Cairo University
Faculty of Engineering
Computer Engineering Department

CMP3020 — VLSI

Lab Assignment 5

15 Dec 2025 • Timing Closure

Muhammad Sayed

Time is the only resource we cannot recycle.

INTRODUCTION

In digital design, verification occurs in two stages: **Functional Verification** ("Does it work?") and **Timing Verification** ("Is it fast enough?").
A design that passes RTL simulation perfectly may still fail in the real world. Why? Because RTL simulation assumes ideal, instant transitions. Real hardware has propagation delays, capacitance, and setup/hold time requirements.
To communicate these physical realities to the synthesis tool, we use an **SDC (Synopsys Design Constraints)** file. Without an SDC, the tool assumes an ideal environment, and the synthesis results can be dangerously optimistic. A module may seem "Satisfactory" (Positive Slack) initially, but once the SDC is introduced to model input delays, output loads, and clock jitter, the design might suddenly fail (Negative Slack).
When this happens, we have two choices:

1. **Change the RTL:** Optimize the logic (e.g., pipelining).
2. **Refine the Synthesis:** Adjust the configuration (e.g., Clock Period) to match the physical limits of the gates.

In this lab, we will focus on the second approach: **Achieving Timing Closure via Configuration Refinement.**

DESIGN SPECIFICATIONS

You are provided with the following source files in the src/ and configs/ directories. These assets simulate a real-world scenario where the RTL and Constraints are fixed requirements from the system architect, and your role is to achieve timing closure.

Traffic Light Controller (src/traffic_light.v)

This module implements an intelligent Traffic Light Controller for a busy intersection. It is designed to be robust and responsive to sensor inputs.

DESIGN SPECIFICATIONS

You are provided with source files in the src/ and configs/ directories. These assets simulate a real-world scenario where the RTL and Constraints are fixed requirements from the system architect, and your role is to achieve timing closure.

Traffic Light Controller

This module implements an intelligent Traffic Light Controller for a busy intersection. It is designed to be robust and responsive to sensor inputs.

System Interface

The controller takes a system clock and sensor inputs from the road. It drives three outputs corresponding to the Red, Yellow, and Green lights.

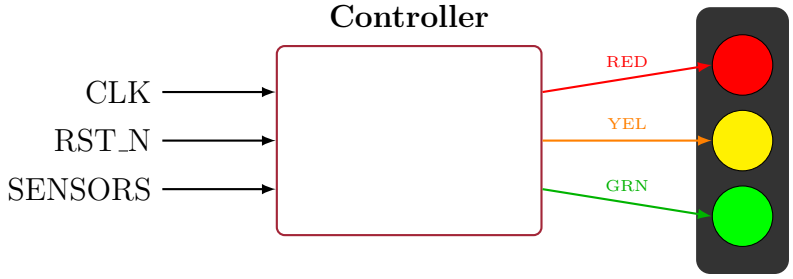

Figure 1: System Interface

Adaptive Timing Logic

Unlike a fixed timer, this controller adapts the Green Light duration based on traffic density. The specific logic for calculating the time limit is defined as:

- **High Traffic (AND):** If both North and East sensors are active, the standard config_time is used to prevent starvation (fair sharing).
- **Medium Traffic (OR):** If only one side has traffic, the Green light duration is extended (+20) to clear the lane.

FSM Operation

The system cycles through IDLE → GREEN → YELLOW → RED. The duration of the GREEN state is determined by the adaptive logic above.

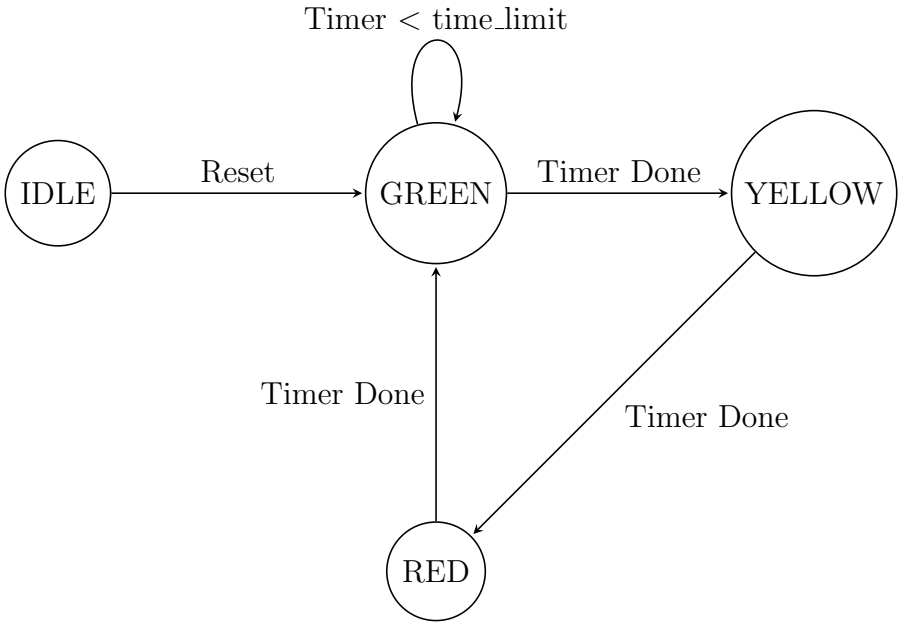

Figure 2: Simplified FSM Diagram

Testbench Strategy

The testbench is designed to stress the adaptive timing logic described above. It performs two specific checks:

1. **Standard Duration:** It asserts NO sensors and measures the Green light duration. It expects the default config_time (e.g., 50 cycles).
2. **Adaptive Extension:** It asserts sensor_north = 1 while keeping sensor_east = 0.
   - It expects the design to switch the time_limit wire to config_time + 20.
   - The simulation waits and verifies that the Green state persists for exactly 20 extra clock cycles compared to the standard test.

Design Constraints

The SDC file is the "Rulebook" for the synthesis tool. It describes the timing budget available for your module.

- **Input Delay:** The time consumed by the external sensor chip before the signal reaches your pin.
- **Output Delay:** The setup time required by the external high-power lamp driver connected to your output.
- **Load:** The capacitance of the PCB trace, which slows down the output transition.

Visualizing the Budget: The diagram below illustrates why adding SDC constraints causes negative slack. The "Available Time" for your internal logic is effectively squeezed from both sides.

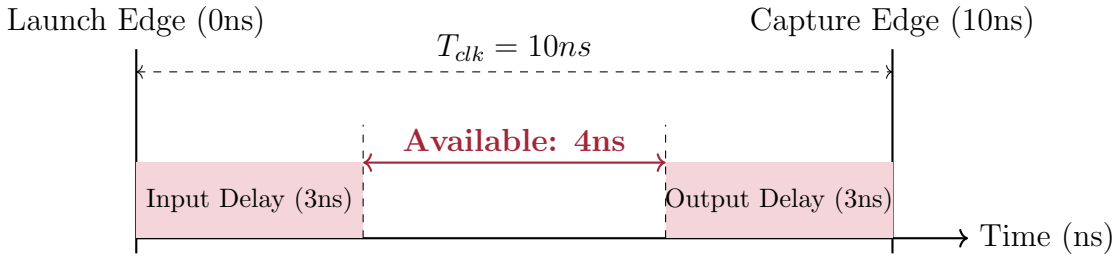

Figure 3: The "Squeezed" Timing Budget

LAB PROCEDURE

Baseline Analysis

We start with a "Baseline" configuration. This config does **not** include the SDC file. The tool assumes an ideal world with zero external delays and zero capacitive load.
Execute the baseline synthesis:

```
openlane configs/config_baseline.json --design-dir . --run-tag
   run_baseline
```

Outcome: The synthesis will likely succeed with **Positive Slack**. The tool reports that the chip is "Fast Enough" because it is assuming ideal conditions.

Constraint Introduction

Now we switch to the "Fail" configuration. This config explicitly includes traffic_light.sdc. Execute the constrained synthesis:

```
openlane configs/config_fail.json --design-dir . --run-tag run_fail
```

Outcome: The synthesis will likely fail with **Negative Slack**.

- **Why?** The SDC defines Input/Output delays and a high Capacitive Load.
- **The Effect:** The "Input Delay" eats into the start of your budget, and the "Output Delay" moves the deadline earlier. Simultaneously, the "Output Load" makes your gates physically slower. These factors combined cause the logic to miss the timing requirements.

Achieving Timing Closure

Since the constraints (SDC) represent real-world requirements we cannot change, and the Clock Period is fixed, we must modify the **Synthesis Strategy** to make the hardware implementation more efficient.
You must create a new configuration file, configs/config_success.json, and populate it with parameters that enable aggressive timing optimizations. Consider implementing the following recommendations:

- **Enable Resizer Optimizations:** OpenLane has specific optimization passes during Placement (PL) and Global Routing (GLB) designed to repair timing violations. These are often disabled by default to save runtime; investigate how to enable timing-driven resizing.
- **Prioritize Speed:** The synthesis tool usually balances Area and Delay. You should adjust the SYNTH_STRATEGY to strictly prioritize **Delay**, forcing the tool to use faster, stronger gates even if they consume more area.
- **Drive Strength:** The SDC defines a high capacitive load on the output. Ensure that SYNTH_BUFFERING is enabled so the tool can insert the necessary buffers to drive this load quickly.

Goal: Apply these optimizations, run the synthesis, and verify that the **Worst Setup Slack** has flipped from negative to **Positive**.

POST-SYNTHESIS VERIFICATION (GLS)

Once you achieve timing closure with config_success.json, you must verify that the synthesized netlist still functions correctly and observe the effect of real-world delays.

Simulation Setup

You will run the simulation using **Icarus Verilog**. However, instead of the original RTL, you will compile the **Synthesized Netlist** (typically .pnl.v) generated by OpenLane.

SDF Annotation

To make the simulation accurate, you must include the **SDF (Standard Delay Format)** file. This file contains the exact delay of every gate and wire in your specific layout.

- You will need to modify the testbench (or create a wrapper) to load the SDF using the system task: $sdf_annotate("file.sdf", uut).
- This enables "Back-Annotation," where the simulator replaces ideal zero-delay gates with realistic timing values.

DELIVERABLES & ANALYSIS

Submit a .zip file containing your new config_success.json and a document ANALYSIS.md answering the following questions.

1. Slack & Budget Analysis

Analyze the results from the **Fail Run** (Step 2).

- **The Constraint Effect:** Why did adding the SDC file cause the slack to drop from Positive to Negative, even though the Verilog didn't change?
- **The Time Budget:** Calculate the actual time available for your internal logic using the formula:

$$T_{logic} = T_{clk} - T_{input\_delay} - T_{output\_delay}$$

- **The Violation:** Report the **Worst Setup Slack** value from the Fail Run. Based on this negative slack, how much faster did your logic need to be to pass?

2. Optimization Trade-offs

Analyze the results from the **Success Run** (Step 3).

- **The Fix:** List the specific parameters you added to config_success.json (e.g., resizing, buffering, strategy). Explain *why* you chose these specific settings.

- **The Cost of Speed:** Synthesis optimizations are rarely free. Compare the **Design Area** (design__instance__area) of the *Fail Run* vs. the *Success Run*.
  - Did the area increase?
  - Explain why achieving faster timing might result in a larger chip area.

3. Post-Synthesis Verification

Perform the Post-Synthesis simulation using your successful netlist and the SDF file.

- **Visualizing Delays:** Take a screenshot of the waveform showing the transition of the red, green, or yellow output.
- **Measurement:** Measure the exact time difference between the clk rising edge and the output signal change. Does this delay match the constraints?
- **Glitches:** Zoom in on the combinational logic transitions. Do you observe any "glitches" or hazards in the post-synthesis waveform that were not present in the ideal RTL simulation?