

# ECE 429 Laboratory 9: Standard Cell Based ASIC Design Flow

Alexander Lukens  
Illinois Institute of Technology

November 23, 2020

Acknowledgment: I acknowledge all of the work (including figures and codes) belongs to me and/or persons who are referenced.

Signature: *Alexander Lukens*

# 1 Introduction

In this laboratory, students will use the ASIC design flow to automate the design of an 8-bit accumulator design. Students will first design the Verilog model for an accumulator and create a testbench to verify that the design functions as expected. Then, students will simulate the result using Cadence Verilog-XL, and view the output waveforms in SimVision. Once functionality of the original verilog file is affirmed, students will utilize Synopsys Design Compiler to perform logic synthesis on the verilog file. The design compiler will create several reports for the synthesized design, such as power, area, and timing information. This synthesized design will be completely implemented using standard cells available in the utilized process library. A post-simulation simulation should be performed to assert that the output design functions identically to the original Verilog file.

After the synthesized design is confirmed to function correctly, the design will be automatically placed and routed using Cadence Encounter. This program will automate the physical logic placement of the design (as done in previous laboratory sessions) and output a final timing report, a 'final.v' Verilog file, and placement information in GDSII format. This placement information can be used to fully realize the VLSI design in Cadence Virtuoso. Using the process library containing the standard cells, the complete layout of the design can be produced.

## 2 Theory\Pre-lab

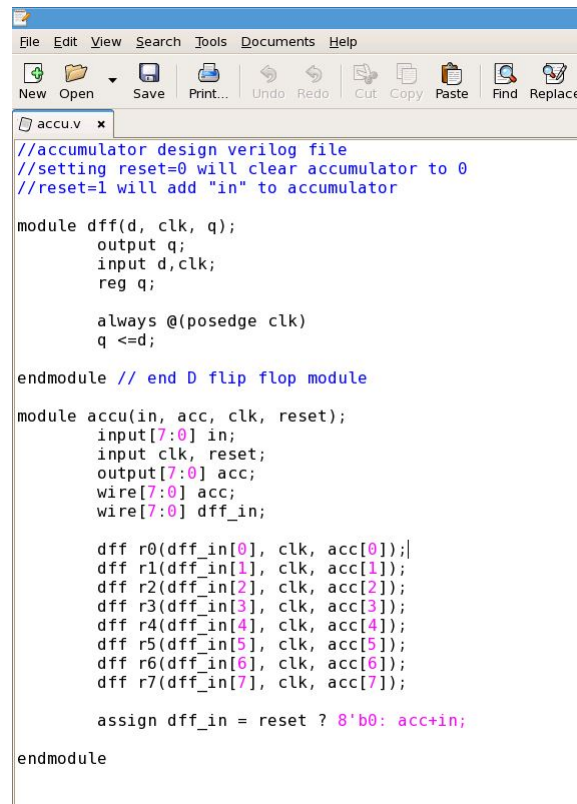
CMOS designs use numerous design “masks” to define the various separate regions that make up the physical design. These designs can be very complex, so Electronic Design Automation (EDA) software is utilized to create a physical layout for the circuit. One such EDA software suite is Cadence Virtuoso. In Cadence Virtuoso, each of the individual layers can be modeled and tested against the physical design rules for a certain process, reducing the time required to design an integrated circuit dramatically.

Inside Cadence Virtuoso, there are a variety of tools available to further decrease the probability of design errors and overall design time. The first tool is design rule checking (DRC). This allows for the physical layout of a circuit to be checked for compliance with the design rules of a certain process (on which the circuit will eventually be produced). This ensures that when the circuit is created, it will not fail due to physical spacing errors. Another tool is Layout vs Schematic (LVS) testing. LVS testing allows the physical layout produced in Cadence Virtuoso to be compared with a schematic cellview to ensure that they will function identically. This means that the overall design can first be created as a schematic (to first ensure the digital logic design functions correctly) before spending time designing the physical layout.

An important process for automating the creation of digital logic circuits is RTL-to-GDSII flow, or 'ASIC' design flow. Using this flow, a design is first created using the Verilog Hardware Design Language (HDL). In this Verilog file, all of the functionality of the intended design will be created. The functionality of this circuit should first be affirmed using an additional Verilog file as a testbench (used to provide stimuli to the main Verilog circuit). This Verilog file can then be run through a logic synthesis program that converts the RTL level Verilog file originally created into a Verilog file that is completely in terms of a standard cell library. This logic synthesis step prepares the file to be placed and routed, and provides insight into the power, area, and delay of the finalized design. After the completion of logic synthesis, the synthesized design should again be tested using the Verilog testbench in Cadence Formality or Cadence Verilog-XL. This ensures that the synthesized design performs identically to the original Verilog design. If the synthesized design passes all design criteria, the next step is to place and route the design. This adds additional specificity to the design (adds physical interconnect between the standard cells) and provides in-depth timing, area, and power calculations. If the routed design again satisfies all constraints, the design can be imported into Cadence Virtuoso using the produced GDSII file, and the complete layout checked.

### 3 Implementation

The first step in implementing an 8-bit accumulator using the ASIC design flow is to create the base Verilog implementation of the accumulator design.



```
//accumulator design verilog file
//setting reset=0 will clear accumulator to 0
//reset=1 will add "in" to accumulator

module dff(d, clk, q);
    output q;
    input d,clk;
    reg q;

    always @(posedge clk)
        q <=d;

endmodule // end D flip flop module

module accu(in, acc, clk, reset);
    input[7:0] in;
    input clk, reset;
    output[7:0] acc;
    wire[7:0] acc;
    wire[7:0] dff_in;

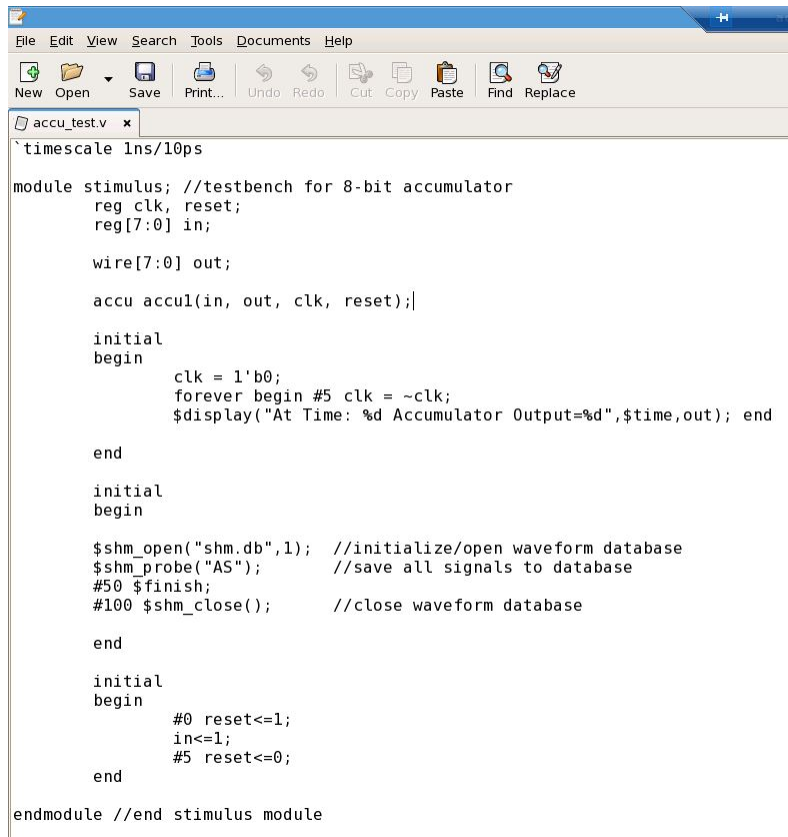
    dff r0(dff_in[0], clk, acc[0]);
    dff r1(dff_in[1], clk, acc[1]);
    dff r2(dff_in[2], clk, acc[2]);
    dff r3(dff_in[3], clk, acc[3]);
    dff r4(dff_in[4], clk, acc[4]);
    dff r5(dff_in[5], clk, acc[5]);
    dff r6(dff_in[6], clk, acc[6]);
    dff r7(dff_in[7], clk, acc[7]);

    assign dff_in = reset ? 8'b0: acc+in;

endmodule
```

Figure 1: Accumulator Base Verilog Design

A testbench file is also needed to verify the functionality of the accumulator verilog design.



```

timescale 1ns/10ps

module stimulus; //testbench for 8-bit accumulator
    reg clk, reset;
    reg[7:0] in;

    wire[7:0] out;

    accu accu1(in, out, clk, reset);

    initial
    begin
        clk = 1'b0;
        forever begin #5 clk = ~clk;
        $display("At Time: %d Accumulator Output=%d", $time, out); end

    end

    initial
    begin

        $shm_open("shm.db", 1); //initialize/open waveform database
        $shm_probe("AS"); //save all signals to database
        #50 $finish;
        #100 $shm_close(); //close waveform database

    end

    initial
    begin
        #0 reset<=1;
        in<=1;
        #5 reset<=0;

    end

endmodule //end stimulus module

```

Figure 2: Accumulator Verilog Testbench

The Cadence Verilog-XL utility is used to test the functionality of the Verilog design, and save waveform information to a file for analysis in Cadence SimVision.

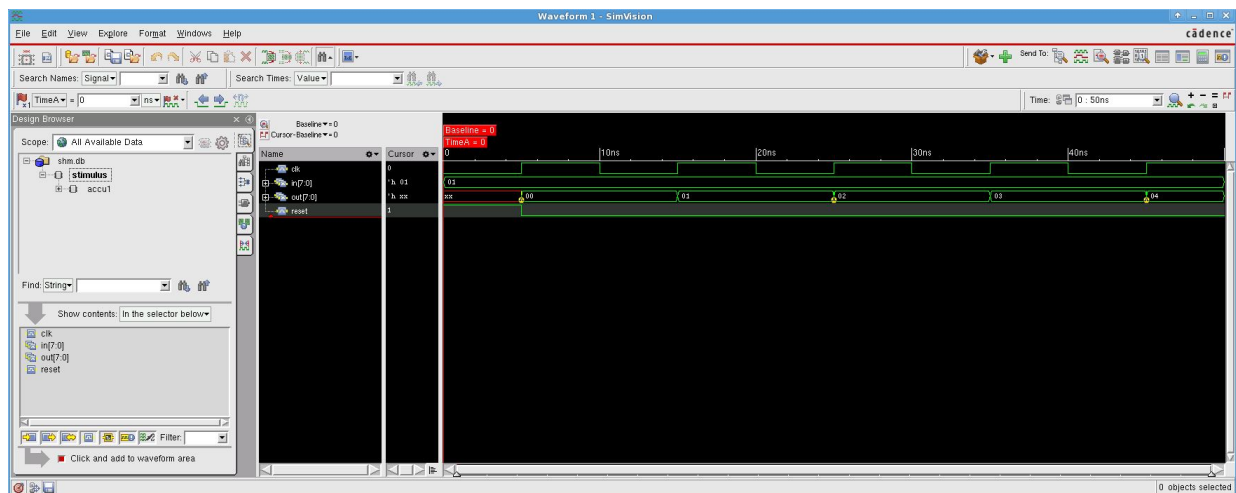
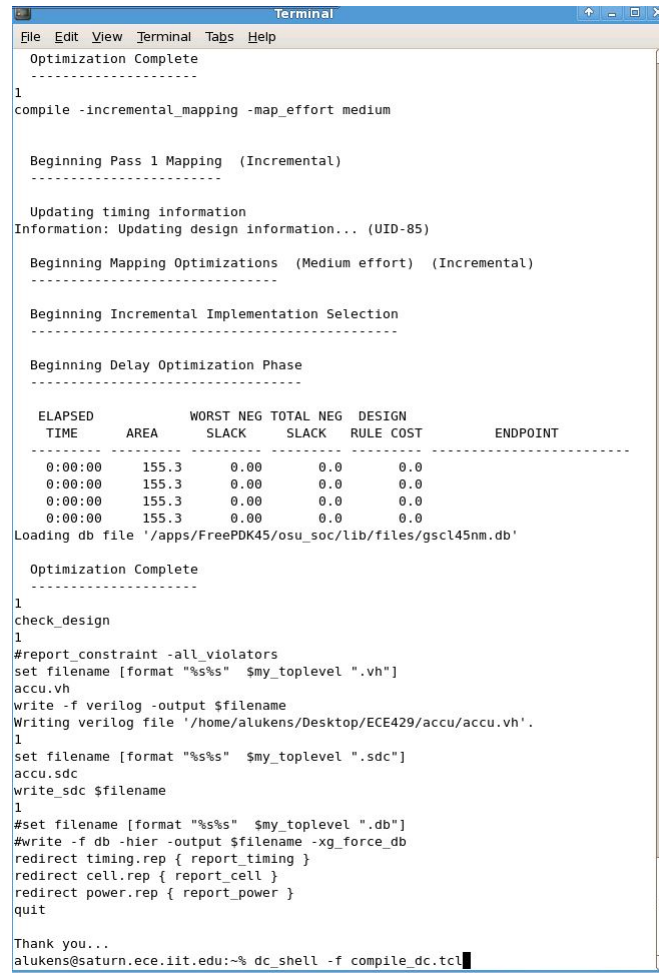


Figure 3: RTL Simulation Results (SimVision)

After confirming the functionality of the original verilog file, the next step is to perform logic synthesis on the design using Synopsys Design Compiler. This will receive the accu.v Verilog file, and output a gate-level (structural) Verilog design that performs identically to the accu.v file, but uses only standard cells available

in the process library. The logic synthesis program also performs automatic design simplification to minimize logic complexity and produce a better VLSI design.



```

Terminal
File Edit View Terminal Tabs Help
Optimization Complete
-----
1
compile -incremental_mapping -map_effort medium

Beginning Pass 1 Mapping (Incremental)
-----

Updating timing information
Information: Updating design information... (UID-85)

Beginning Mapping Optimizations (Medium effort) (Incremental)
-----

Beginning Incremental Implementation Selection
-----

Beginning Delay Optimization Phase
-----

ELAPSED      WORST NEG TOTAL NEG  DESIGN
TIME         AREA      SLACK   SLACK   RULE COST      ENDPOINT
-----
0:00:00      155.3      0.00    0.0     0.0
0:00:00      155.3      0.00    0.0     0.0
0:00:00      155.3      0.00    0.0     0.0
0:00:00      155.3      0.00    0.0     0.0
Loading db file '/apps/FreePDK45/osu_soc/lib/files/gscl45nm.db'

Optimization Complete
-----
1
check_design
1
#report_constraint -all_violators
set filename [format "%s%s" $my_toplevel ".vh"]
accu.vh
write -f verilog -output $filename
Writing verilog file '/home/alukens/Desktop/ECE429/accu/accu.vh'.
1
set filename [format "%s%s" $my_toplevel ".sdc"]
accu.sdc
write_sdc $filename
1
#set filename [format "%s%s" $my_toplevel ".db"]
#write -f db -hier -output $filename -xg_force_db
redirect timing.rep { report_timing }
redirect cell.rep { report_cell }
redirect power.rep { report_power }
quit

Thank you...
alukens@saturn.ece.iit.edu:~% dc_shell -f compile_dc.tcl

```

Figure 4: Logic Synthesis Result

Once logical synthesis is performed, the output from Synopsys Design Compiler (accu.vh) should be simulated to ensure correct functionality versus the original Verilog design. This is done by again running Cadence Verilog-XL, and examining the results.

Once the functionality of the synthesized design is confirmed, the next step is to place and route the design. This is done using Cadence Encounter. This will optimize the placement and interconnect of the individual cells in the synthesized design, and output a final.v Verilog file with additional timing information, and a GDSII database file (final.GDSII) that holds all placement information. The Encounter program also produces various timing, power, and area reports, which can be used to estimate the constraints of the design in the specified design process

```

saturn.ece.iit.edu
File Edit View Terminal Tabs Help
Total Power: 0.415

-----
report_power consumed time (real time) 00:00:00 : peak memory (475M)
Output file is power.final
*****
* Encounter script finished *
*
* Results: *
* ----- *
* Layout: final.gds2 *
* Netlist: final.v *
* Timing: timing.rep.5.final *
* Area: area.final *
* Power: power.final *
*
* Type 'win' to get the Main Window *
* or type 'exit' to quit *
*
*****
encounter 1> reportGateCount -limit 0
Gate area 2.8158 um^2
[0] accu Gates=56 Cells=30 Area=158.6 um^2
encounter 2> report_power
-----

*
* Encounter 10.13-s292_1 (32bit) 08/15/2012 15:12 (Linux 2.6)
*
*
* Date & Time: 2020-Nov-12 19:52:25 (2020-Nov-13 01:52:25 GMT)
*
*
-----

```

Figure 5: Encounter Area Report

```

3
Terminal
File Edit View Terminal Tabs Help

Total Power
-----
Total Internal Power: 0.2402 57.88%
Total Switching Power: 0.1738 41.89%
Total Leakage Power: 0.000954 0.2299%
Total Power: 0.415
-----

Group Internal Power Switching Power Leakage Power Total Power Percentage
-----
Sequential 0.1531 0.0291 0.0004398 0.1827 44.02
Macro 0 0 0 0 0
IO 0 0 0 0 0
Combinational 0.05951 0.03121 0.0004403 0.09116 21.97
Clock (Combinational) 0.02752 0.1135 7.386e-05 0.1411 34.01
Clock (Sequential) 0 0 0 0 0
Total 0.2402 0.1738 0.000954 0.415 100

Rail Voltage Internal Power Switching Power Leakage Power Total Power Percentage
-----
vdd 1.1 0.2402 0.1738 0.000954 0.415 100

Clock Internal Power Switching Power Leakage Power Total Power Percentage
-----
clk 0.02752 0.1135 7.386e-05 0.1411 34.01
Total 0.02752 0.1135 7.386e-05 0.1411 34.01

-----
Power Distribution Summary:
* Highest Average Power: clk_L1_I0 (INVX8): 0.04083
* Highest Leakage Power: r1/q_reg (OFFF05K1): 5.498e-05
* Total Cap: 4.01408e-13 F
* Total instances in design: 30
* Total instances in design with no power: 0
* Total instances in design with no activity: 0
* Total Fillers and Decap: 0
-----
report_power consumed time (real time) 00:00:00 : peak memory (475M)
1
encounter 6>

```

Figure 6: Encounter Power Report

```

Terminal
File Edit View Terminal Tags Help
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 30%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 35%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 40%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 45%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 50%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 55%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 60%
... Calculating internal and leakage power
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 65%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 70%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 75%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 80%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 85%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 90%
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT): 95%

Finished Calculating power
2020-Nov-12 19:48:37 (2020-Nov-13 01:48:37 GMT)
*

Total Power
-----
Total Internal Power: 0.2402 57.88%
Total Switching Power: 0.1738 41.89%
Total Leakage Power: 0.000954 0.2299%
Total Power: 0.415

-----
report power consumed time (real time) 00:00:00 : peak memory (487M)
Output file is power.final
*****
* Encounter script finished *
* Results: *
* ----- *
* Layout: final.gds2 *
* Netlist: final.v *
* Timing: timing.rep.5.final *
* Aros: area.final *
* Power: power.final *
* Type 'win' to get the Main Window *
* or type 'exit' to quit *
* *
*****
encounter >
encounter > exit

*** Memory Usage v#1 (Current mem = 268.953M, initial mem = 46.070M) ***
... Ending "Encounter" (totcpu=0:00:05.2, real=0:00:23.0, mem=269.0M) ...
alukens@saturn.ece.iit.edu:~$

```

Figure 7: Encounter Command Result

After the completion of the Encounter command, a GDSII database file was created with the finalized layout of the design. Before importing this file into Cadence Virtuoso, it is important to ensure that the final.v file created by Encounter has the same functionality as the original file. This can be done using Cadence Formality (not Formality ESP) to check the validity of the final.v file versus the accu.v file that was initially created.

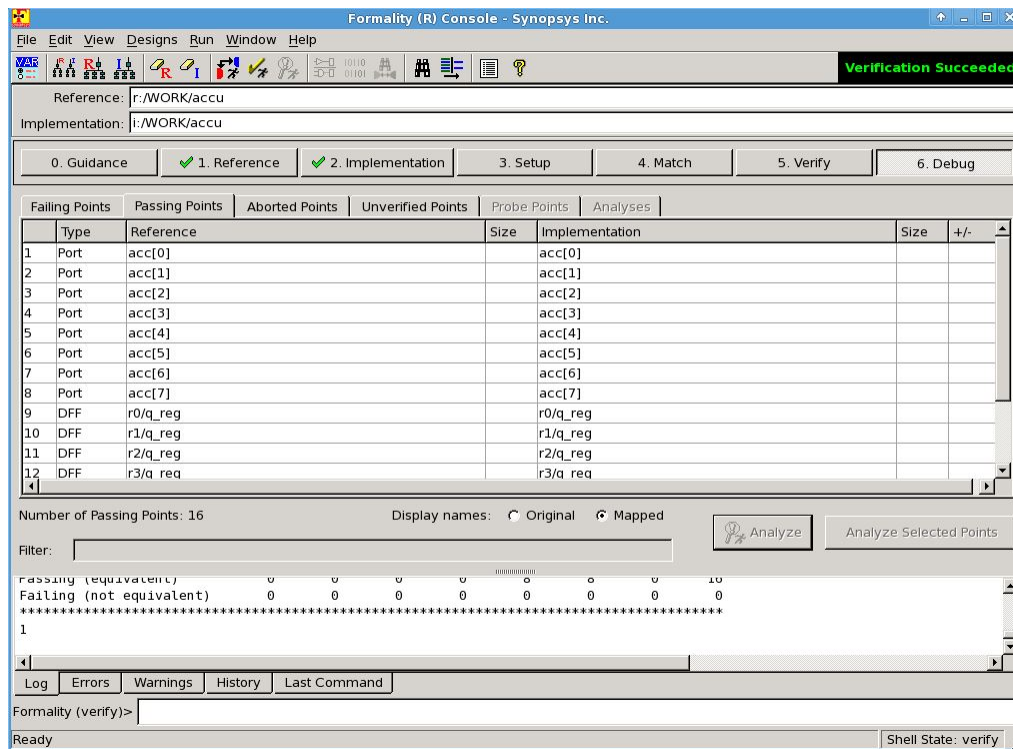


Figure 8: Formality Equivalency Result

At the completion of the Formality Equivalency Checking, one can be assured that the routed design will function identically to the original design. The design can now safely be imported and used in Cadence Virtuoso via the final.GDSII file created by Encounter, knowing that the layout will function correctly. Before sending the design to a silicon foundry to be produced, it is still important to run design rule checking inside Cadence Virtuoso to ensure the place and route did not make any errors, or that any additional logic does not interfere.



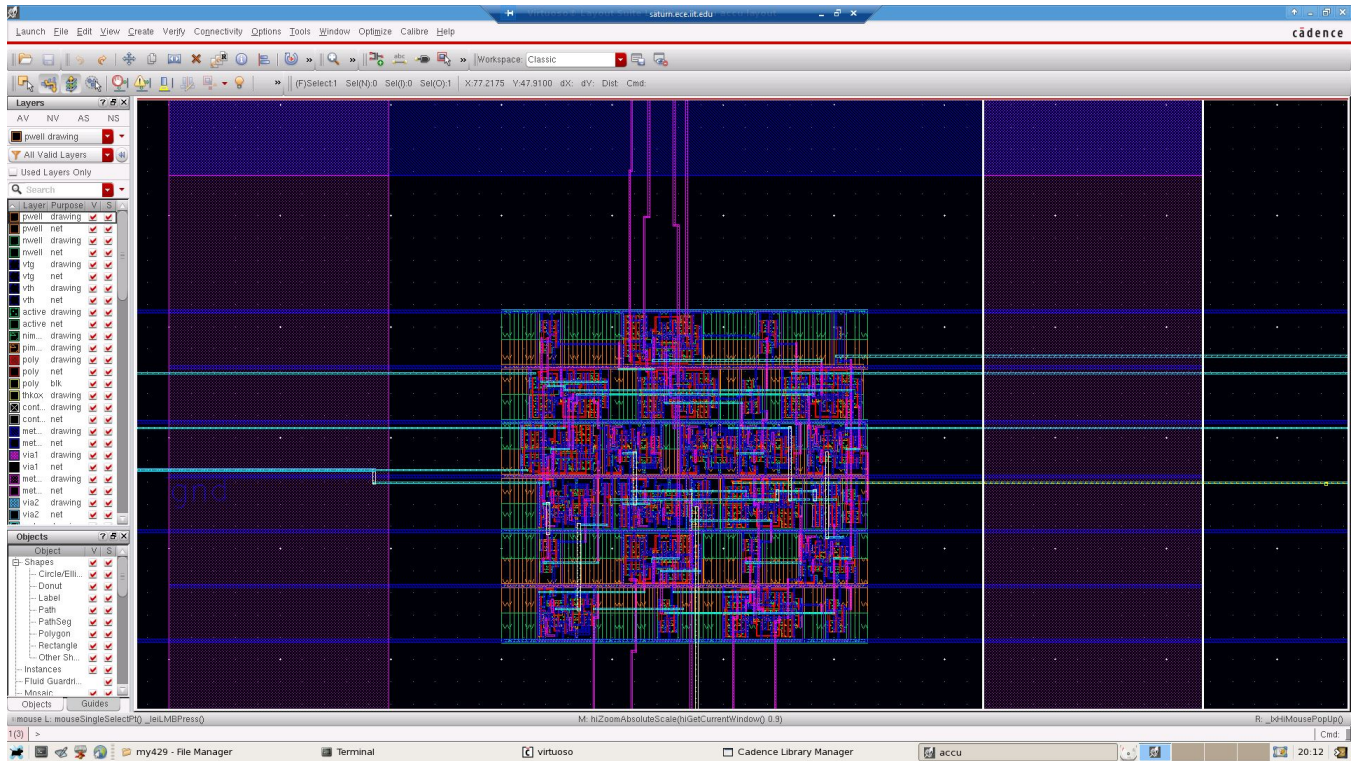


Figure 9: Layout Result in Cadence Virtuoso

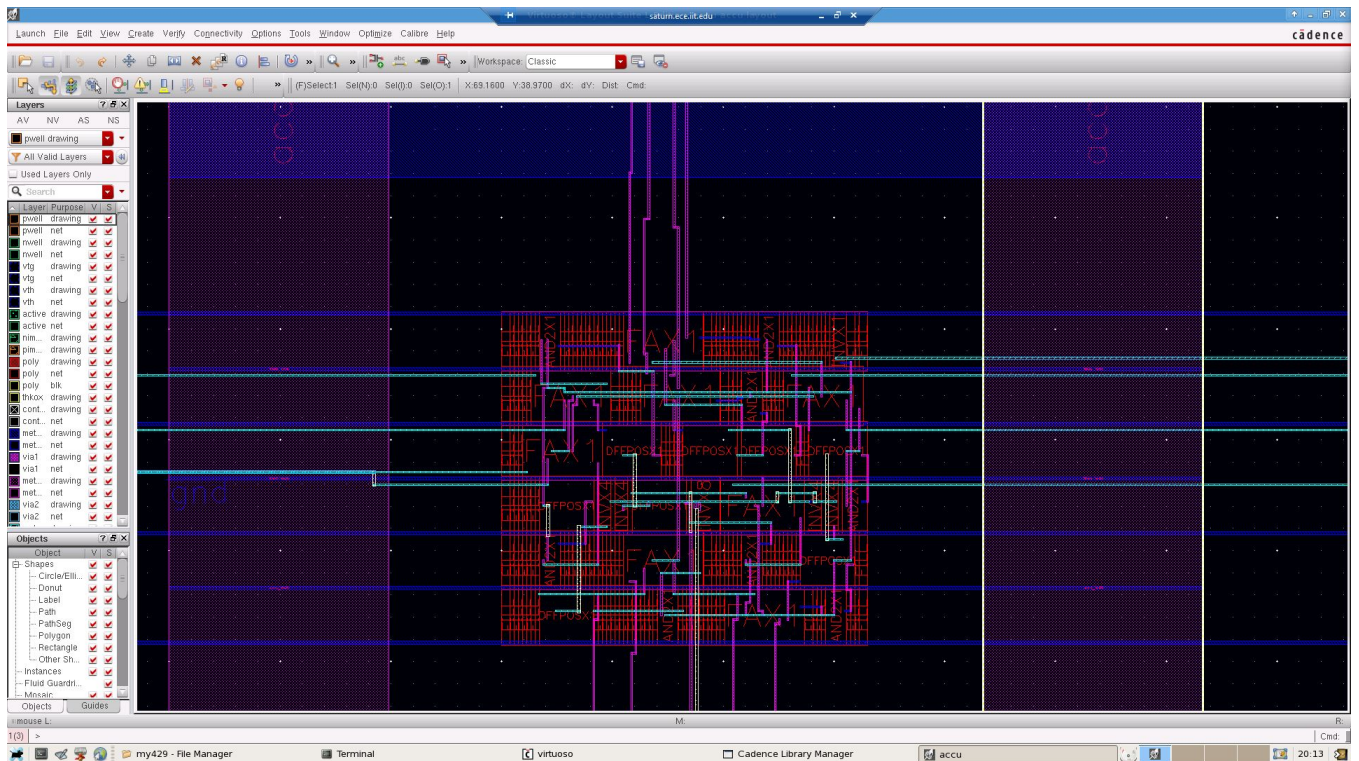


Figure 10: Layout Result in Cadence Virtuoso

## 4 Deliverables

- **What is a standard cell?**

A standard cell is a digital logic gate that is designed specifically to be utilized with a certain VLSI fabrication/process node. For example, in a process library, there will be standard cells for various sizes of inverters, NAND gates, NOR gates, and other logic gates, each designed to be used with other gates from the same library. By utilizing standard cells from the same library, VDD, GND, input, and output locations can be optimized to reduce circuit complexity. This will result in better area, power, and timing for circuits created on this process.

Furthermore, because the layout of the standard cell is known, the functional characteristics of the cell can be approximated much better than traditional cells, allowing for increased accuracy in pre-synthesis simulations (SPICE, etc).

- **What are the differences among 'accu.v', 'accu.vh', and 'final.v'?**

The original 'accu.v' Verilog file is the most general of the three files, containing no specific implementation details about the 8-bit accumulator design. The description of the accumulator in this file is considered to be at the register transfer level (RTL level design). The 'accu.vh' file is the Verilog produced after the completion of logic synthesis on the original design (using Synopsys Design Compiler). This file creates the functionality of the original design completely in terms of standard cells. This enables placement and routing to be performed on the design using Cadence Encounter, which will result with the 'final.v' file. The 'final.v' file has additional considerations (such as buffers on the interconnect to reduce delay) in order to optimize the performance of the design when implemented in a physical design. This file also is represented completely in terms of standard cells, and is considered to be a "structural" design.

- **How does the area of your design change after place & route?**

Before place and route, the synthesized design has no information about how the different logic cells are to be physically connected. In other words, the area consumed by the synthesized design only accounts for the area of the standard cells. After place and route, the layout of the VLSI design has additional information about the interconnect between each of the standard cells, resulting in a larger area estimation. In practice, optimizations can be performed in the place and route process to optimize for minimal delay, area, or power as needed.

- **How does the timing of your design change after place & route?**

The required time (delay) of the circuit logic was found to be increased after the place and route process. After the logic synthesis stage (in Synopsys Design Compiler), the required time was found to be approximately 0.94ns. After running place and route using Cadence Encounter, the total required time was found to be approximately 1.041ns. This result makes sense, as Encounter may have added additional logic (such as buffers on certain interconnect paths) to decrease power or area consumption at the expense of increased delay.

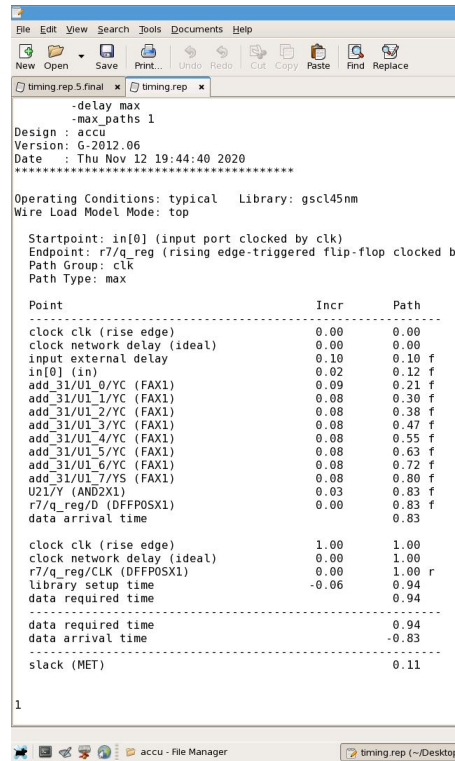


Figure 11: Logic Synthesis Timing Results

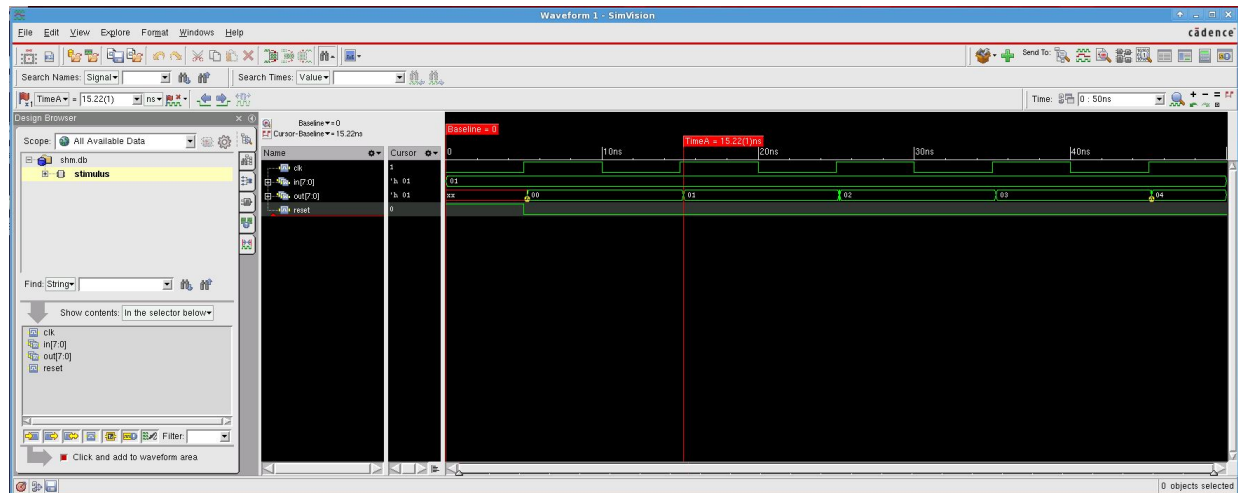


Figure 12: Logic Synthesis Simvision

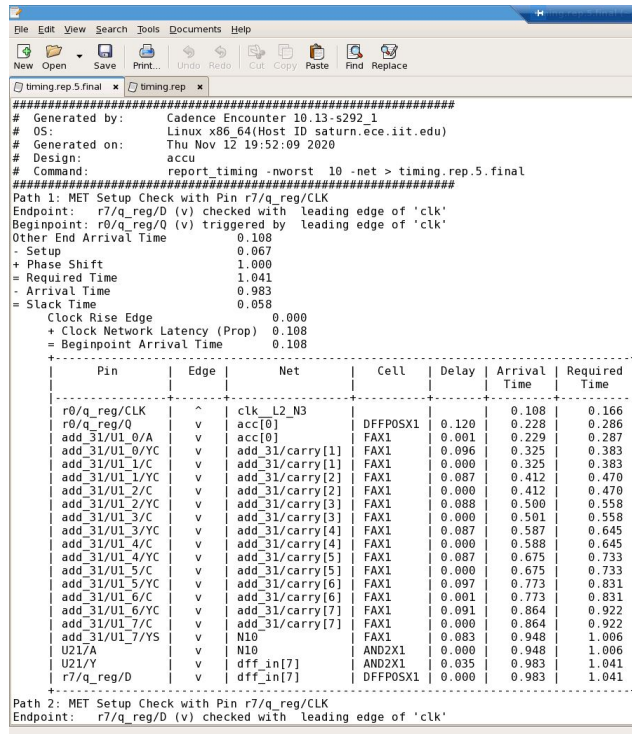


Figure 13: Place & Route Timing Results

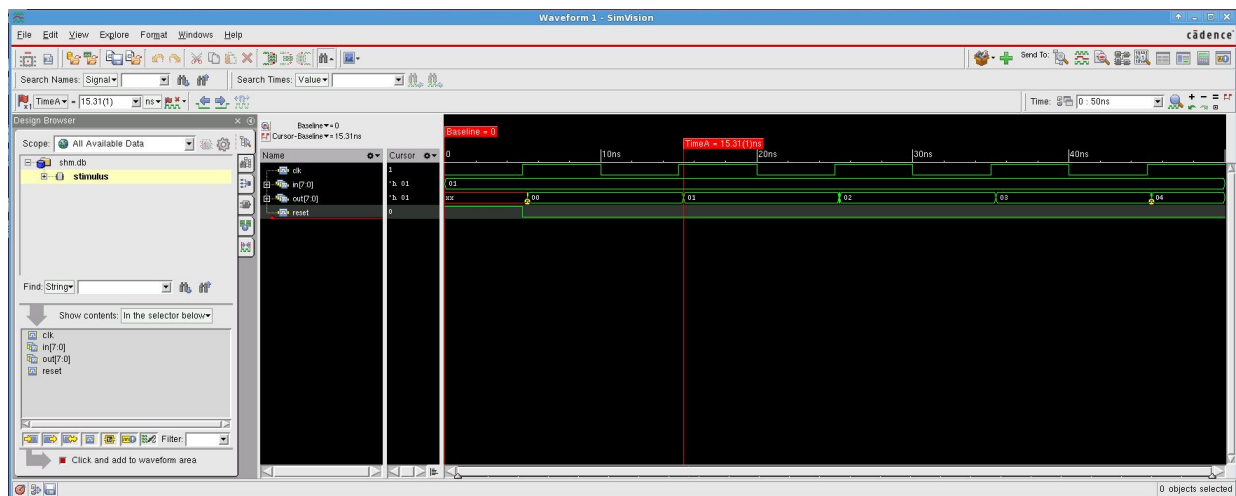


Figure 14: Place & Route Simvision

- Why do we need to use Virtuoso to generate the final layout for fabrication? What information is available from 'final.gds2'?

Cadence Virtuoso is required to generate the final layout for fabrication, because the GDSII file does not contain all of the necessary information required to fully realize the VLSI design. Specifically, the GDSII file does not include all of the layout information of each of the individual cells (polysilicon, wells, implant regions, etc.). Only the placement of cells and the layout of the metal interconnect between the standard cells are included in the GDSII file. To add the information within each standard cell, the design must be imported into Cadence Virtuoso, where a process library can be specified that has

all of the internal information about each standard cell.

## 5 Conclusions

This laboratory should be considered a success. Students were successfully able to implement an 8-bit accumulator in Cadence Virtuoso by following the Standard Cell Based ASIC Design Flow. This required students to design and implement an 8-bit accumulator in Verilog, utilize Synopsys Design Compiler to produce the design in terms of standard cells, and then use Cadence Encounter to place & route the design. Throughout this process, Verilog-XL and a verilog testbench file were utilized to ensure proper functionality of the design at every stage of the ASIC flow. The result of this design process is a fully realized 8-bit accumulator circuit, with specific layout information such that it can be produced on the specified design process.

Students will utilize the topics learned in this laboratory to produce increasingly complex digital logic circuits in the Cadence EDA suite. In this laboratory, students skills at producing circuits in Verilog were also enhance

## 6 Resources

- Choi, Ken. “ECE 429 Laboratory 9 Manual.” Illinois Institute of Technology, November 23, 2020.
- Kim, Victoria. “ECE 429 Guideline for Writing Report & Grading Criteria.” Illinois Institute of Technology, November 23, 2020.