# Lab 4: CORDIC
## (Coordinate Rotation Digital Computer)

# Outline

- Introduction
- Hardware Description
  - I/O Information
  - System Information
- Lab 4 Implementation
  - Precision and Iterations in the CORDIC Algorithm
  - LUT (lookup table)
  - Testbench
- Criteria
  - Grading Policy
  - Requirement & File Format
  - Deadline
  - Commands in Makefile

# Introduction

- What is CORDIC?

  A hardware-friendly algorithm for computing trigonometric, hyperbolic, and other functions.

- How it works ?
  - Iteration Computation:

    Approximate functions using a sequence of shift-and-add operations.

  - Two Modes:
    1. Rotation Modes:

       Rotates a vector by a given angle.
    2. Vector Modes:

       Determines the magnitude and angle of a vector.

  - Precomputed Angles:

    Uses a lookup table for efficient convergence.

# Introduction

- In this lab, we use the CORDIC algorithm in rotation mode to compute sine and cosine values.
- The formulas are shown below.

$$x_{i+1} = x_i - d_i \cdot y_i \cdot 2^{-i} \qquad d_i = \begin{cases} +1, & \text{if } z_i \geq 0 \\ -1, & \text{if } z_i < 0 \end{cases}$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot \theta_i \qquad \theta_i = \tan^{-1}(2^{-i})$$

**You need to generate these values in this lab.**

$$K_n = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \qquad \cos(\theta) \approx K_n \cdot x_n, \quad \sin(\theta) \approx K_n \cdot y_n$$

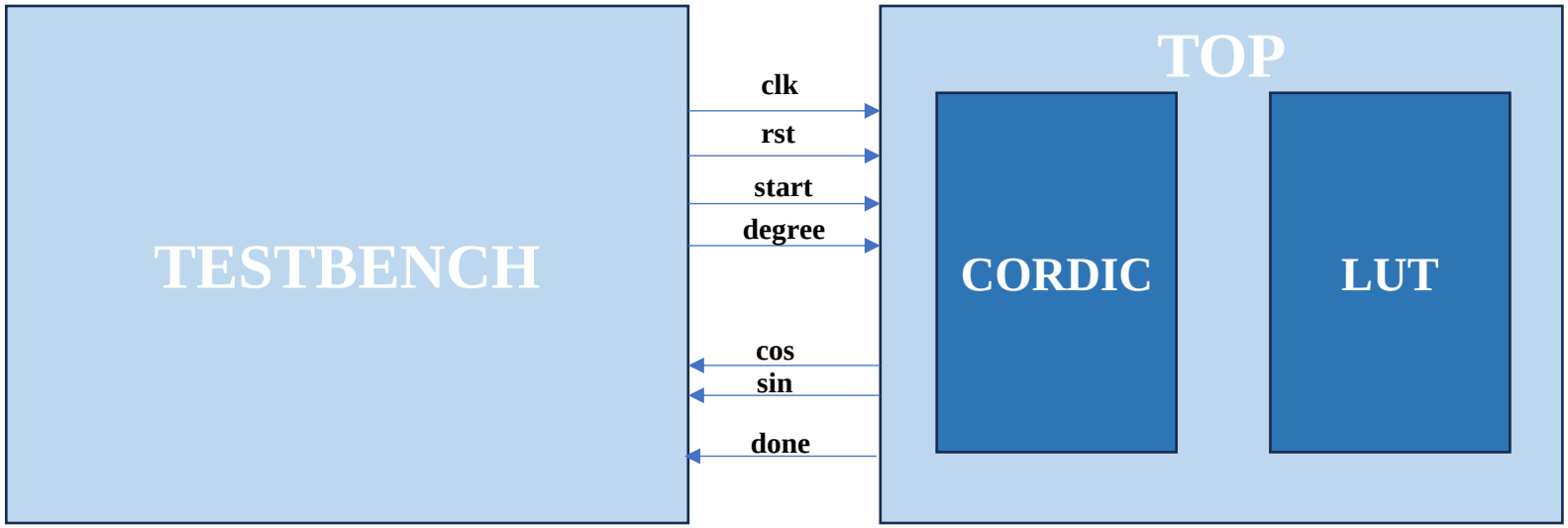
# Hardware Description

- I/O Information (top.v)

| Signal | I/O | Width | Description |
|---|---|---|---|
| clk | I | 1 | Clock signal |
| rst | I | 1 | Active-high reset |
| start | I | 1 | Start signal |
| degree | I | 9 | An unsigned integer ranging from 0 to 359 |
| cos | O | 12 | Signed Q2.10 fixed-point number |
| sin | O | 12 | |
| done | O | 1 | Indicates operation is done |

# Hardware Description

- System Information




**Only top module should follow the I/O specification, as the design of LUT may differ.**

# Lab 3 Implementation

- Precision and Iterations in the CORDIC Algorithm

  In this lab, you should determine the precision and the number of iterations during computation and ensure a bit-by-bit match with the golden reference.

  The formula to generate golden reference is shown below and note that the rounding policy is round to nearest, ties to even.

$$\cos(x)_{Q2.10} = round(\cos(x) * 2^{10})$$

$$\sin(x)_{Q2.10} = round(\sin(x) * 2^{10})$$

  You can use any high-level language to generate the golden values.

# Lab 3 Implementation

- LUT (lookup table)

  You should design LUT yourself, and there are two key points to consider.

  1. Precision of Precomputed Angles and Rotation Factor
  2. LUT Implementation:

     You can follow the context in the textbook on page 231 to model a simple ROM as your LUT, and you must initialize your LUT value within LUT (ROM) module.

# Lab 3 Implementation

- Testbench

  You should write a testbench to test your top module and print the simulation results on the terminal.

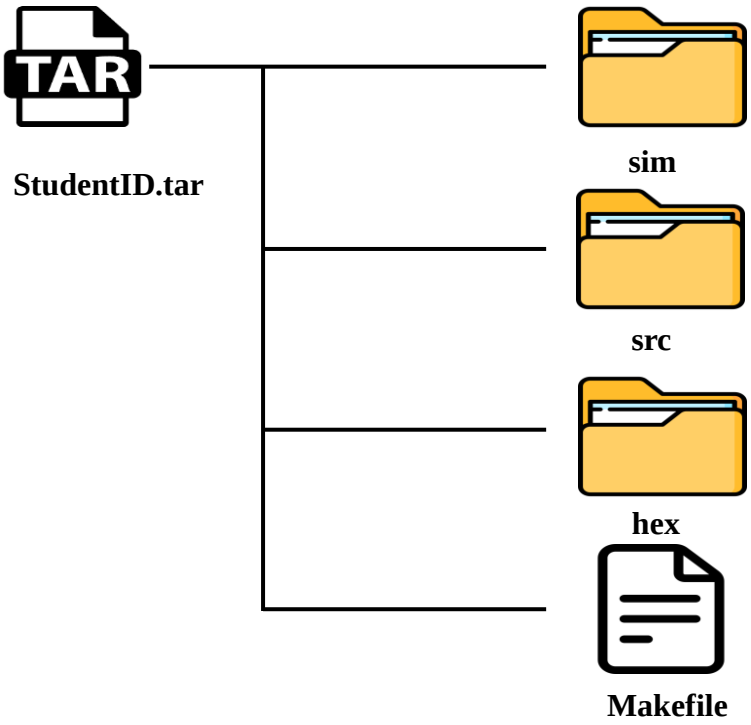
  TA will provide a testbench along with a subset of the test cases initially. Please ensure your top module follows the I/O specification and works correctly with TA's testbench. The full set of test cases will cover the range from 0° ~ 359°.

# Criteria

- Grading Policy (100%)
  - Simulation Pass with TA's Testbench (0° ~ 90°) (60 %)
  - Simulation Pass with Your Testbench (10 %)
  - Simulation Pass with TA's Testbench (0° ~ 359°) (30 %)

# Criteria

- Requirement & File Format




**put your golden or LUT text files here.**

# Criteria

- Deadline: 2025/04/08 (Tue) 14:00
  - Late submissions will receive a partial score as follow:
    - 1 day late -> 80 %
    - 2 day late -> 50 %
    - 3 day late -> 20 %
    - Over 3 days late - > 0 %

# Criteria

- Commands in Makefile

| Situation | Command |
|---|---|
| RTL simulation with your testbench | make vcs |
| RTL simulation with TA's testbench | make ta |
| Launch nWave | make wave |
| Delete waveform files | make clean |
| Compress homework to tar format | make tar |

# Criteria

- Commands in Makefile

  Even if your file path in your Verilog module is correct, the working directory might cause import issues. you can follow the context below to read your LUT or golden text file:

```
string hex_path;

initial begin
    $value$plusargs("hex_path=%s", hex_path);
    $readmemh({hex_path, "/my_rom_data.hex"},  mem);
end
```

```
vcs: | $(wv_dir)
        @cd $(wv_dir); \
        vcs -R -sverilog $(PWD)/$(sim_dir)/lab4_tb.sv -debug_access+all -full64 -debug_region+cell +memcbk \
        +incdir+$(root_dir)/$(src_dir) \
        +notimingcheck \
        +define+$(FSDB_DEF) \
        +hex_path=$(root_dir)/$(hex_dir)
```

**The $value$plusargs command lets you get arguments when starting the simulation.**

# Reference


- introduction-to-the-cordic-algorithm