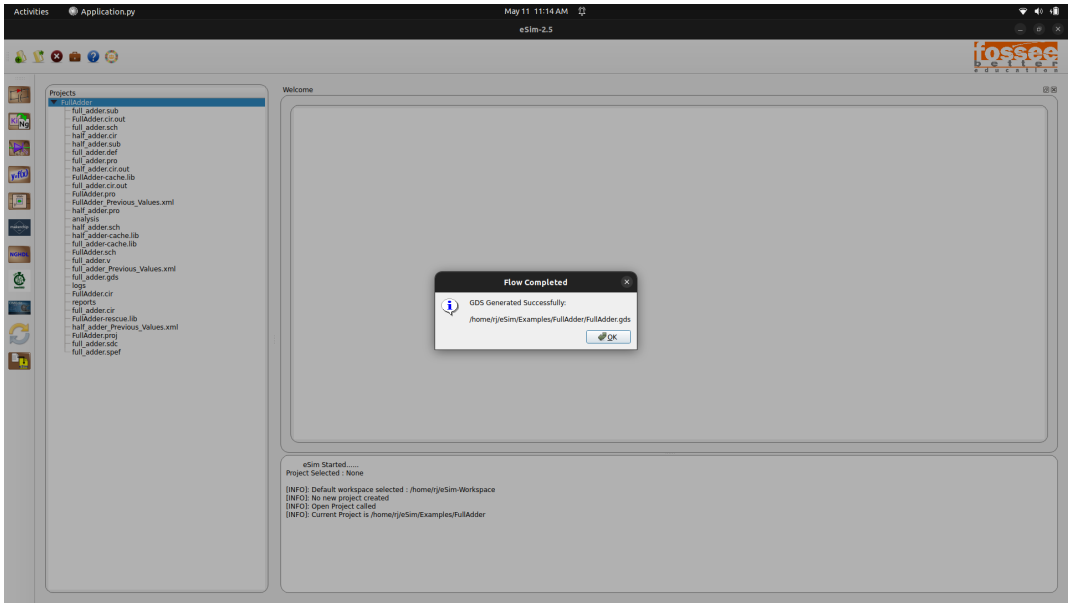

Figure 7.1. Successful RTL-to-GDSII flow completion for the FullAdder circuit.

References

1. FOSSEE, *eSim Repository*, GitHub Repository.
   `https://github.com/FOSSEE/eSim`

2. The OpenROAD Project, *OpenROAD-flow-scripts*, GitHub Repository.
   `https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts`

3. The OpenROAD Project, *OpenROAD Repository*, GitHub Repository.
   `https://github.com/The-OpenROAD-Project/OpenROAD`

4. The OpenROAD Project, *OpenROAD Official Website*.
   `https://theopenroadproject.org/`

5. The OpenROAD Project, *OpenROAD Documentation*.
   `https://openroad.readthedocs.io/en/latest/`

6. FOSSEE, *eSim Official Website*.
   `https://esim.fossee.in/home`

7. FOSSEE, *eSim-to-OpenROAD Design Flow Plugin*, GitHub Repository.
   `https://github.com/FOSSEE/eSim-to-OpenROAD_Design_Flow_Plugin`

8. Rishabh Jain, *eSim OpenROAD Plugin*, GitHub Repository.
   `https://github.com/RISHABH12005/eSim-OpenROAD-Plugin`

9. Rishabh Jain, *eSim Repository Fork*, GitHub Repository.
   `https://github.com/RISHABH12005/eSim`