

EECE 423/623: Reconfigurable Computing
Project #2: CORDIC Angle IP Block

1. Overview

In Project 1, you developed an AXI-Lite SPI Master IP block and used it to interface with the Digilent PmodACL module containing the ADXL345 accelerometer. You configured the device over SPI, periodically read the six data registers (DATA_{X0}–DATA_{X1}, DATA_{Y0}–DATA_{Y1}, DATA_{Z0}–DATA_{Z1}), and reconstructed signed 11-bit acceleration samples (X, Y, Z). An AXI Timer generated an interrupt every 500 ms. The corresponding interrupt handler performed the following tasks:

- Read the ADXL acceleration values via your SPI Master,
- Computed the tilt angle in software using

$$\theta_{SW} = \text{atan2f}(Y, Z),$$

- Displayed the inclination in radians on the PuTTY terminal.

Project 2 builds directly on this system. The SPI interface, timer, and interrupt infrastructure remain unchanged, but now you must design a dedicated hardware module that uses the CORDIC (COordinate Rotation DIgital Computer) algorithm to compute the same angle using integer fixed-point arithmetic. The updated workflow is:

1. The timer interrupt fires every 500 ms.
2. The PS reads X, Y, Z using the SPI Master (as in Part 1).
3. The PS writes Y and Z to the new `cordic_angle` peripheral in the PL.
4. The CORDIC block computes the angle Θ_{CORDIC} using hardware shift, add, and subtract operations.
5. The PS reads the angle in Q3.12 format (1 sign bit, 3 integer bits, and 12 fractional bits) and displays it on the terminal.
6. For debugging and comparison purposes, the PS also computes $\text{atan2f}(Y, Z)$ in software.

2. Updated System Architecture

Figure 1 illustrates the updated system.

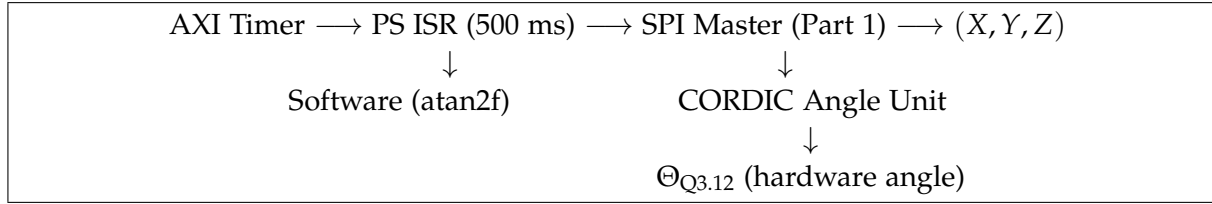


Figure 1: Updated system with the CORDIC angle-computing IP block.

The SPI driver you developed for Project 1 can be reused as is. The only new component is the CORDIC IP block, which uses an AXI-Lite memory-mapped interface to connect to the PS.

3. AXI-Lite Interface for the IP Block

Your hardware module must support the following register map:

Offset	Name	Description
0x00	C_Y	signed 16-bit input Y (bits[15:0])
0x04	C_Z	signed 16-bit input Z (bits[15:0])
0x08	C_CTRL	bit 0 = start; other bits reserved
0x0C	C_STATUS	bit 0 = done; other bits reserved
0x10	C_ANGLE	signed Q3.12 output angle (radians)

The top-level AXI interface files (*_v1_0.v and *_v1_0_S00_AXI.v) must:

- Expose these registers to the PS,
- Feed Y and Z to the CORDIC datapath,
- Assert a start signal upon writing to C_CTRL,
- Raise a done bit in C_STATUS after 13 iterations.

4. Fixed-Point Formats

CORDIC datapath variables. The internal CORDIC vector components will be denoted by (U, V, Θ) , where:

$$U_0 = Z(\text{ADXL output}), \quad V_0 = Y(\text{ADXL output}), \quad \Theta_0 = 0(\text{initial angle value}).$$

The datapath uses:

- U_i, V_i : signed 16-bit integers (Q15.0),
- Θ_i : signed 16-bit Q3.12 (1 sign, 3 integer, 12 fractional bits).

Software–hardware conversion. To convert between floating-point and Q3.12 fixed-point angles:

$$\Theta_{\text{fixed}} = \text{round}(\theta \cdot 2^{12}), \quad \theta \approx \frac{\Theta_{\text{fixed}}}{2^{12}}.$$

5. Angle Look-Up Table (LUT)

Each CORDIC iteration uses:

$$\alpha_i = \arctan(2^{-i}), \quad i = 0, \dots, 12.$$

These constants must be stored in the Q3.12 fixed-point format:

$$A[i] = \text{round}(\alpha_i \cdot 2^{12}).$$

i	α_i (rad)	$A[i]$ (Q3.12)
0	0.785398	3217
1	0.463648	1899
2	0.244979	1003
3	0.124355	509
4	0.062419	256
5	0.031240	128
6	0.015624	64
7	0.007812	32
8	0.003906	16
9	0.001953	8
10	0.000977	4
11	0.000488	2
12	0.000244	1

6. CORDIC Algorithm (Vectoring Mode)

Initialization:

$$U_0 = Z, \quad V_0 = Y, \quad \Theta_0 = 0.$$

On each iteration:

$$d_i = \begin{cases} +1, & V_i \geq 0, \\ -1, & V_i < 0. \end{cases}$$

Updates:

$$U_{i+1} = U_i + d_i (V_i \gg i),$$

$$V_{i+1} = V_i - d_i (U_i \gg i),$$

$$\Theta_{i+1} = \Theta_i + d_i A[i].$$

After 13 iterations:

$$\Theta_{\text{CORDIC}} = \Theta_{13}.$$

This algorithm computes $\text{atan2}(Y, Z)$ for all Y, Z (positive or negative).

7. Application Behavior After Integrating the CORDIC IP

You can reuse the application you developed for Project 1 with minimal changes. The AXI Timer interrupt still fires every 500 ms. The interrupt handler performs:

1. Read (X, Y, Z) from the ADXL using the SPI Master.
2. Save the Y and Z acceleration values to global variables.
3. Set a global flag variable.

The main program function performs the following operations:

1. Check that the Y and Z acceleration values have been updated and clearing the global flag variable.
2. Write the new Y and Z values to C_Y and C_Z .
3. Start the angle calculation by setting the start bit in C_CTRL .
4. Poll C_STATUS until the done bit becomes 1.
5. Read Θ_{CORDIC} from C_ANGLE .
6. Convert to floating point:

$$\theta_{\text{hw}} = \frac{\Theta_{\text{CORDIC}}}{2^{12}}.$$

7. Display θ_{hw} via `xil_printf`.
8. Compute $\theta_{\text{SW}} = \text{atan2f}(Y, Z)$ and print the difference (error).

8. Worked Example

Suppose:

$$Y = 256, \quad Z = 512.$$

The true angle is:

$$\theta_{\text{true}} = \arctan(0.5) \approx 0.46365 \text{ rad.}$$

The expected Q3.12 result is:

$$\Theta_{\text{fixed}} = \text{round}(0.46365 \cdot 2^{12}) \approx 1897.$$

Your CORDIC hardware should output approximately 1897.

9. Deliverables

1. The complete `cordic_angle` Verilog design:
 - top-level `*_v1_0.v`,
 - AXI interface `*_v1_0_S00_AXI.v`,
 - CORDIC datapath implementation using (U, V, Θ) .
2. Updated application program (`.c`) integrating the CORDIC IP.
3. Testbench and waveforms for the cordic IP block demonstrating correct operation.
4. A short report including:
 - updated system block diagram,
 - fixed-point design explanation,
 - verification method,
 - table comparing hardware and software angles.

10. Grading

Correct CORDIC hardware functionality	40%
Correct application code	20%
Testbench and verification quality	20%
Report clarity and completeness	15%
Code readability and documentation	5%

11. Due Date

All project deliverables must be uploaded to Moodle by **11:59 PM on Friday, November 28, 2025**.