UNDERGRADUATE PROJECT

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

Presented by Rachit Jain

Supervisor: Dr. Chithra

Overview

- Introduction
- Methodology
  - Logic Synthesis using Yosys
  - Timing and Power analysis using OpenSTA
  - Physical design using OpenROAD
- Conclusion
- References

Introduction

- Digital integrated circuits are at the core of all modern electronic systems.
- Students learning digital IC design need hands-on experience with the full implementation flow.
- However, the high cost and NDA restrictions of commercial EDA tools and PDKs make them difficult to deploy in academic settings.
- Recent advancements in open-source tools and publicly available PDKs have now made it possible to create accessible digital design flows for educational purposes.

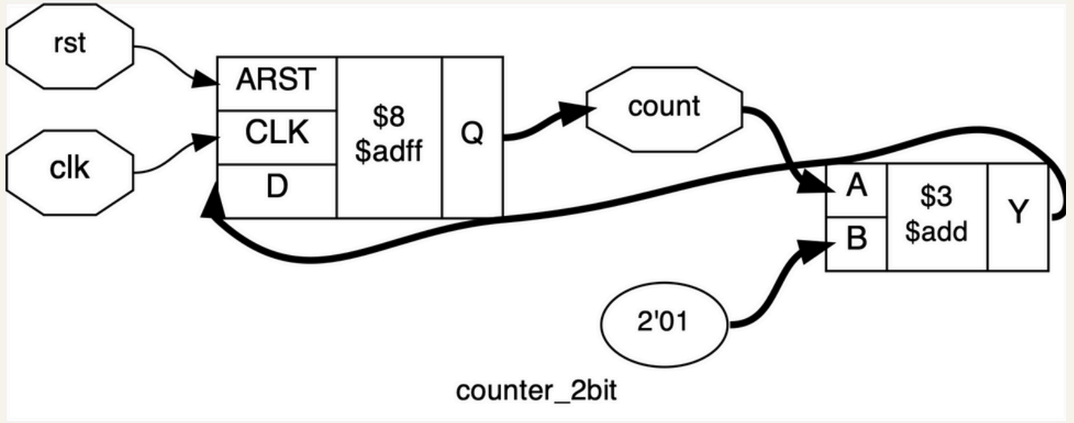
Methodology

- Began with RTL design and coding, followed by logic synthesis using Yosys with the Nangate45 open cell library.
- Conducted a comparison with Cadence Genus to highlight the stronger timing-driven optimization capabilities of commercial tool.
- Performed post-synthesis timing and power analysis using OpenSTA.
- Explored physical design at an initial level using OpenROAD, using test designs and default flow scripts.
- Finally, packaged the entire toolchain and libraries into a Docker environment to enable easy deployment on lab machines.

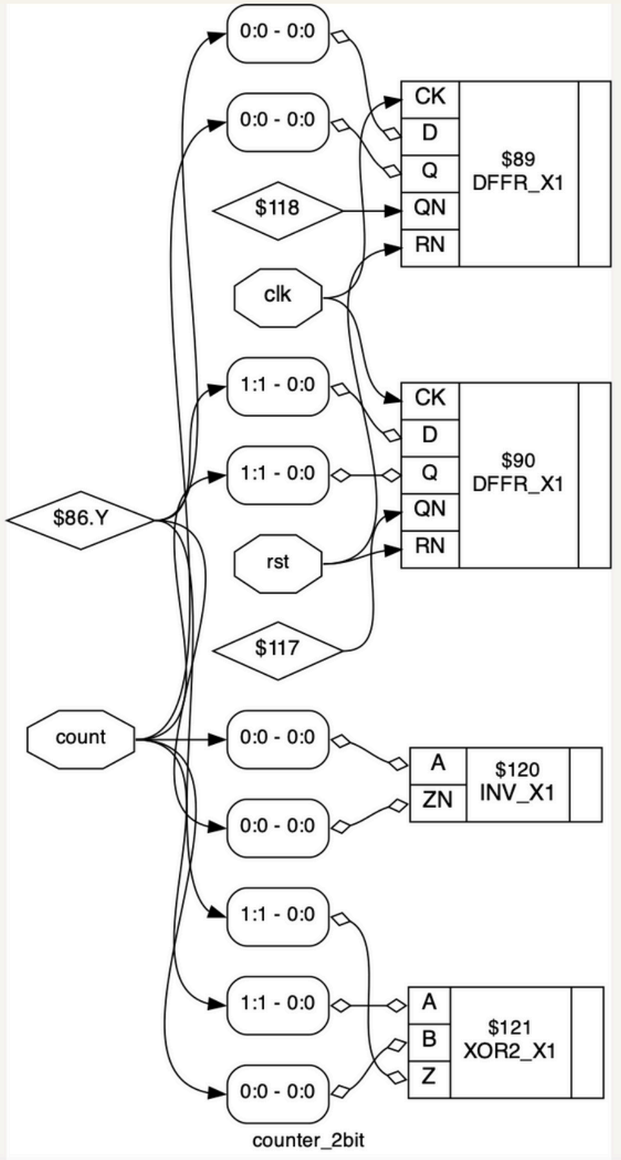
Logic Synthesis using Yosys

- Logic synthesis converts RTL hardware description into a gate-level netlist using a standard-cell library.
- Standard cell libraries provide technology-specific building blocks, such as logic gates and flip flops, along with their timing, power, and functional models.
- The synthesis tool interprets the Verilog constructs, applies structural and logic optimizations, and maps the high-level behavior to these technology-defined cells.
- Yosys is an open-source framework for Verilog synthesis, that is now widely used in academia and hobbyist projects.

- The designer can visualize the generated netlist as a schematic-style graph, making the mapping process easier to understand.
- Schematic of a 2-bit counter obtained after initial Yosys front-end processing.
- 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.



- Then, the design is mapped to the Nangate 45nm standard-cell library, producing the technology-specific gate-level netlist as shown alongside.



Limitations of Yosys

- Yosys has several limitations when compared to commercial tools like Cadence Genus.
- Its optimization engines are simpler, and its mapping decisions are generally based on functional equivalence rather than advanced timing-driven optimization.
- It has limited support for complex timing constraints. As a result, Yosys may fail to meet aggressive clock requirements that commercial tools can satisfy using logic restructuring

```
module timing_contrained_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
```

A simple Verilog design to showcase the limitations of Yosys

Yosys-generated netlist

```
module timing_contrained_synthesis(clk, A, B, C, D, F);
  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
```

Genus-generated netlist

```
module timing_contrained_synthesis(clk, A, B, C, D, F);
  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
```

OpenSTA: Timing and Power Analysis

- After synthesis, the flow proceeds to timing and power analysis to verify whether the design meets its performance targets.
- OpenSTA is an open-source static timing analyzer that supports Verilog netlists, Liberty (.lib) models, and SDC constraints.
- It identifies critical paths and computes arrival times, required times, and setup/hold slacks based on library cell delays, input transition times, and output loads.
- It can also provide estimates of internal, switching, and leakage power consumption of the design.

```
% 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%
```

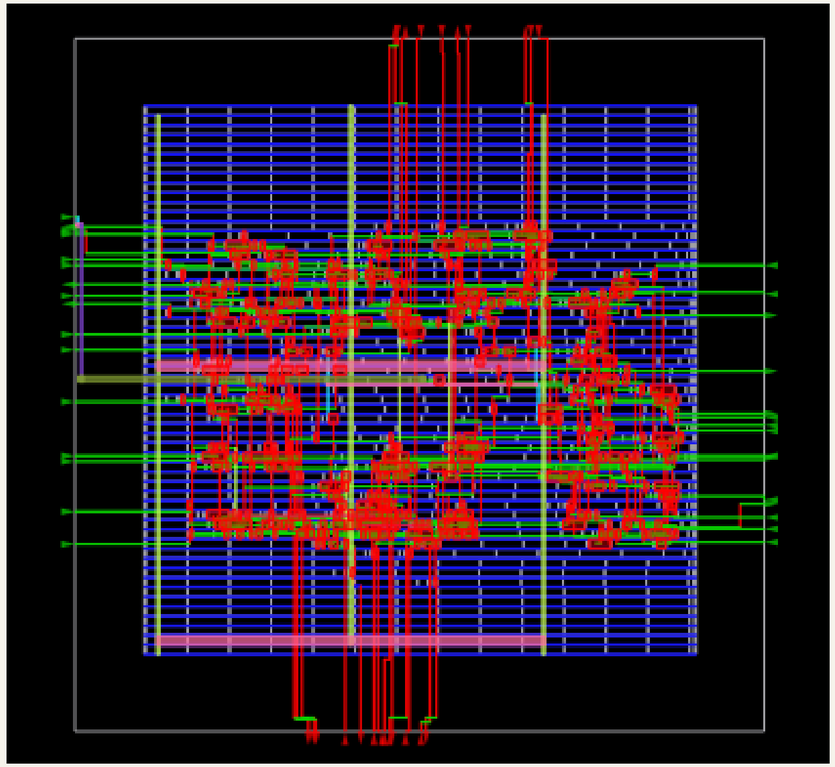
A sample power report generated by OpenSTA

```
% 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)
```

A sample timing report generated by OpenSTA

OpenROAD

- Physical design was explored only at an initial level using OpenROAD, a open-source physical design tool.
- A layout of a gcd test design generated using default flow scripts is shown alongside.
- Further work is required to perform a detailed study of the physical-design stages, such as floorplanning, placement, routing, parasitic extraction, and to gain a deeper understanding of individual tool commands in order to develop a complete physical-design flow.



Layout of the gcd test design generated by OpenROAD

Conclusion

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

- However, commercial tools from vendors like Cadence and Synopsys continue to be the industry standard for high-end, production-level chip manufacturing.

References

- C. Wolf, “Yosys Open Synthesis Suite,” GitHub: https://github.com/yosyshq/yosys
- Nangate and Si2, “Nangate 45nm Open Cell Library,” https://si2.org/open-cell-library/
- J. Cherry, “OpenSTA: Static Timing Analyzer,” GitHub: https://github.com/parallaxsw/OpenSTA
- The OpenROAD Project, “OpenROAD: Open-Source Physical Design Tool,” https://github.com/The-OpenROAD-Project/OpenROAD
- C. Wolf and J. Glaser, “Yosys — A Free Verilog Synthesis Suite,” in Proc. 21st Austrian Workshop Microelectron. (Austrochip), 2013
- Z. Pei, “Modeling Power Terminology,” Columbia University Blog: https://blogs.cuit.columbia.edu/zp2130/modeling_power_terminology/

THANK YOU

For your attention