Shiraz University

Computer Aided Digital Systems Design

Fourth Assignment

Deadline: 20 January 2026 (30 Dey 1404)

1. **Multi-Mode CORDIC Architecture Design and Implementation-**
Background:
Trigonometric functions and arithmetic operations are fundamental to Digital Signal Processing (DSP). While methods like Taylor series expansion or interpolation are common, they often rely on multipliers that consume significant hardware resources. The CORDIC (COordinate Rotation DIgital Computer) algorithm offers a hardware-friendly alternative, using only shift-and-add operations to perform complex rotations and multiplications.
Objective: The objective of this assignment is to implement a Multimode CORDIC algorithm in Rotation Mode. The design must be flexible enough to support both Circular ($m = 1$) and Linear ($m = 0$) coordinate systems.
Detailed Requirements:

   (a) **Numerical Representation Quantization:** Inputs and outputs should follow a fixed-point representation. It is the designer's responsibility to select an appropriate word length and quantization format (integer vs. fractional bits) to maintain precision throughout the iterations.

   (b) **Accuracy Analysis Iteration Control:** Before hardware implementation, use a high-level simulator (e.g., Python, MATLAB, or C++) to model the algorithm. You must determine the minimum number of iterations required to achieve your desired target accuracy.

   Note: Due to the nature of fixed-point arithmetic and the CORDIC approximation, a small margin of error between the theoretical value and the hardware output is expected and acceptable.

(c) **Hardware Implementation:** Implement the architecture in Verilog HDL. You are required to optimize the design using either Resource Sharing (sequential architecture) or Pipelining.The implementation must support:

i. **Circular Mode:** For calculating sin and cos.

ii. **Linear Mode:** For calculating linear functions (e.g., multiplication).

(d) **Hardware Constants:** Properly manage the scaling factor K and precomputed step values (w_k) within your hardware registers or ROM, ensuring they match your chosen quantization.

(e) **Verification:** Verify your design with a testbench. Your report should compare the **Ideal Results** (from a calculator/simulator) with your **Hardware Results** highlighting that the slight deviation is within the acceptable CORDIC approximation range.

Type	m	w_k	$d_n = \text{sign} z_n$ (Rotation Mode)	$d_n = -\text{sign} y_n$ (Vectoring Mode)
circular	1	$\arctan 2^{-k}$	$x_n \to K(x_0 \cos z_0 - y_0 \sin z_0)$ $y_n \to K(y_0 \cos z_0 + x_0 \sin z_0)$ $z_n \to 0$	$x_n \to K\sqrt{x_0^2 + y_0^2}$ $y_n \to 0$ $z_n \to z_0 + \arctan \frac{y_0}{x_0}$
linear	0	2^{-k}	$x_n \to x_0$ $y_n \to y_0 + x_0 z_0$ $z_n \to 0$	$x_n \to x_0$ $y_n \to 0$ $z_n \to z_0 + \frac{y_0}{x_0}$
hyperbolic	-1	$\tanh^{-1} 2^{-k}$	$x_n \to K'(x_1 \cosh z_1 + y_1 \sinh z_1)$ $y_n \to K'(y_1 \cosh z_1 + x_1 \sinh z_1)$ $z_n \to 0$	$x_n \to K'\sqrt{x_1^2 - y_1^2}$ $y_n \to 0$ $z_n \to z_1 + \tanh^{-1} \frac{y_1}{x_1}$

Figure 1: Forms of the CORDIC Algorithm

2. **Pseudo-Random Number Generation (PRNG) using Galois LFSRBackground:**
A Linear-Feedback Shift Register (LFSR) is a sequential shift register where the input bit is a linear function (typically XOR or XNOR) of its previous state. LFSRs are fundamental components in digital systems for generating pseudo-random sequences, which are essential for applications like cryptography, built-in self-test (BIST), and data scrambling. An LFSR is a deterministic Finite State Machine (FSM); its output stream is entirely dictated by its initial value, known as the seed.A well-chosen feedback function allows the LFSR to produce a sequence with a very long period, appearing pseudo-random. For an N-bit register, if the sequence cycles through all $2^N - 1$ possible states (excluding the all-zero state), it is referred to as a maximum-length LFSR.
Objective: The objective of this assignment is to design and implement a 16-bit Galois LFSR based on the architecture shown in the provided diagram.
Detailed Requirements:

(a) **Architecture Implementation:** Implement a 16-bit Galois-style LFSR in Verilog HDL. Unlike the Fibonacci LFSR, the Galois LFSR places XOR gates between the register stages to improve timing and performance in high-speed hardware.

(b) **Tap Configuration:** Refer to the provided image to identify the feedback taps. Your implementation must use the taps at positions 16, 14, 13, and 11 as shown in the logic diagram.

(c) **Functional Features:**

i. **Reset Seed:** The module must include an active-low asynchronous reset. Upon reset, the register should be loaded with a non-zero seed value to ensure it can exit the all-zero state.

ii. **Enable Signal:**Include a clock enable signal to control the shifting process.

(d) **Verification:** Write a testbench to simulate the LFSR for at least 100 clock cycles.Verify that the register does not get stuck in the all-zero state.Monitor the output and demonstrate that the sequence cycles through different pseudo-random states.

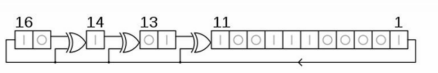

Figure 2: 16 bit Galoris LFSR

3. (a) Display the number 8.75 as a floating point number according to IEEE 754 Single-Precision Standard.

(b) Display all the calculation steps for obtaining the result (using FPGA system). The result should be 5.25×10^{-40}, and the required exponent to represent this value is -6.25×10^{-41}.

(c) Calculate the sum and show all steps of the calculation. To explain the result, first try to clarify how this number can be shown using Q2.5 format.

(d) For part (c), implement the code that adds two numbers the absolute error in Q2.5 format.

Note: All answers must be submitted via Quera The deadline has not been extended.

Late Submission Policy:

Submitting one day late will result in a 25% grade reduction.
Submitting two days late will result in a 50% grade reduction.

Submissions beyond two days will not be accepted under any circumstances.

Deliverables:The following items must be submitted:

1. Verilog source codes
2. Testbench files
3. Documentation (report)

Note: In addition, the documentation must include the following screenshots taken directly from Xilinx Vivado:

1. Successful compilation/synthesis results
2. Simulation waveform results corresponding to the testbench
3. Any other relevant outputs demonstrating correct functionality

Notes:

1. All implementations, simulations, and results must be performed exclusively using Xilinx Vivado.
2. All screenshots must be captured from Vivado.
3. Submitting the assignment using any other software, simulator, or method will result in a grade deduction.

If you have any questions regarding the installation or usage of Xilinx Vivado, you are encouraged to ask for assistance.