Solutions - Homework 3

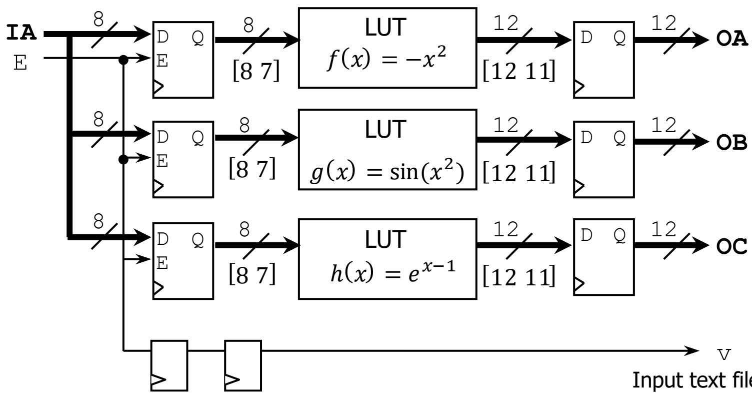
(Due date: March 19th @ 11:59 pm)

Presentation and clarity are very important! Show your procedure!

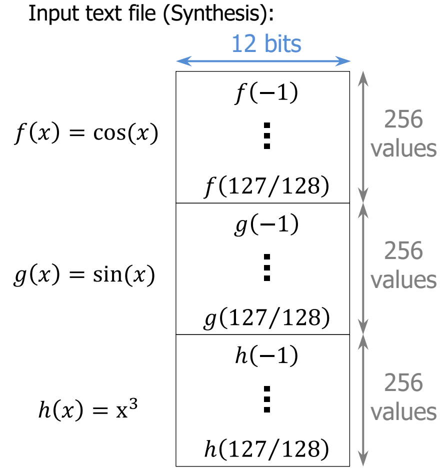
PROBLEM 1 (15 PTS)

- The purpose of this exercise is to explore how to rapidly fill up LUT values and verify the correct operation. Here, you are asked not to write VHDL code, but rather to set up the parameters for it, synthesize, and simulate.

- **LUT approach** for calculating arbitrary functions: We want to implement the following system.
  - ✓ The figure shows three 3 LUT 8-to-12, where each LUT holds the pre-computed results of 3 functions:
    - Input format: [8 7] (signed). Input data range: $[-1,1)$.
    - Output format: [12 11] (signed). Output data range: $[-1,1)$
  - ✓ Input data is captured using the signal E. When the corresponding output data is available, the signal v is asserted.



Input text file (Synthesis):



- The VHDL code and testbench for this system can be found here.
  - `test.vhd` → Top file where all the components are interconnected.
    - `LUT_group.vhd` → File that includes all the LUTs.
      - `LUT_NItoNO.vhd` → File that implement one LUT.
    - `dffe.vhd`
  - `atb_test_sim.vhd` → Testbench

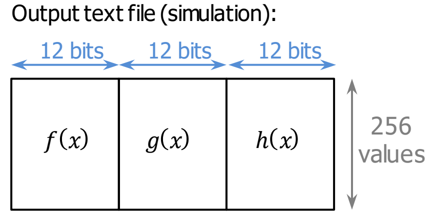
PROCEDURE

- Select the proper parameters:
  - ✓ `test.vhd`: NC=3, NI=8, NO=12, SAME="NO". This generates the system shown in the figure. This file reads the LUT contents from LUT_values8to12.txt file (you need to generate this file).
  - ✓ `atb_test_sim.vhd`: NC=3, NI=8, NO=12. For proper simulation.

- Generate the input text file (Synthesis): LUT_values8to12.txt. The text file contains the pre-computed values (12-bit signed FX numbers). It lists 256 entries per function (as per the figure). An L separator is included between each 256-entry group. You can use the provided MATLAB script (LUTvalGen8to12.m) to generate this file. This script requires the FX converter.

- Create a Vivado project and synthesize your circuit.
- Perform Functional Simulation:
  - ✓ The testbench atb_test_sim.vhd will generate all possible input cases (from 00000000 to 11111111) and write the output results in a text file (out_bench_NI8_NO12.txt). Three 12-bit words are written per output line (256 lines), each 12-bit word represents the output of a different function.
  - ✓ Simulate the circuit until all the 256 input cases are processed. To verify the correct operation of your circuit, compare the values in the text file generated by the Simulation with those in the input text file you generated for Synthesis.

Output text file (simulation):

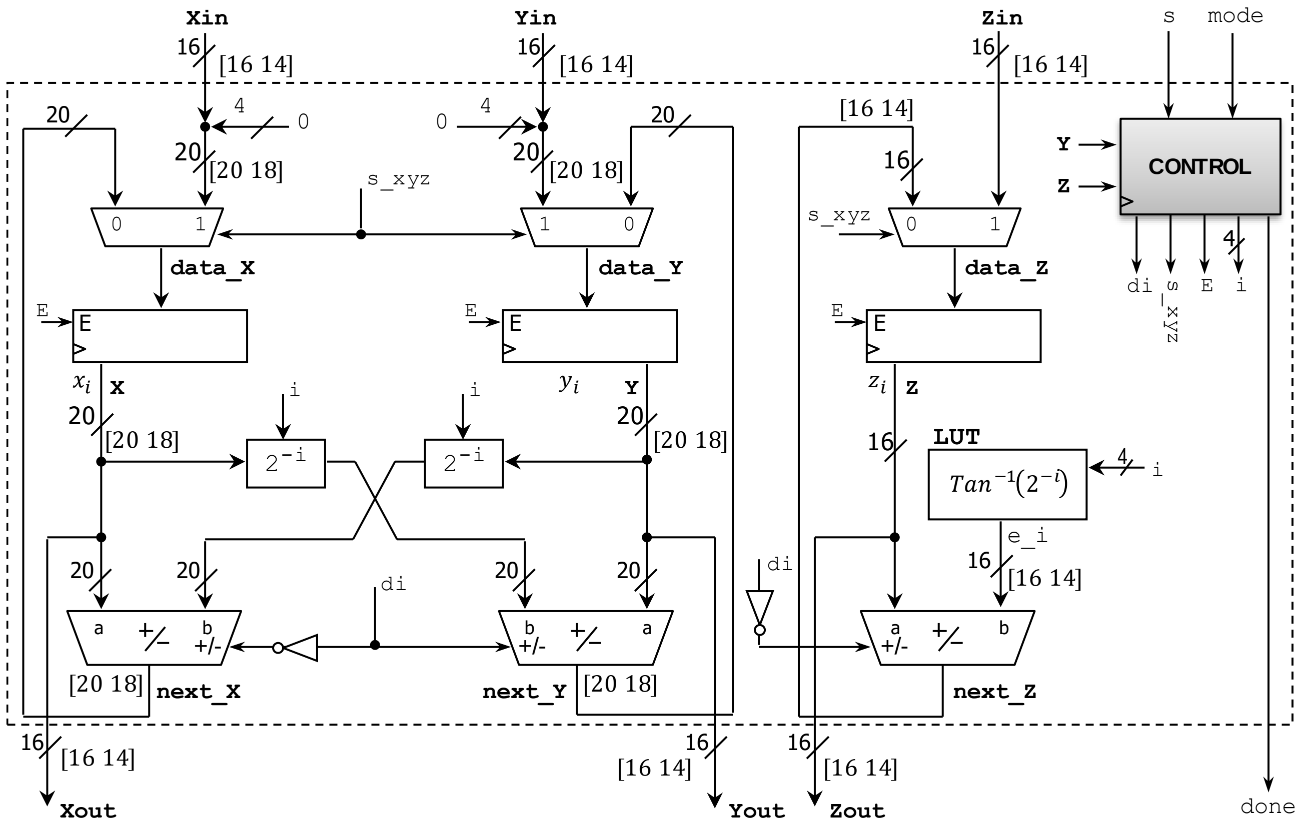


- Upload (as a .zip file) the following files to Moodle (an assignment will be created). DO NOT submit the whole Vivado project.
  - ✓ VHDL code, VHDL testbench: You modified these files by assigning the proper VHDL parameters.
  - ✓ Input text file (LUT_values8to12.txt) and output text file (out_bench_NI8_NO12.txt).

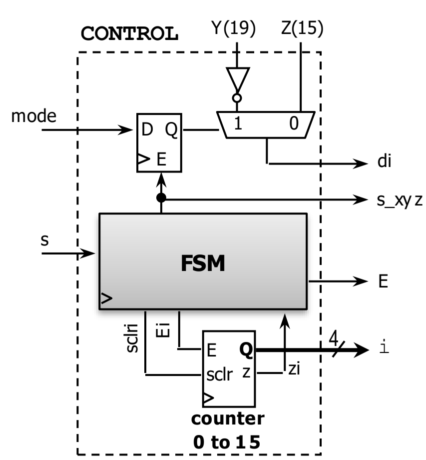
See attached .zip file: `SolutionsHW3p1.zip`.

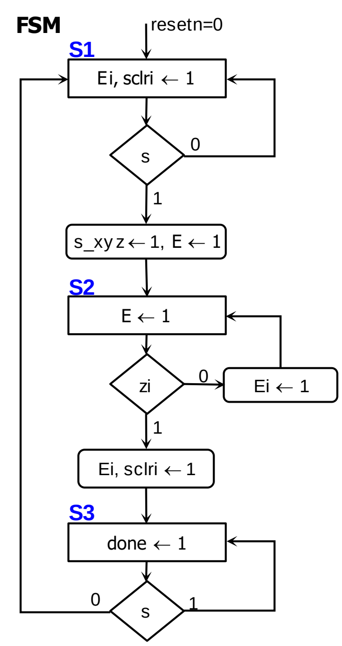
PROBLEM 2 (65 PTS)

- Design (write the VHDL code) for the iterative Circular CORDIC FX architecture with 16 iterations. $i = 0,1,2,3,\dots 15$. x_0, y_0, z_0: initial conditions. $mode$ = '0' → Rotation Mode. $mode$ = '1' → Vectoring Mode. (35 pts)
- **Operation**: When $s = 1$, x_{in}, y_{in}, z_{in} and $mode$ are captured. Data will then be processed iteratively. When data is ready ($done$ = '1'), output results appear in $x_{out}, y_{out}, z_{out}$.
- **Input/Intermediate/Output FX Format**:
  - ✓ Input values: x_{in}, y_{in}, z_{in}: [16 14]. Output values: $x_{out}, y_{out}, z_{out}$: [16 14]
  - ✓ Intermediate values: z_i: [16 14]. x_i, y_i: [20 18]. Here, we use 4 extra bits (add four 0's to the LSB) for extra precision.
  - ✓ We restrict the inputs $x_0 = x_{in}, y_0 = y_{in}$ to $[-1,1)$. Then, CORDIC operations need up to 2 integer bits (determined via MATLAB simulation). For consistency, we use 2 integer bits for all input/intermediate/output data.
- **Angles**: They are represented in the format [16 14]. Units: radians. Pre-compute the values and store them in an LUT.
- **Barrel shifters**: Use the file `mybarrelshift_gen.vhd` with `SHIFTTYPE="ARITHMETIC"` (signed data), `N=20`, `SW=4`, `dir='1'`.



- **Control**: This circuit controls the iteration index i, as well as the internal signals:





SIMULATION (Behavioral)

- To represent the input data and LUT angles in Fixed-Point arithmetic (and vice versa), you can use any online tool or the provided Fixed-Point to decimal converter (my_dec2fx.m, my_fx2dec.m, my_bitcmp.m).
  - ✓ For example, in MATLAB/Octave, you can run the following script that converts some data (e.g. An) as well as the angles $Tan^{-1}(2^{-i})$ into 16-bit numbers in [16 14] signed FX representation:

```
An = 1.6468; % you can use any number here
fx_n = 16; fx_p = 14; type = 's'; % [16 14] signed FX representation
dat = my_dec2fx(An, fx_n, fx_p, type);
disp(dat); % prints a 16-bit value representing An

for i = 1:16
   angle(i) = atan(2^(-(i-1))); % angle (radians)
   e_i(i,1:16) = my_dec2fx(angle(i), fx_n, fx_p, type);
   disp (e_i(i,1:16)); % prints a 16-bit value representing angle(i)
end
```

- A Circular CORDIC MATLAB/Octave model is also available. Make sure to use the 'Basic CORDIC'. This script can be useful to verify the hardware output data. The .zip file contains the following files:
  - ✓ run_examples_cordic.m: This is the top script that contains examples of how to emulate a CORDIC computation given input data (the plotting part only works in MATLAB).
    - Ancillary files (functions): cordic_circular.m, get_scalefactor.m. They implement the CORDIC equations.
  - ✓ my_dec2fx.m, my_fx2dec.m, my_bitcmp.m: This Fixed-Point to decimal converter is helpful to convert the LUT angles and input data to their Fixed-Point representation (binary).

- **First testbench**: Simulate the circuit for the cases shown in the table. You can use $A_n = 1.6468$. Convert the real numbers to the signed FX format $[16\ 14]$. For each case, verify that x_{16}, y_{16}, z_{16} reach the proper values. (10 pts)

$A_n = 1.6468$	Input Data			Expected Output Results		
	x_0	y_0	z_0	x_N	y_N	z_N
Rotation Mode $(mode = 0)$	0	$1/A_n$	$\pi/6$	$-\sin(\pi/6)$	$\cos(\pi/6)$	0
	0	$1/A_n$	$-\pi/3$	$-\sin(-\pi/3)$	$\cos(-\pi/3)$	0
Vectoring Mode $(mode = 1)$	0.8	0.8	0	$A_n\sqrt{0.8^2 + 0.8^2}$	0	$\tan^{-1}(1)$
	0.5	1	0	$A_n\sqrt{0.5^2 + 1^2}$	0	$\tan^{-1}(2)$

- **Second Testbench**: Simulate the circuit reading input values (x_0, y_0, z_0) from input text files and writing output values (x_{16}, y_{16}, z_{16}) on an output text file. (20 pts). Your testbench must:

  - ✓ Read input values (x_0, y_0, z_0) from two input text files (provided):
    - `in_benchR.txt`: Data for Rotation Mode testing.
      20 data points (x_0, y_0, z_0). Data format: [16 14]. Each line per data point written as hexadecimals: |x₀|y₀|z₀|.
      Data set: $x_0 = 0, y_0 = 1/A_n, z_0 = -\pi/2\ to\ \pi/2$. z_0: 20 equally-spaced values between $-\pi/2\ to\ \pi/2$.
      With this data set in the rotation mode, note that $x_{16} \rightarrow -sin(z_0)$, $y_{16} \rightarrow cos(z_0)$.

    - `in_benchV.txt`: Data for Vectoring Mode testing.
      20 data points (x_0, y_0, z_0). Data format: $[16\ 14]$. Each line per data point written as hexadecimals: |x₀|y₀|z₀|.
      Data set: $x_0 = 0.0\ to\ 0.5, y_0 = 1, z_0 = 0$. x_0: 20 equally-spaced values between $0.0\ to\ 0.5$.
      With this data set in the vectoring mode, note that $x_{16} \rightarrow A_n\sqrt{x_0^2 + y_0^2}, z_{16} \rightarrow atan(y_0/x_0)$.

  - ✓ Write output results (x_{16}, y_{16}, z_{16}) on `out_bench.txt`. Data format: $[16\ 14]$, each line per data point written as hexadecimals: |x₁₆|y₁₆|z₁₆|. The output text file should have 40 data points (20 from the rotation mode and 20 from the vectoring mode). Using a handful of data points, verify that your results are correct.

  - ✓ Vivado tips:
    - Make sure that the input text files are loaded as simulation sources.
    - The output text file should appear in `sim/sim_1/behav`.
    - To verify that the output results are correct, you need to represent data as fixed-point numbers. Use `Radix` → `Real Settings` in the Vivado simulator window.
  - ✓ For reference, the MATLAB script cordic_example_ece4710.m generates the input text files and reads the output textfile (out_bench.txt) as specified here. It uses the Circular CORDIC MATLAB/Octave model.

- Submit (as a .zip file) the generated files: VHDL design code, VHDL testbenches, and output text file to Moodle (an assignment will be created). DO NOT submit the whole Vivado Project.
  - ✓ `.zip` file: Include only the `.vhd` and `.txt` files in a single folder (no subdirectories). Points will be deducted otherwise.

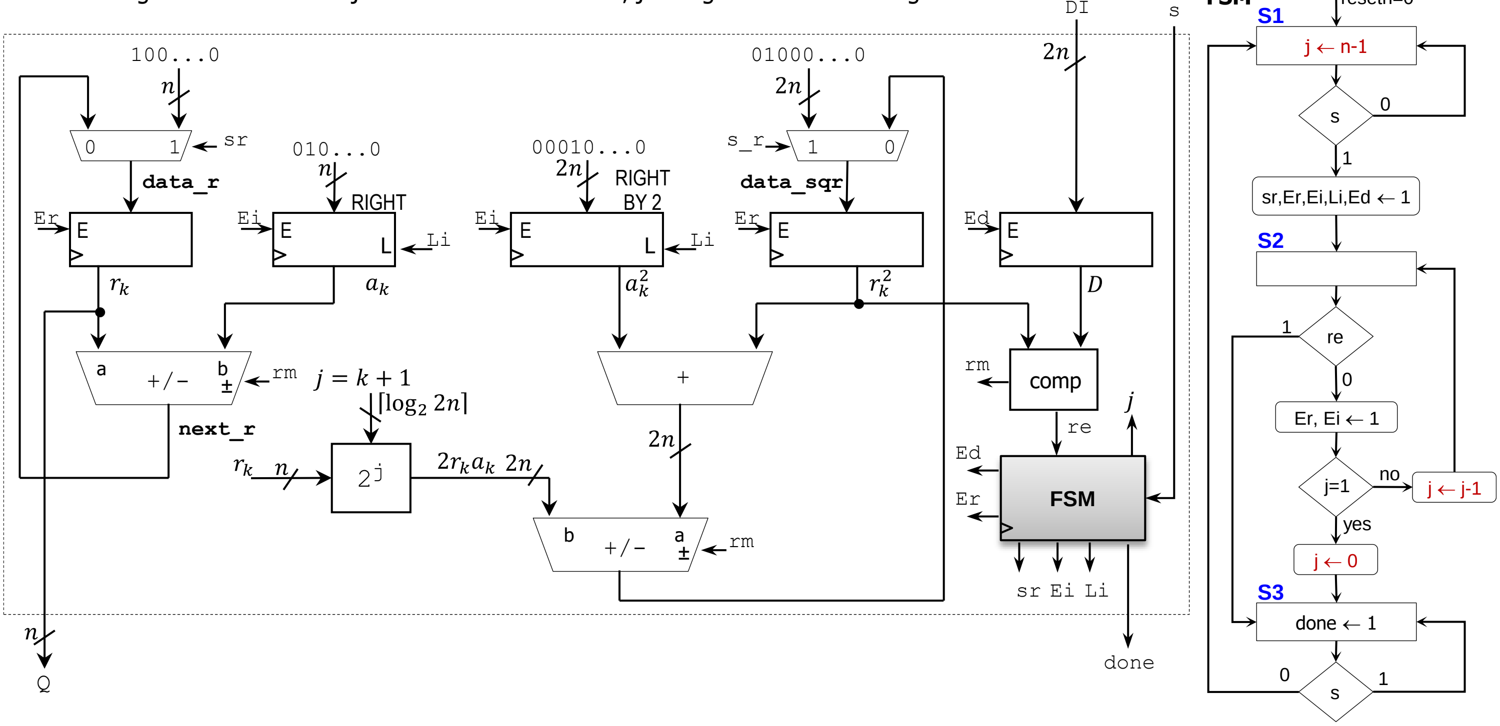
Output text file (`out_bench.txt`): They were obtained with a very precise A_n value. To verify your values, just make sure that they approximate the real values of the functions that the CORDIC algorithm approximates to.

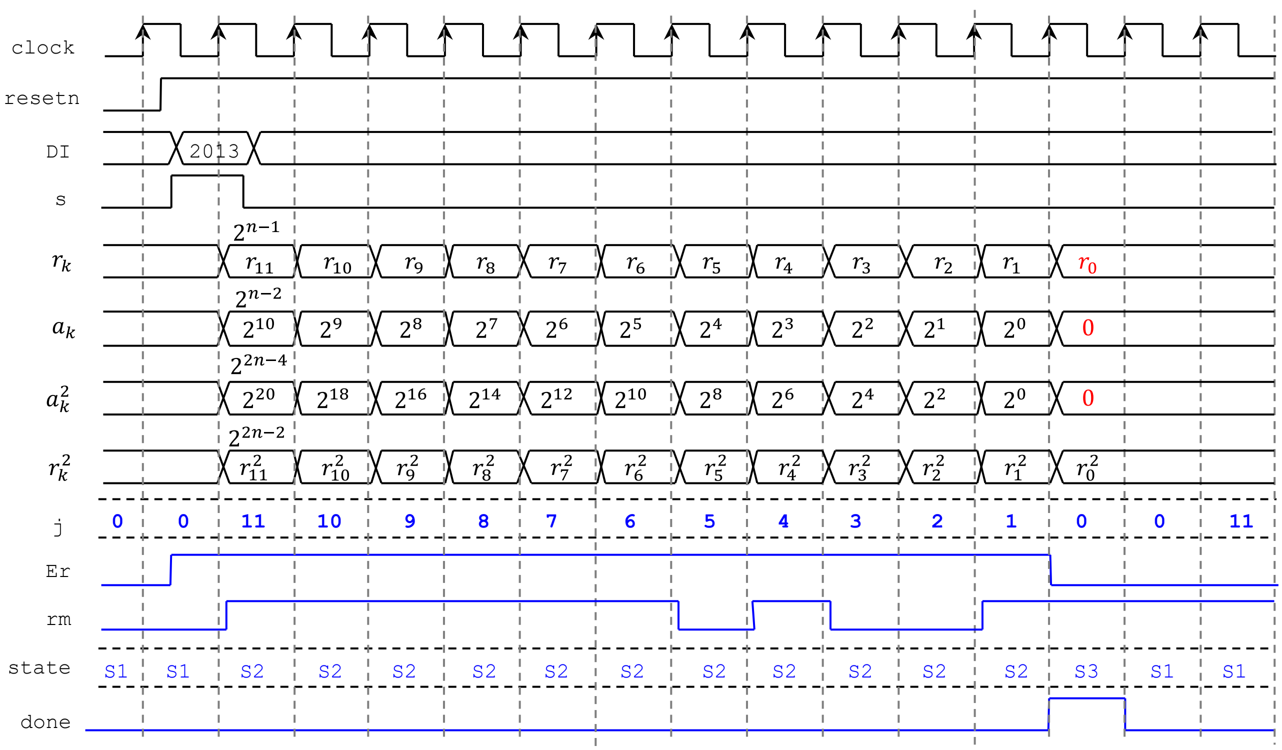
	Rotation	Vectoring
First testbench	DFFD376A0001	773CFFFF3241
	376C2000FFFF	75D5FFFF46DB
1	3FFFFFFCFFFD	6964FFFF6483
2	3F1F0A8B0000	696E000062D7
3	3C8814C60001	698A00006129
4	38471E780000	69B800005F75
5	3281274E0001	69F9FFFF5DD1
6	2B5A2F130002	6A4DFFFF5C2B
7	22FF3594FFFF	6AB200005A7F
8	19B23A9C0000	6B2A000058DB
9	0FB53E0A0003	6BB40000573B
10	05463FC7FFFE	6C4EFFFF55A1

	Rotation	Vectoring
11	FAB93FC70002	6CFB0000540F
12	F04A3E0AFFFD	6DB80000527D
13	E64A3A9B0000	6E86FFFF50F7
14	DCFF3594FFFF	6F6300004F6D
15	D4A52F13FFFE	7051FFFF4DED
16	CD7E274EFFFF	714E00004C7B
17	C7B71E750000	725AFFFF4B03
18	C37714C6FFFF	73750000499B
19	C0DF0A880000	749E0000482F
20	C000FFFAFFFD	75D5FFFF46DB

PROBLEM 3 (10 PTS)

- Complete the timing diagram of the following circuit, which computes integer square root using a binary search approach. $n = 12$. Note that $rm = 1\ if\ r_k^2 > D, else\ 0$, $re = 1\ if\ r_k^2 = D, else\ 0$. Shift registers: serial input is '0'. The value of D is an unsigned decimal. FSM: j is an embedded counter, j changes on the clock edge.





r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
2048	1024	512	256	128	64	32	48	40	44	46	45

PROBLEM 4 (10 PTS)

- Attach your Project Status Report (no more than 1 page, single-spaced, 2 columns, only one submission per group). This report should contain the project title, a brief project description, and the current status of the project, including a block diagram of your system. For formatting, you can use the following template (`Final Project - Report Template.docx`).