Project: Analyzing a Digital Timing Report Using Python

Oklahoma State University – School of Electrical and Computer Engineering
Instructor: James E. Stine and Marcus Mellor
Date Assigned: November 9, 2025

Background: Synthesis and System-on-Chip (SoC) Design

Modern electronic devices—from smartphones and laptops to automobiles and satellites— depend on custom digital integrated circuits known as System-on-Chip (SoC) designs [1]. A System-on-Chip integrates millions (or even billions) of transistors onto a single piece of silicon to perform complex functions such as computation, communication, signal processing, and control. Unlike a traditional printed circuit board that uses many separate chips, an SoC combines processors, memory, I/O interfaces, and specialized hardware accelerators into a unified design [2].

Digital Design Flow Overview

Designing an SoC begins at a high level of abstraction and progresses through several stages that gradually convert human-readable code into physical hardware:

1. **Specification and RTL Design:** Engineers describe the circuit's intended behavior using a hardware description language (HDL) such as `SystemVerilog` or `VHDL`. This level is called *Register-Transfer Level* (RTL) design, where designers specify how data moves between registers and how logic operates each clock cycle.

2. **Logic Synthesis:** A synthesis tool (such as Synopsys Design Compiler or Cadence Genus) automatically converts the HDL description into a gate-level netlist made of logic gates and flip-flops from a standard-cell library. A standard-cell library is a collection of pre-characterized logic gates, flip-flops, and other building blocks—each with known electrical, timing, and physical properties—developed for a specific fabrication technology. These cells form the basic components used by synthesis and layout tools to implement digital designs efficiently and predictably. During synthesis, the tool selects gates that meet the desired timing, area, and power constraints. The result is a structural representation that can be implemented in silicon.

3. **Static Timing Analysis (STA):** Once the circuit has been mapped to gates, engineers perform timing analysis to ensure that signals can propagate through the logic fast enough to meet the clock period. Tools generate reports showing the critical paths—the slowest signal paths that determine the maximum achievable clock frequency. These reports include details such as incremental gate delays, wire capacitances, transition times, and total path slack.

4. **Physical Design and Fabrication:** After timing is verified, place-and-route tools arrange the gates on the chip, connect them with metal wires, and prepare layout data for fabrication. The final design is then manufactured using a semiconductor process (for example, a 130 nm or 7 nm technology node).

Why Analyze Timing Reports?

Timing reports are essential for verifying that a digital design will operate correctly at its target clock frequency. Each *path* in a timing report traces the delay from a **startpoint** (such as a flip-flop output or input port) to an **endpoint** (such as another flip-flop or output port). The cumulative delay determines whether the circuit meets timing constraints.

For this project, you are analyzing a sample timing report produced after synthesis of a small component, such as a leading-zero counter (`lzc`). By writing a Python program to parse and analyze the data, you will gain insight into how numerical computation, data parsing, and visualization techniques can be applied to real-world engineering workflows in the semiconductor industry.

Connecting to System-on-Chip Design

Even though this assignment focuses on a single synthesized block, the same principles scale up to large SoC projects: a modern chip may contain thousands of modules like the one in this exercise, each with hundreds of timing paths analyzed automatically. By learning to interpret and visualize these reports, you are exploring the quantitative foundation of digital design—how mathematical analysis, data processing, and optimization interact to produce high-performance computing systems.

After a digital design is synthesized, an Electronic Design Automation (EDA) tool (such as Synopsys Design Compiler or Cadence Genus) generates a timing report that lists delays along critical signal paths in the circuit. Several sample reports are included in our class respository. Each report contains information about gate delays, wire capacitances, and total slack for timing verification.

A typical portion of a report looks as follows:

```
****************************************
Report : timing
        -path full
        -delay max
        -nworst 10
        -nets
        -max_paths 10
        -transition_time
        -capacitance
Design : lzc
Version: W-2024.09-SP4-1
Date   : Fri Nov  7 12:47:36 2025
****************************************

Operating Conditions: PVT_1P2V_25C   Library: scc9gena_tt_1.2v_25C
Wire Load Model Mode: top

  Startpoint: num[28] (input port clocked by vclk)
  Endpoint: ZeroCnt[0] (output port clocked by vclk)
  Path Group: vclk
  Path Type: max

  Point                        Fanout      Cap     Trans      Incr       Path
  -------------------------------------------------------------------------------
  clock vclk (rise edge)                                    0.000000   0.000000
  ...
  ZeroCnt[0] (out)                               0.036924   0.000000   0.215938 r
  data arrival time                                                    0.215938

  clock vclk (rise edge)                                    0.217391   0.217391
  ...
  slack (MET)                                                          0.001453
```

Objective

The goal of this assignment is to use Python to automatically parse and analyze such timing reports, extracting structured information and computing useful statistics about delay, capacitance, and slack.

Tasks

(a) **Parsing the Report**
Write a Python script to read a text file containing a synthesis timing report. Your code should:

- Identify the `Startpoint`, `Endpoint`, and `Path Group`.

- Extract all rows of the timing table (Point, Fanout, Capacitance, Transition, Incremental Delay, Path Delay).
- Locate and extract the final `slack` value.

(b) **Computing Statistics**
Compute and display:

- Total number of stages (rows) in the path.
- Average and maximum incremental delay.
- Total wire capacitance (sum of Cap values where available).
- The final data arrival time and overall path delay.
- The reported slack.

(c) **Visualization**
Create a line or bar chart showing the cumulative delay at each stage along the path using `matplotlib`. Label each cell or gate name on the x-axis.

(d) **Extensions (Optional, +10 points)**

- Allow the script to parse multiple paths from the same report.
- Write parsed path data to a CSV file.
- Compute and visualize the worst slack across all paths.

Numerical Extension: Delay Modeling and Regression Analysis

Background

In digital design, the propagation delay through a logic gate depends primarily on two factors: (1) the gate's intrinsic delay, and (2) the capacitive load it drives. This relationship can often be approximated by a simple linear model,

$$t_{pd} = a + b \cdot C,$$

where:

- t_{pd} = propagation delay of the gate (ns),
- C = total output load capacitance (pF),
- a = intrinsic or no-load delay, and
- b = delay sensitivity to capacitive loading.

This expression is the foundation of timing characterization for standard cells in modern CMOS design. By analyzing timing reports and fitting data from synthesis or static timing analysis (STA), engineers can estimate the parameters a and b using the least-squares method.

Digital Gate Delay and the Logical Effort Model

Every digital circuit, from a single inverter to a full System-on-Chip (SoC), is built from logic gates that process and propagate signals over time. The speed at which a signal travels through a logic gate is limited by the transistor sizes and the capacitive load that the gate must drive. Understanding this delay is fundamental to timing analysis and circuit optimization.

A. Gate Delay Fundamentals

When a logic signal changes, the transistors in the gate charge or discharge internal and external capacitances through a finite resistance. The resulting propagation delay can be approximated as:

$$t_p = k \cdot R_p \cdot C_L,$$

where:

- R_p is the effective resistance of the transistors,
- C_L is the total load capacitance (from wiring and driven inputs),
- k is a technology-dependent proportionality constant.

Smaller transistors switch slower because of higher resistance, while larger transistors switch faster but occupy more area and add more capacitance to the previous stage. Designers must therefore balance speed or performance, power, and area. This is sometimes called PPA for SoC designs.

B. Logical Effort: A Simple Delay Model

The Logical Effort (LE) method provides a practical way to estimate and compare gate delays without transistor-level simulation [3]. The delay of a logic gate relative to an ideal inverter is modeled as:

$$d = g \cdot h + p,$$

where:

- d is the delay in units of an inverter delay,
- g is the **logical effort** — a measure of how much worse a gate is at driving current compared to an inverter with the same input capacitance,
- h is the **electrical effort** (fanout) — the ratio of output load capacitance to the gate's input capacitance, $h = \frac{C_{\text{out}}}{C_{\text{in}}}$,
- p is the **parasitic delay** — internal capacitances within the gate.

The total delay of a multi-stage path can be approximated by summing the stage delays:

$$D_{\text{path}} = \sum_i (g_i \cdot h_i + p_i).$$

This abstraction lets engineers estimate the delay of complex paths using only gate ratios and capacitances, which is especially useful for quick optimization and comparison during synthesis.

C. Design Implications

Logical Effort reveals two key design insights:

1. **Gate topology matters.** NAND and NOR gates are inherently slower than inverters because their series transistors give them a higher logical effort ($g > 1$).
2. **Load balancing is critical.** Each stage in a path should drive approximately the same electrical effort to minimize total delay. This leads to the concept of the **optimum stage effort**, $f = gh \approx 4$, for many CMOS technologies.

D. Relation to Timing Reports and Synthesis

During synthesis and static timing analysis, EDA tools use far more detailed transistor models (including nonlinear capacitances and wire RC networks), but the same principles still apply:

- Each gate contributes an intrinsic delay (analogous to p),
- Each output load contributes a dynamic delay (analogous to $g \cdot h$),
- The sum of these determines the total path delay reported after synthesis.

Thus, the timing report you are analyzing numerically reflects the same trade-offs described by the Logical Effort model. Higher fanout or complex gates increase electrical and logical effort, leading to longer propagation times and potential critical paths in the design.

Least-Squares Definition

The least-squares method finds parameters a and b that minimize the sum of squared errors between the observed and predicted delays [4, 5]:

$$\min_{a,b} \sum_{i=1}^{n} \left(t_{pd,i} - (a + b \cdot C_i)\right)^2 .$$

The analytical solution is obtained by solving the normal equations:

$$\begin{bmatrix} n & \sum C_i \\ \sum C_i & \sum C_i^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum t_{pd,i} \\ \sum C_i \cdot t_{pd,i} \end{bmatrix} .$$

This approach can be extended to multiple variables, such as fanout or transition time, resulting in a multivariate linear model:

$$t_{pd} = a + b \cdot C + c \cdot F,$$

where F is the gate fanout.

Objective

Using the timing data you extracted from the synthesis report, your goal is to numerically model the relationship between incremental delay and the load parameters (capacitance and fanout) using least-squares regression.

Tasks

(a) **Linear Delay Model**
Fit a least-squares linear model of the form

$$t_{pd} = a + b \cdot C.$$

Report the estimated values of a and b, and compute the coefficient of determination (R^2) to evaluate the quality of the fit.

(b) **Multivariate Model**
Extend your model to include fanout:

$$t_{pd} = a + b \cdot C + c \cdot F.$$

Compare the regression coefficients and discuss whether fanout significantly improves prediction accuracy.

(c) **Visualization**
Plot measured incremental delay versus capacitance, overlaying the fitted regression line. On a second plot, show measured versus predicted delay for both the single-variable and multivariate models.

(d) **Numerical Sensitivity (Optional +10 pts)**
Estimate the sensitivity of total path delay to small changes in capacitance using finite differences:

$$\frac{\partial t_{\text{total}}}{\partial C_i} \approx \frac{t_{\text{total}}(C_i + \Delta C) - t_{\text{total}}(C_i)}{\Delta C}.$$

Discuss which gates are most sensitive to load changes.

(e) **Discussion**
Comment on whether a linear model adequately describes the delay behavior. Would a quadratic fit ($t_{pd} = a + b \cdot C + c \cdot C^2$) better capture the observed data? Support your argument with numerical evidence.

Deliverables

- A Python script named `timing_analyzer.py`.
- Example terminal output showing extracted timing information.
- A plotted chart of cumulative delay vs. path element.
- A report including your regression coefficients, R^2 values, and plots.
- Python source code implementing both single and multivariate least-squares fits.
- A short discussion describing whether the data suggests a linear or nonlinear dependency between delay and load.
- A brief report ($\approx$ 1 page) summarizing what your program finds and how it could be used to automate timing analysis in a VLSI design flow.
- You must explain what is going on for full credit. Any partial explanation could mean that you used some large-language model to try to solve the problem.

Discussion Prompt

How could this parsing approach be extended to automatically find the most critical path in an entire synthesis report? What other metrics (e.g., power, capacitance, or transition time) could be used to build a regression model predicting delay?

References

[1] D. Harris, J. Stine, R. Thompson, and S. Harris, *RISC-V System-On-Chip Design.* Elsevier Science, 2025. [Online]. Available: https://books.google.com/books?id=53fS0AEACAAJ

[2] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. USA: Addison-Wesley Publishing Company, 2010.

[3] I. Sutherland, B. Sproull, and D. Harris, *Logical effort: designing fast CMOS circuits.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[4] S. C. Chapra and P. J. Clough, *Applied Numerical Methods with Python for Engineers and Scientists.* New York: McGraw-Hill Education, 2021.

[5] J. H. Mathews and K. K. Fink, *Numerical Methods Using MATLAB*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2004.