1 Simulation results

1.1 Clock cycle count & Performance

Answer:

The simulation results demonstrating the system performance and functional correctness are shown in Fig. 1. The design has been upgraded from an iterative FSM-based approach to a **fully pipelined architecture**, enabling significantly higher throughput.

```
# run -all
# Starting Burst Input...
# Done driving inputs. Sent 721 vectors.
# ----------------------------------------
# Simulation Complete.
# Total Vectors Processed: 721
# Total Errors: 0
# Total Time: 7420 ns
# Throughput: 0.971698 samples/cycle
# SUCCESS: Functionality Verified.
# ** Note: $finish    : ../sv/cordic_tb_pipe.sv(143)
```

(a) Direct Testbench output verifying correctness. The design achieves a throughput of ≈ 1 sample/cycle (0.97 due to initial latency).

```
# UVM_INFO ../uvm/cordic_scoreboard.sv(56) @ 9235: uvm_test_top.env.sb [SCB] PASSED: Checked 721 vectors
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    5
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCB]     1
# [TEST_DONE]     1
```

(b) UVM Report Summary confirming 0 errors and 721 test vectors checked.

Figure 1: Simulation verification results for the Pipelined CORDIC Co-processor.

Fig. 1a shows the final simulation output from the direct testbench. The simulation successfully processed 721 randomized input vectors with a consistent throughput of nearly **1 sample per clock cycle**. The efficiency is driven by the 19-stage pipeline structure in the `cordic.sv` module, which processes multiple coordinate rotations concurrently:

1. **Input Streaming & Buffering:** The module integrates a 16-element FIFO to handle burst inputs. Unlike the FSM approach, the pipelined core manages flow control using valid signals without explicit `IDLE` or `BUSY` states. It accepts new data every clock cycle as long as the pipeline is not stalled.

2. **Pipelined Execution (Latency = 19 Cycles):** The calculation is distributed across a deep pipeline to maximize operating frequency. Data flows through the following stages:

   - **Stage 0-2 (Pre-Processing):** Input latching and range reduction logic (handling $\pm 2\pi$ and $\pm \pi/2$) are split into initial pipeline stages to break critical paths.

- **Stage 3-18 (CORDIC Iterations):** The core performs 16 micro-rotations (iterations 0-15) using 16 instantiated `cordic_stage` modules connected in series. Each stage contains its own registers, allowing 16 different samples to be processed simultaneously.
- **Stage 19 (Output):** The final result is registered and presented at the output.

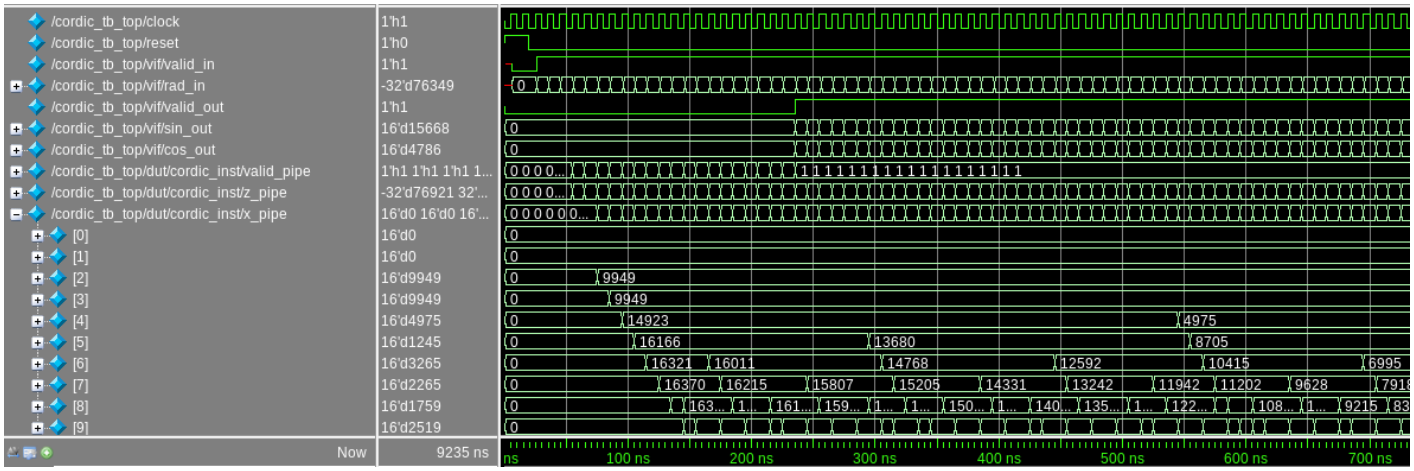

Figure 2: Detailed waveform of the CORDIC Pipeline processing input vectors.

Fig. 2 provides a detailed waveform view of the co-processor's operation. The waveform highlights the pipeline's efficiency:

- **Continuous Data Flow:** Instead of a state machine toggling between states, the waveform shows valid data (`valid_pipe`) propagating through each stage (visible as a diagonal wave of activity).
- **High Throughput:** Once the initial latency (filling the pipeline) is complete, valid output vectors (`valid_out`) are generated on every rising clock edge, demonstrating the design's capability to handle high-bandwidth data streams effectively.

1.2 Errors reported

Answer:

Functional verification was performed using two robust methods to ensure the Pipelined CORDIC correctly calculates sine and cosine values under high-throughput conditions:

1. **Bit-True Comparison (Direct Test):** The pipelined direct testbench (`cordic_tb_pipe.sv`) compares the hardware output against a Golden Reference Model (C++ double-precision implementation). The testbench drives inputs in a continuous burst and checks the corresponding outputs as they emerge from the pipeline. As shown in Fig. 1a, the **SIMULATION PASSED** with a total error count of 0 for all 721 randomized input vectors.
2. **UVM Verification:** The design was also verified using a complete UVM environment. The UVM Scoreboard compared the transaction-level objects from the DUT against the reference model, properly handling the pipeline latency. Fig. 1b confirms that the test passed with **checked 721 vectors** (matching the direct testbench coverage) and **0 UVM Errors**, ensuring robust handling of random inputs and corner cases.

2 Synthesis results

2.1 Maximum frequency

Answer:

The estimated maximum frequency of the design is **207.0 MHz**. As shown in the timing report in Figure 3, the design achieves a minimum clock period of approximately **4.83 ns**. Although the tool reports a negative slack of -0.725 ns against a very high automated target of 243.5 MHz.

Timing Summary			
Clock Name (clock_name)	**Req Freq (req_freq)**	**Est Freq (est_freq)**	**Slack (slack)**
cordic_top|clock	243.5 MHz	207.0 MHz	-0.725
Detailed report		Timing Report View	

Figure 3: Timing summary showing an estimated frequency of 207.0 MHz.

2.2 Registers/LUTs/Logic Elements

Answer:

The resource utilization for the Pipelined CORDIC Co-processor on the Cyclone IV-E FPGA is summarized below in Figure 4:

1. **Total Registers:** 1038
2. **Total Combinational Functions (LUTs):** 1439
3. **Total Memory Bits:** 512

The resource usage reflects the **fully pipelined architecture**:

- **Registers (1038):** The high register count is due to the 19 levels of pipeline registers required to store the x, y, z and valid signals at each stage, enabling 1 cycle throughput.
- **LUTs (1439):** The CORDIC core is fully unrolled (16 physical stages instead of 1 reused stage), using more combinational logic to perform all iterations in parallel.

Area Summary			
LUTs for combinational functions (total_luts)	1439	Non I/O Registers (non_io_reg)	1038
I/O Pins	69	I/O registers (total_io_reg)	0
DSP Blocks (dsp_used)	0 (266)	Memory Bits	512
Detailed report		Hierarchical Area report	

Figure 4: Area summary report detailing the resource usage: 1439 LUTs, 1038 Registers, and 512 Memory Bits.

2.3 Memory utilization

Answer:

The design utilizes **512 bits** of memory. This usage corresponds exactly to the instantiated Input FIFO in `cordic_top.sv`:

- **Input FIFO Configuration:** Depth = 16, Width = 32 bits.
- **Calculation:** $16 \times 32 = 512$ bits.

This memory serves as a buffer to queue incoming 32-bit radian angle requests.

2.4 Multipliers (DSPs)

Answer:

The DSP block utilization is **0**. The CORDIC algorithm is fundamentally designed to perform complex trigonometric calculations using only **shift-and-add** operations.

2.5 Worst path(timing analysis)

Answer:

The final critical path analysis in Figure 5 reveals the timing bottleneck is located between the pipeline registers of Stage 1 and Stage 2, specifically within the Z-datapath (Angle Pre-calculation).

1. **Critical Path Slack:** -0.725 ns (at 243.5 MHz target)
2. **Start Point:** `cordic_inst.z_pipe[1][30]` (Pipeline Reg Stage 1)
3. **End Point:** `cordic_inst.z_pipe[2][1]` (Pipeline Reg Stage 2)
4. **Logic Components:** Range Reduction Logic ($\pm\pi/2$ check and add/sub).

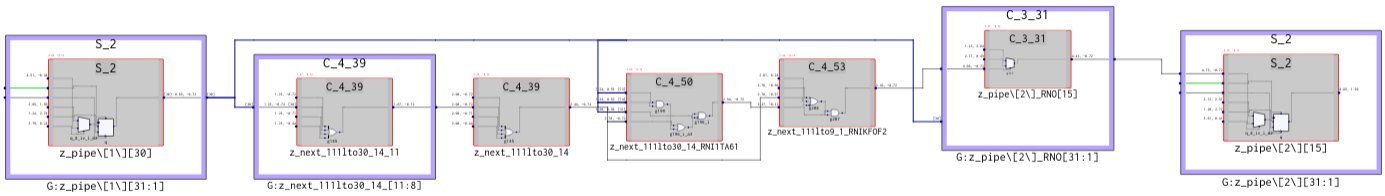

Figure 5: RTL Schematic of the worst timing path: The combinational logic between Pipeline Stage 1 and Stage 2.

The critical path originates from the registered output of Pipeline Stage 1 (`z_pipe[1]`) and propagates to the input of Pipeline Stage 2 (`z_pipe[2]`). As shown in the timing report in Figure 6, the path traverses the combinational logic required for the second step of range reduction. Unlike the FSM design where logic could loop back, the pipeline enforces that all calculation must complete within a single clock cycle (4.83 ns), making this complex range reduction step the limiting factor for the maximum frequency.

Starting Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Arrival Time	Slack
cordic_inst.z_pipe\[1\][30]	cordic_top|clock	dffeas	q	z_pipe\[1\][30]	0.845	-0.725
cordic_inst.z_pipe\[1\][26]	cordic_top|clock	dffeas	q	z_pipe\[1\][26]	0.845	-0.719
cordic_inst.z_pipe\[1\][29]	cordic_top|clock	dffeas	q	z_pipe\[1\][29]	0.845	-0.719
cordic_inst.z_pipe\[1\][25]	cordic_top|clock	dffeas	q	z_pipe\[1\][25]	0.845	-0.713
cordic_inst.z_pipe\[1\][18]	cordic_top|clock	dffeas	q	z_pipe\[1\][18]	0.845	-0.711
cordic_inst.z_pipe\[1\][27]	cordic_top|clock	dffeas	q	z_pipe\[1\][27]	0.845	-0.711
cordic_inst.z_pipe\[1\][17]	cordic_top|clock	dffeas	q	z_pipe\[1\][17]	0.845	-0.705
cordic_inst.z_pipe\[1\][23]	cordic_top|clock	dffeas	q	z_pipe\[1\][23]	0.845	-0.705
cordic_inst.z_pipe\[1\][15]	cordic_top|clock	dffeas	q	z_pipe\[1\][15]	0.845	-0.697
cordic_inst.z_pipe\[1\][8]	cordic_top|clock	dffeas	q	z_pipe\[1\][8]	0.845	-0.503

(a) Starting points (z_pipe[1] Registers).

Ending Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Required Time	Slack
cordic_inst.z_pipe\[2\][1]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_1__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][2]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_2__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][3]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_3__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][4]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_4__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][5]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_5__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][6]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_6__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][7]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_7__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][8]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_8__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][9]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_9__q2	4.029	-0.725
cordic_inst.z_pipe\[2\][10]	cordic_top|clock	dffeas	asdata	z_pipe\[2\]_0_0_10__q2	4.029	-0.725

(b) Ending points (z_pipe[2] Registers).

Figure 6: Synthesis timing report result showing the critical path is between Pipeline Stage 1 and 2.

2.6 Schematic architecture (RTL)

Answer:

The RTL architecture implements a fully pipelined CORDIC processor. The design is organized hierarchically, with a top-level wrapper managing I/O and a core module interconnecting 16 instantiated computing stages. The architecture consists of the following key components:

a. **Top-Level Integration**: `cordic_top` integrates the `input_fifo` and the `cordic` core. It manages the flow control to drive the pipeline, ensuring data is valid at the core input.

b. **CORDIC Core (Pipeline)**: The main core replaces the old FSM with a 19-stage pipeline. Stages 0-2 handle pre-calculation (Range Reduction), while Stages 3-18 (Iterations 0-15) are implemented using 16 instances of the `cordic_stage` module.

c. **CORDIC Stage**: A modular computing unit that performs one iteration of the CORDIC algorithm $(x_{i+1}, y_{i+1}, z_{i+1})$.

Figure 7 illustrates the system-level integration. The `input_fifo` buffers the `rad_in` data, feeding the continuous pipeline.

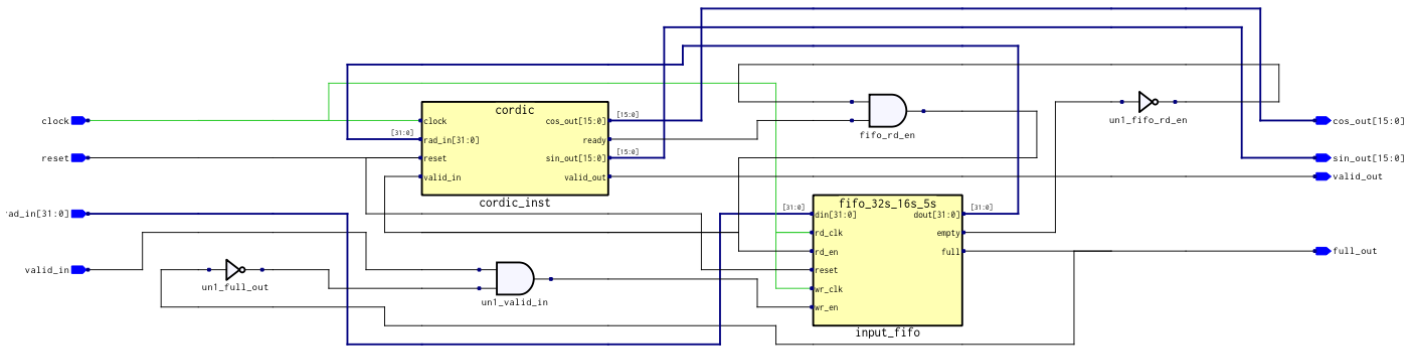

Figure 7: Top-level RTL schematic showing the integration: Input FIFO feeding the Pipelined CORDIC Core.

Figure 8 zooms in on the input stage of the pipeline (Stage 0 to Stage 2). This section is critical for handling range reduction before the iterative CORDIC algorithm begins.

1. **Input Latching (Leftmost):** The registers 'valid_pipe[0]' and 'z_pipe[0]' capture the incoming data from the FIFO.

2. **Range Reduction 1 (Middle):** The logic calculates $z \pm 2\pi$ if the input angle is outside the standard range.

3. **Range Reduction 2 (Right):** The block before the first yellow 'cordic_stage' checks for $\pm\pi/2$ convergence range and initializes the (x, y) coordinates. This setup ensures that the iterative stages receive angles within the valid convergence domain.

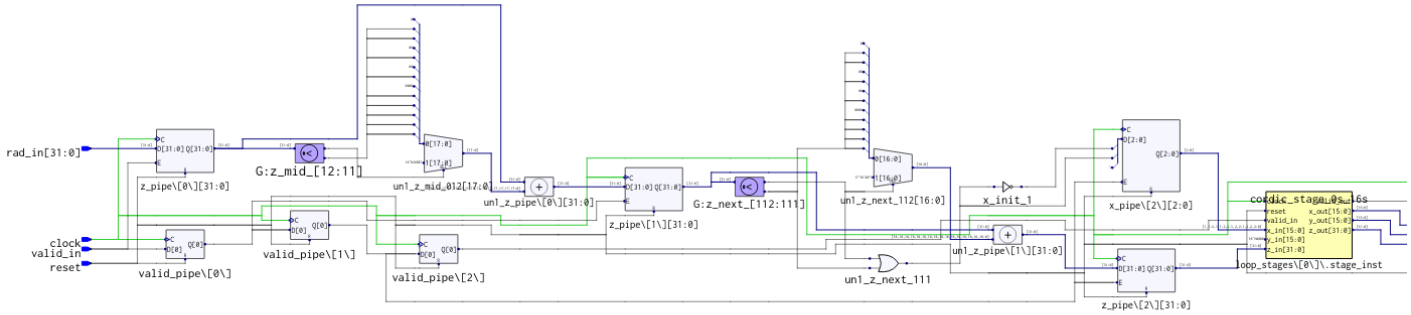

Figure 8: RTL Schematic of the Pre-calculation Front-end (Stage 0-2). It shows the pipeline registers and logic for input latching and two-step range reduction prior to the first CORDIC iteration.

Figure 9 shows the internal structure of the CORDIC core. Instead of a looping FSM, the schematic reveals a linear chain of `cordic_stage` instances connected in series. Data flows from left to right, with each block representing a hardware stage that processes one iteration.

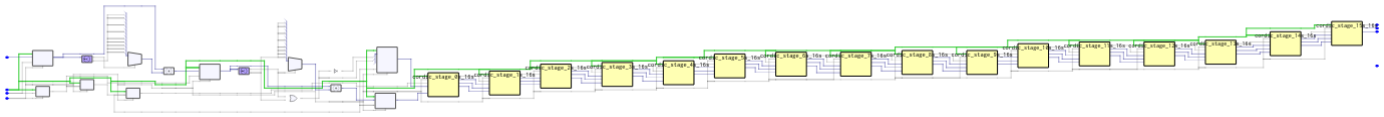

Figure 9: Overview of the CORDIC Core Pipeline. The 16 yellow blocks represent the instantiated `cordic_stage` modules connected in series.

Figure 10 details the internal logic of a single `cordic_stage`. This module implements the fundamental micro-rotation logic:

- **Z-Path (Angle Update):** The comparator checks the sign of z_{in} ('z_in[31]') to determine the rotation direction. The adder/subtractor then updates the angle: $z_{out} = z_{in} \pm \tan^{-1}(2^{-i})$.

- **XY-Path (Coordinate Rotation):** Two barrel shifters (implemented as hard-wired shifts for constant i) and adders/subtractors calculate the new coordinates: $x_{out} = x_{in} \mp (y_{in} \gg i)$ and $y_{out} = y_{in} \pm (x_{in} \gg i)$.

- **Pipeline Registers:** The outputs $(x_{out}, y_{out}, z_{out})$ are registered at the end of the stage to isolate timing paths and enable high-frequency operation.

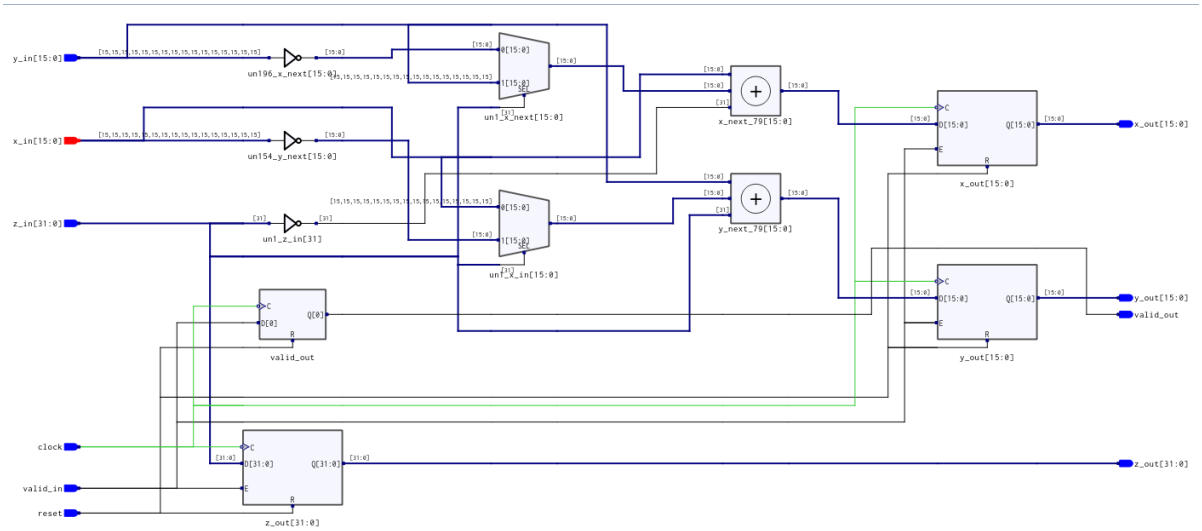

Figure 10: Internal schematic of a single `cordic_stage`. It contains the cross-coupled Adders/Subtractors for X/Y and the angle updater for Z, followed by pipeline registers.

2.7 Performance / Speedup

Answer:

The transition from an iterative FSM architecture to a fully pipelined design has resulted in a dramatic performance increase. We compared the hardware execution time against the reference software and the previous FSM-based hardware implementation.

- **Software Baseline:** Processing 10,000,000 iterations of the CORDIC rotation took **2.128 seconds** on the lab server CPU, resulting in an average execution time of **212.8 ns** per operation.
- **Previous FSM Hardware (Iterative):** The iterative design required 18 clock cycles per calculation. At 134.4 MHz (7.44 ns), the throughput was limited to one output every $18 \times 7.44 \approx$ **133.9 ns**.
- **Current Pipelined Hardware (High-Performance):** The pipelined design runs at a much higher frequency of **207.0 MHz** (Period ≈ 4.83 ns) and produces **one output every clock cycle**.

$$\text{Pipeline Effective Time} = 1 \text{ cycle} \times 4.83 \text{ ns} \approx \mathbf{4.83\ ns} \tag{1}$$

- **Speedup Factor:** Compared to Software:

$$\text{Speedup vs SW} = \frac{212.8 \text{ ns}}{4.83 \text{ ns}} \approx \mathbf{44.0}\times \tag{2}$$

Compared to FSM Hardware:

$$\text{Speedup vs FSM} = \frac{133.9 \text{ ns}}{4.83 \text{ ns}} \approx \mathbf{27.7}\times \tag{3}$$

The hardware implementation achieves a massive **44x speedup** over software. The pipelining strategy itself contributes a **27x improvement** over the iterative approach by enabling concurrent processing of 19 different samples.

2.8 Throughput (Calculations-per-second)

Answer: The throughput capability of the co-processor has increased by an order of magnitude due to the architectural upgrade.

- **FSM Architecture:** Throughput was limited by the sequential nature of the algorithm (Latency = 1 / 18 cycles).

$$\text{Throughput}_{\text{FSM}} = \frac{134.4 \text{ MHz}}{18 \text{ cycles}} \approx \mathbf{7.47 \text{ M Ops/s}} \tag{4}$$

- **Pipelined Architecture:** The design completes one coordinate rotation every **single clock cycle** at the enhanced frequency of **207.0 MHz**.

$$\text{Throughput}_{\text{Pipeline}} = \frac{207.0 \text{ MHz}}{1 \text{ cycle}} \approx \mathbf{207.0 \text{ M Ops/s}} \tag{5}$$

This **207.0 MSPS (Million Samples Per Second)** capability represents a performance leap, making the co-processor suitable for high-bandwidth applications such as Digital Down Conversion in software-defined radios or high-speed radar signal processing.