EE497 - Undergraduate Project
Development of an Open-Source RTL-to-GDS Digital Implementation Flow for Instructional Use in EE619

Rachit Jain (220846)
B.Tech, Department of Electrical Engineering
Indian Institute of Technology Kanpur
rachitjain22@iitk.ac.in

***Abstract*—This report presents the development and evaluation of an open-source digital implementation flow suitable for instructional use in the lab sessions of the course EE619. The flow integrates Yosys for logic synthesis, OpenSTA for timing and power analysis, and OpenROAD for physical design, using the Nangate open cell library. The complete toolchain was packaged in a Docker environment to enable easy deployment on lab machines and ensure reproducibility.**

I. Introduction

Digital integrated circuits form the computational backbone of almost every modern electronic system, supporting applications that range from embedded controllers and communication devices to larger computing platforms. As semiconductor technologies continue to advance and design demands grow, engineering students learning digital IC design require not just theoretical understanding but also hands-on experience with the full digital implementation flow. Lab sessions in VLSI courses, such as EE619, are therefore essential, as they provide students with practical exposure to key design stages, including RTL modelling, synthesis, timing analysis, and power estimation.

However, commercial EDA tools and foundry PDKs—which are typically used in industry—are expensive, require strict NDAs, and often cannot be freely distributed or installed in classroom environments. This limits the ability to conduct open and flexible lab work as part of an academic course. In recent years, open-source EDA tools and publicly available PDKs have matured significantly, making it increasingly feasible to build practical digital design flows without relying on proprietary infrastructure. This semester-long project aims to develop such an accessible, open-source-based flow tailored for instructional use in digital design courses.

II. Methodology

The work covered RTL design and coding, logic synthesis using the open-source framework Yosys[1] with the Nangate45[2] open cell library, and a brief comparison with timing-driven synthesis in commercial tools such as Cadence Genus. Post-synthesis timing and power analysis were performed using the open-source engine OpenSTA[3], followed by preliminary exploration of physical design through OpenROAD[4]. Finally, all tools and libraries were set up within a Docker-based environment to enable easy installation and deployment for course lab sessions.

A. Logic Synthesis using Yosys

Logic synthesis is the process of translating an RTL hardware description into a gate-level netlist using a chosen standard-cell library. A standard-cell library provides the technology-specific building blocks—such as NAND gates, inverters, and flip-flops—along with their timing, power, and functional models. During synthesis, the tool interprets the Verilog constructs, applies structural and logic optimizations, and maps the high-level behavior to these technology-defined cells. The process also accounts for user-specified constraints such as input transition times, output loads, and clock definitions, which affect timing paths and cell selection. The industry-standard tools for RTL synthesis include Cadence Genus and Synopsys Design Compiler, which offer advanced timing-driven optimization capabilities.

Yosys is an open-source framework for Verilog synthesis and verification. It supports all commonly used synthesizable features of Verilog-2005 and is widely used in academic research and commercial applications. Yosys uses Berkeley ABC for combinational logic minimization and fine-grain technology mapping[5]. It enables the designer to inspect intermediate representations and optimization passes, and also view the synthesized netlist as a schematic-style graph, making the mapping process much easier to understand.

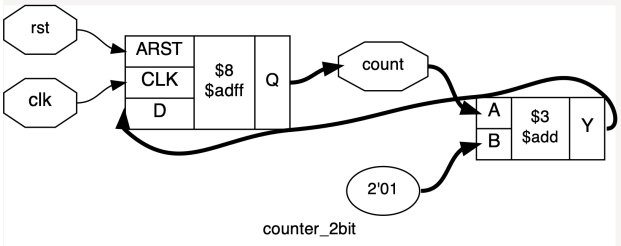

Fig. 1. Yosys generated coarse-grain netlist of a 2-bit counter.

Fig. 1 shows the high-level schematic of a simple 2-bit counter obtained after the initial Yosys front-end processing. At this stage, the design is represented using Yosys's abstract coarse-grain cell types like adders and multipliers, and has not yet been mapped to any technology library. After this,

the design is mapped to the Nangate45 standard-cell library, producing the technology-specific gate-level netlist shown in Fig. 2.

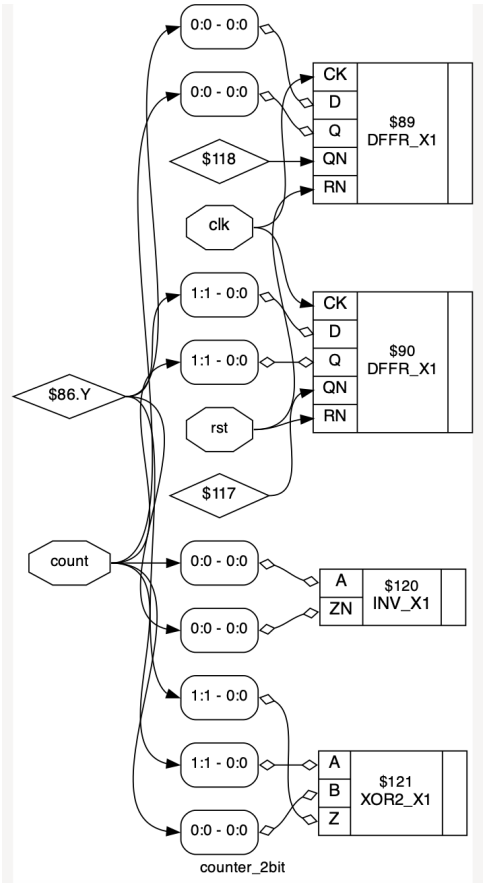

Fig. 2. Gate-level netlist of the 2-bit counter.

1) Limitations of Yosys Compared to Commercial Tools: While Yosys is well-suited for instructional use, it has several limitations when compared to commercial tools, such as Cadence Genus. Its optimization engines are relatively simple, which can lead to netlists that are not as well optimized for area or timing under tight constraints. In addition, Yosys has limited support for handling advanced timing constraints, and its mapping decisions are generally based on functional equivalence rather than advanced timing-driven optimization.

```
module timing_constrained_synthesis (clk,A,B,C,D,F);
  input clk, A, B, C, D;
  output reg F;
  reg A_r, B_r;
  always @(posedge clk) begin
    A_r <= A;
    B_r <= B;
    F <= A_r & B_r & C & D;
  end
endmodule
```

To illustrate this, a simple design (shown below) was synthesized under a high clock-frequency constraint to enforce tight timing requirements. Yosys performs a primarily logic-equivalent mapping, selecting cells based on functional correctness rather than timing pressure, and does not apply timing-driven optimizations such as gate resizing or critical path restructuring. As a result, the Yosys-generated netlist typically fails to meet the timing constraints.

```
module timing_contrained_synthesis(clk, A, B, C, D,
    F); // Yosys-generated netlist
  input clk, A, B, C, D;
  output F;
  wire clk, A, B, C, D;
  wire F;
  wire _0_, _1_, _2_, _3_, A_r, B_r;
  AND4_X4 _4_(.A1(A_r), .A2(B_r), .A3(C), .A4(D), .
      ZN(_0_));
  DFF_X1 _5_(.CK(clk), .D(_0_), .Q(F), .QN(_2_));
  DFF_X1 _6_(.CK(clk), .D(A), .Q(A_r), .QN(_3_));
  DFF_X1 _7_(.CK(clk), .D(B), .Q(B_r), .QN(_1_));
endmodule
```

In contrast, Genus uses a full timing-driven synthesis flow. When provided with the same constraints, it analyzes critical paths, restructures the logic and selects faster or higher-drive strength cells from the library to meet the required timing, producing an optimized gate-level netlist as shown below.

```
module timing_contrained_synthesis(clk, A, B, C, D,
    F); // Genus-generated netlist
  input clk, A, B, C, D;
  output F;
  wire clk, A, B, C, D;
  wire F;
  wire A_r, B_r, UNCONNECTED, UNCONNECTED0,
      UNCONNECTED1, n_0, n_1;
  DFF_X1 F_reg(.CK (clk), .D (n_1), .Q (F), .QN (
      UNCONNECTED));
  NOR3_X1 g64__2398(.A1 (B_r), .A2 (A_r), .A3 (n_0),
       .ZN (n_1));
  DFF_X1 A_r_reg(.CK (clk), .D (A), .Q (UNCONNECTED0
      ), .QN (A_r));
  NAND2_X1 g67__5107(.A1 (D), .A2 (C), .ZN (n_0));
  DFF_X1 B_r_reg(.CK (clk), .D (B), .Q (UNCONNECTED1
      ), .QN (B_r));
endmodule
```

B. Static Timing and Power Analysis with OpenSTA

After synthesis, the next step in the flow is to perform timing and power analysis on the synthesized netlist to verify whether the design meets its performance targets. This is done using OpenSTA, an open-source static timing analyzer that supports Verilog netlists, Liberty (.lib) models, and SDC constraints.

OpenSTA identifies critical paths and computes arrival times, required times, and setup and hold slacks using library cell delays and user-specified input transition and output load conditions. By reading power values from the Liberty file, it also provides estimates of internal, switching, and leakage power. This provides students with an accessible way to understand how constraints impact cell choices, path delays, and the overall design feasibility.

Fig 3. shows a sample timing report generated by OpenSTA. It lists the critical paths in the design along with their arrival times, required times, and resulting setup or hold slacks, and tells whether the design meets the timing requirements.

Fig. 4 illustrates a sample power report produced by OpenSTA. Using the power values defined in the technology library,

```
% report_checks -path_delay max -digits 4
Startpoint: _5_ (rising edge-triggered flip-flop clocked by CLK)
Endpoint: _6_ (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

    Delay      Time   Description
---------------------------------------------------------
   0.0000    0.0000   clock CLK (rise edge)
   0.0000    0.0000   clock network delay (ideal)
   0.0000    0.0000 ^ _5_/CK (DFFR_X1)
   0.1278    0.1278 ^ _5_/Q (DFFR_X1)
   0.0483    0.1761 ^ _4_/Z (XOR2_X1)
   0.0000    0.1761 ^ _6_/D (DFFR_X1)
             0.1761   data arrival time

   0.5000    0.5000   clock CLK (rise edge)
   0.0000    0.5000   clock network delay (ideal)
   0.0000    0.5000   clock reconvergence pessimism
             0.5000 ^ _6_/CK (DFFR_X1)
  -0.0358    0.4642   library setup time
             0.4642   data required time
---------------------------------------------------------
             0.4642   data required time
            -0.1761   data arrival time
---------------------------------------------------------
             0.2881   slack (MET)
```

Fig. 3. Sample timing report generated by OpenSTA.

OpenSTA breaks down total power into internal, switching, and leakage components.

```
% report_power -digits 4
Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power (Watts)
----------------------------------------------------------------
Sequential           2.2084e-05 4.9924e-05 1.7243e-07 7.2180e-05  95.0%
Combinational        3.1966e-06 5.4609e-07 5.0517e-08 3.7932e-06   5.0%
Clock                0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00   0.0%
Macro                0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00   0.0%
Pad                  0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00   0.0%
----------------------------------------------------------------
Total                2.5281e-05 5.0470e-05 2.2294e-07 7.5973e-05 100.0%
                         33.3%      66.4%       0.3%
```

Fig. 4. Sample power report generated by OpenSTA.

The power dissipation of a circuit falls into two broad categories: static power and dynamic power[6]. Static power is the power dissipated by a gate when it is not switching—that is, when it is inactive or static. It primarily results from source-to-drain subthreshold leakage and junction leakage between the diffusion regions and the substrate. For this reason, static power is often referred to as leakage power. Dynamic power, on the other hand, is the power dissipated when a circuit is active and is composed of internal power and switching power. Internal power includes power dissipated by a momentary short circuit between the P and N transistors of a gate, as well as the charging and discharging of any capacitance internal to the cell. The short-circuit current largely depends on the input and output transition times. Switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. It is a function of both the total load capacitance and the rate of logic transitions. Fig. 5 illustrates these different components of power dissipation.

III. Future Work

Although the broader goal of this project was to build a complete open-source RTL-to-GDS digital implementation

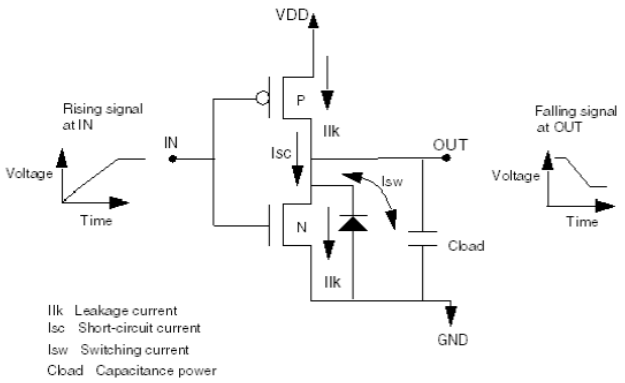

Fig. 5. Components of Power Dissipation.

flow, the physical design stage could not be fully completed within the project timeline. The work completed primarily focused on RTL design, synthesis using Yosys, and static timing and power analysis using OpenSTA. Physical design was explored only at a preliminary level using the open-source tool OpenROAD, using test designs and default flow scripts. Fig. 6 shows the layout of a gcd test design generated by the tool. A detailed study of the physical design stages—such as floorplanning, placement, routing, and parasitic extraction—along with a deeper understanding of the individual tool commands and the development of custom physical design flow remains to be done.

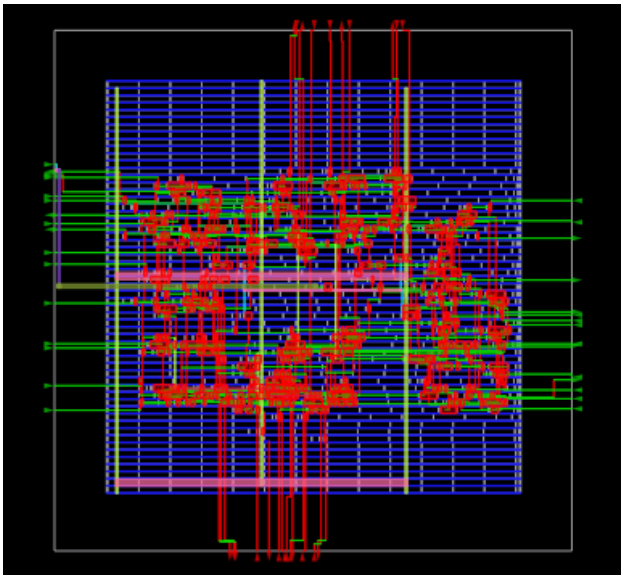

Fig. 6. Layout of the gcd test design generated by OpenROAD.

IV. Results and Discussion

This project explored the construction of a digital implementation flow using open-source tools such as Yosys, OpenSTA, and OpenROAD-tools that have gained significant maturity in recent years as academic and research alternatives to commercial EDA environments.

Synthesis experiments were performed using Yosys with the open Nangate45 standard cell library, illustrating the transformation from RTL to gate-level netlists. Timing and power analysis using OpenSTA gave insights into the critical paths, slack margins, and distribution of internal, switching, and leakage power in the synthesized design. Preliminary physical design exploration using OpenROAD showed the potential of a fully open-source backend. Finally, the complete toolchain, along with the required libraries, was packaged into a Docker environment to ensure reproducible installation and to simplify deployment on lab machines.

V. Conclusion

The rise of open-source Electronic Design Automation (EDA) tools, such as OpenROAD, marks a significant shift toward democratizing hardware design and fostering innovation, even though commercial tools from vendors like Cadence and Synopsys continue to be the industry standard for high-end, production-level chip manufacturing.

VI. Acknowledgment

I sincerely thank Dr Chithra for introducing me to open-source digital design methodologies, for assigning me this project, and for her invaluable guidance throughout its execution. This project and the lessons learned from it would not have been possible without her support and encouragement. I would also like to acknowledge the open-source EDA communities behind Yosys, OpenSTA, and OpenROAD for providing the tools and documentation that made this work possible.

References

[1] Wolf, C., "Yosys open synthesis suite," https://github.com/yosyshq/yosys.
[2] Nangate and Silicon Integration Initiative (Si2), "Nangate 45nm open cell library," https://si2.org/open-cell-library/.
[3] Cherry, J., "Opensta: Static timing analyzer," https://github.com/parallaxsw/OpenSTA.
[4] TheOpenROADProject, "Openroad: Open-source physical design tool," https://github.com/The-OpenROAD-Project/OpenROAD.
[5] Wolf, C. and Glaser, J., "Yosys - a free verilog synthesis suite," in *Proc. 21st Austrian Workshop Microelectron. (Austrochip)*, 2013.
[6] Pei, Z., "Modeling power terminology," https://blogs.cuit.columbia.edu/zp2130/modeling_power_terminology/.