# Synthesis and Optimization of Digital Systems - Labs

Valentino Peluso

March 10, 2026

# Contents

# LAB 1 – Synopsys Tutorial: Logic Design

In this lab, you will familiarize yourself with a basic **logic synthesis flow using Synopsys Design Compiler (DC)**. You will run synthesis for a reference RTL design, explore the synthesis scripts, and analyze the impact of **clock constraints** and **operating conditions (PVT corners)** on the synthesis results.

## 1. Download the Lab Material from the Remote VM

The synthesis workspace is provided as a compressed archive on the Remote VM. You must first copy it to your home directory and then transfer it to the LABINF machine.

### 1.1 Connect to the Remote VM

- Connect to the Remote VM using **VS Code with SSH**.

### 1.2 Copy the synthesis workspace

From the Remote VM terminal, copy the provided archive to your current working directory:

```
cp /space/shared/sods/WORK_SYNTHESIS.zip .
```

### 1.3 Transfer and extract the archive

After copying the archive to your home directory on the **Remote VM**, you must download it to the **LABINF** workstation using **VS Code**.

Follow these steps carefully:

1. In **VS Code**, make sure you are connected to the **Remote VM** via SSH.
2. Open the **File Explorer** panel on the left.
3. **Right-click** on the file `WORK_SYNTHESIS.zip`.
4. Select **Download** from the context menu.
5. Choose the **`SODS` directory** on the LABINF computer as the destination.
6. Click **Download** to start the file transfer.

Wait until the download completes, then proceed to the extraction step:

1. Open the **`SODS` directory** in **VS Code**.
2. Open a terminal.
3. Extract the archive:

```
unzip WORK_SYNTHESIS.zip
```

After extraction, a folder named `WORK_SYNTHESIS` will be created.

# 2. Run the Synthesis Flow

In this section, you will inspect the provided TCL scripts and run logic synthesis using **Synopsys Design Compiler**.

## 2.1 Open the synthesis workspace

Move to the synthesis directory and open it in VS Code:

```
code ./WORK_SYNTHESIS
```

## 2.2 Inspect the main TCL scripts

Before running synthesis, take some time to **read and understand** the key configuration files:

- **Design Compiler setup file**

  This file defines Technology libraries and search paths:

  `./tech/STcmos65/synopsys_dc.setup`

- **SDC constraints for the design**

  This file defines timing constraints (clock, I/O delays, driving cells):

  `./rtl/keccak_f/sdc/keccak_f.sdc`

- **Main synthesis script**

  This file defines the synthesis flow executed by Design Compiler:

  `./scripts/synthesis.tcl`

## 2.3 Run synthesis

Launch Design Compiler and execute the synthesis script:

```
dc_shell-xg-t -f ./scripts/synthesis.tcl
```

Monitor the console output and generated reports (in the `saved/keccak_f/synthesis` folder) to verify that synthesis completes successfully.

## 2.4 Run Static Timing and Power Analysis

Launch PrimeTime and execute the Static Timing Analysis (STA) script:

```
pt_shell -f ./scripts/pt_analysis.tcl
```

Read the results in the generated CSV file `keccak_f.csv`.

## 3. Change the Clock Period

The target clock period is defined as a TCL variable in the SDC file.

- The clock period is expressed **in nanoseconds (ns)**.
- It is stored in the variable `clockPeriod`.

### Steps

1. Open the following file:

   `./rtl/keccak_f/sdc/keccak_f.sdc`

2. Modify the value of the `clockPeriod` variable.

3. Save the file.

4. Re-run the synthesis & STA flow:

```
dc_shell-xg-t -f ./scripts/synthesis.tcl
pt_shell -f ./scripts/pt_analysis.tcl
```

Observe how tightening or relaxing the clock constraint affects synthesis results (e.g., timing, area, and optimization effort).

## 4. Change the Operating Conditions (PVT Corner)

In this section, you will modify the synthesis scripts to target a different **Process–Voltage–Temperature (PVT)** corner.

### Target change

- **From:** `nom_1.20V_25C`
- **To:** `bc_1.30V_m40C`

### 4.1 Update the clock period

Open:

`./rtl/keccak_f/sdc/keccak_f.sdc`

Set the target clock period (line 5):

```
set clockPeriod "5.0"
```

### 4.2 Update the driving cell for primary inputs

In the same SDC file, modify the driving cell definition (line 21):

```
set_driving_cell -library "CORE65LPSVT_bc_1.30V_m40C.db:CORE65LPSVT" -lib_cell "HS65_LS_BFX7" [all_inputs]
```

This ensures that input timing is characterized using the correct library for the selected PVT corner.

### 4.3 Update the target library

Open the Design Compiler setup file:

./tech/STcmos65/synopsys_dc.setup

Modify the target library configuration (line 15):

```
lappend target_library [lindex $link_library 0]
```

### 4.4 Update the operating conditions

Open the synthesis script:

./scripts/synthesis.tcl

Modify the `set_operating_conditions` command (line 74):

```
set_operating_condition -library  "${target_library}:CORE65LPSVT" bc_1.30V_m40C
```

This command instructs Design Compiler to perform timing analysis and optimization at the selected PVT corner.

### 4.5 Re-run the Synthesis

After completing all modifications, run synthesis & STA again:

```
dc_shell-xg-t -f ./scripts/synthesis.tcl
pt_shell -f ./scripts/pt_analysis.tcl
```

Compare the results with the previous runs and analyze how **clock constraints** and **operating conditions** impact synthesis outcomes.

# LAB 1 – Exercises

## With this lab you learn how to:

- set the environment to run EDA tools,
- synthesize a circuit using the available scripts,
- change design constraints and operating conditions,
- analyze area, timing, power from the synthesis reports.

## Exercise 1 – Logic Synthesis

### 1. Directory Structure Analysis

Analyze the structure and contents of the `WORK_SYNTHESIS` directory:

- **`rtl/`**: contains the RTL description of the KECCAK-f core to be synthesized.
- **`tech/`**: contains all the technology files required for the logic-design flow.
- **`scripts/`**: contains all the TCL scripts used for logic synthesis and post-synthesis Static Timing Analysis (STA).
- **`saved/`**: stores the synthesized netlist and all synthesis and analysis reports.

### 2. Logic Synthesis with Design Compiler

Run the logic synthesis using **Design Compiler** (`dc_shell-xg-t`):

- Execute the synthesis scripts **row by row** by copying and pasting TCL commands one at a time into the `dc_shell-xg-t` shell.
- For each command:
  - Carefully read its documentation using the `man` command.
  - Understand the purpose and effect of the command in the synthesis flow.
- Analyze the generated output files located in `./saved/keccak_f/synthesis`

### 3. Synthesis with Different Timing Constraints

- Synthesize the KECCAK-f core using different clock periods by modifying the variable `clockPeriod` in `rtl/keccak_f/sdc/keccak_f.sdc`.
- Perform synthesis for the following values:
  - `clockPeriod = {1, 2, 3, 4, 5} ns`
- Run synthesis with the following command:
  - `dc_shell-xg-t -f ./scripts/synthesis.tcl`

### 4. Post-Synthesis Analysis and Data Collection

For each synthesized circuit:

- Run PrimeTime to analyze area, timing, power:
  - `pt_shell -f ./scripts/pt_analysis.tcl`
- From the output file `keccak_f.csv`, extract and record the following metrics:
  - **Slack** (ns)
  - **Cell Area** ($\mu m^2$)
  - **Dynamic Power** (mW)
  - **Leakage Power** (µW)
- Report the collected data in the shared spreadsheet at the following link (sheet: **"LAB1 Logic Design"**): SODS 26 Low Power Contest Group List.xlsx

### 5. Results Discussion

Comment on the observed results, highlighting trends and trade-offs among: - Timing constraints - Area - Power consumption

## Exercise 2 – Changing the Operating Conditions

1. Synthesize the KECCAK-f core using different **PVT (Process, Voltage, Temperature)** corners with a clock period of **5 ns**, following the instructions provided in Section 4 of `LAB1-Tutorial.pdf`
2. Repeat the synthesis for the following clock periods:
   - `clockPeriod = {1, 2, 3, 4, 5}` ns
3. For each configuration, collect **area**, **timing**, and **power** metrics with PrimeTime and report them in the shared spreadsheet (see link above).
4. Comment on the results, discussing how PVT variations impact performance, area, and power.