3x3 Matrix Operations using Verilog

Project Overview

This project implements various matrix operations using Verilog, specifically focusing on 3x3 matrices. It provides modules for matrix multiplication, matrix inversion (for upper triangular matrices), and rotation matrix generation using trigonometric functions. The core of the mathematical operations relies on a `CORDIC` (COordinate Rotation DIgital Computer) module, which performs efficient fixed-point arithmetic calculations for division, sine, and cosine values.

MATLAB Models

To verify and validate the Verilog implementation, MATLAB models were developed for both floating-point and fixed-point calculations. The models simulate the matrix operations and ensure that the results match the hardware outputs in accuracy and behavior.

The key modules in this project include:

- `cordic`: Implements CORDIC-based division and trigonometric calculations.
- `matrix_multiply`: Performs matrix multiplication between two 3x3 matrices.
- `inv_3x3_upper_triangular`: Computes the inverse of an upper triangular 3x3 matrix.
- `rotation_matrix1`, `rotation_matrix2`, `rotation_matrix3`: Generate rotation matrices using trigonometric inputs.
- `matrix_operations_top`: A top-level module integrating all the above components.

Table of Contents

Project Structure

```
.
├── cordic.v
├── matrix_multiply.v
├── inv_3x3_upper_triangular.v
├── rotation_matrix1.v
├── rotation_matrix2.v
├── rotation_matrix3.v
├── matrix_operations_top.v
├── testbenches/
│   ├── tb_cordic.v
│   ├── tb_matrix_multiply.v
│   ├── tb_inv_3x3_upper_triangular.v
│   └── tb_rotation_matrix.v
├── matlab_models/
│   ├── floating_point_model.m
│   └── fixed_point_model.m
├── README.md
└── LICENSE
```

Project Statement

- Write RTL for a 3x3 Matrix inversion block using QR decomposition implemented with CORDIC.
- CORDIC number of iterations: 15 iterations.
- Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit
- Target Frequency: 150 MHz

Design steps

- Problem of the Project
- Way to the Solution: QR Decomposition
- Modeling Floating point on MATLAB
- Modeling Fixed point
- Implementation RTL
- FPGA Flow

Problem Definition

Problem Definition:Inverting a 3x3 matrix is a fundamental operation in many systems, such as signal processing, control systems, and wireless communications. In digital IC design, matrix inversion must be implemented efficiently to meet the high-performance and Time requirements.

Challenges: In hardware, matrix inversion is computationally intensive.

Direct methods (e.g., Gaussian elimination) involve complex divisions, multiplications, and addition operations, making them costly in terms of area, power, and timing in digital circuits.

Challenges of Traditional Methods in Hardware

Resource Intensive: The number of operations grows quadratically with the size of the matrix **($O(n^3)$)**.

- This consumes more hardware resources (adders, multipliers, dividers), leading to large chip area and higher power consumption.

Latency:

- Complex arithmetic operations cause longer delays, impacting real-time system performance.
- Traditional methods can introduce significant delays in processing.
- Division operations in particular are Hard to implement in hardware. (To save area or time)

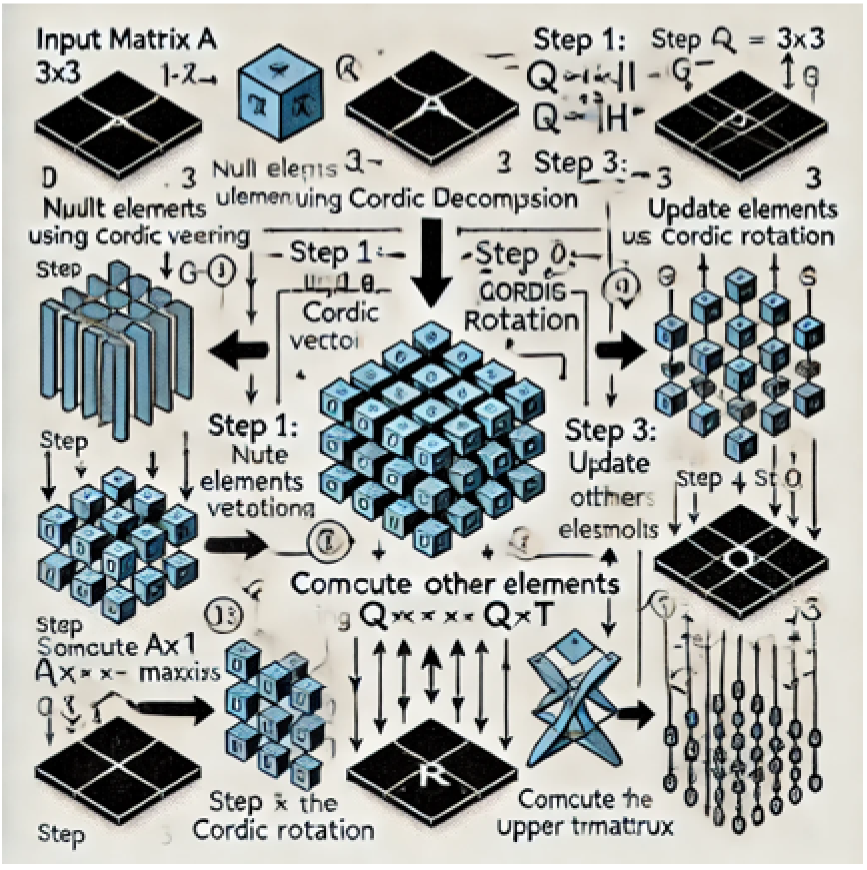
Why QR Decomposition with CORDIC?

CORDIC:

- It avoids the need for multipliers and dividers, using only shifts and adds, which are more hardware-efficient.
- Fixed-point arithmetic allows for a reduction in hardware complexity compared to floating-point implementations.

QR Decomposition:

- Breaks the matrix inversion problem into simpler steps involving rotations.
- Givens rotations (used in QR) can be easily implemented using CORDIC, allowing for efficient triangularization of the matrix.
- Once the matrix is triangular, the inverse is easier to calculate using back-substitution.



Hardware Benefits of CORDIC-based QR Decomposition

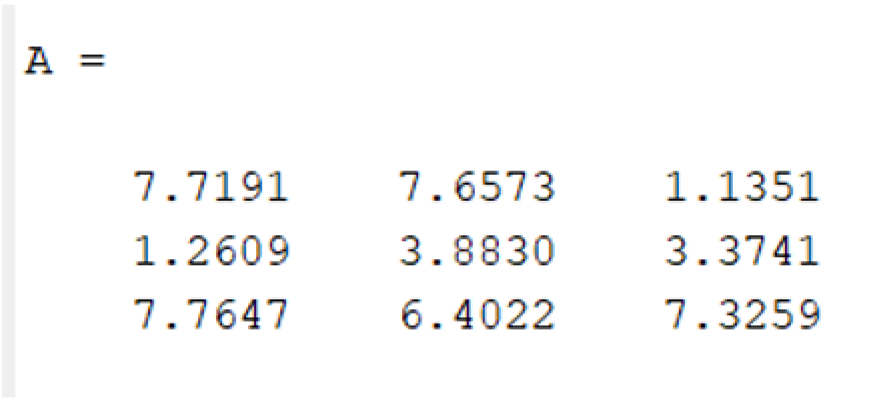
Reduced Complexity:

- No division operations required, making it suitable for hardware implementation.
- CORDIC operations use shift-add algorithms, minimizing the use of multipliers. **Pipelining and Parallelism:**
- The design can be pipelined to achieve high throughput.
- Iterative CORDIC processing allows for parallel execution, optimizing speed. **Area and Power Efficiency:**
- CORDIC is known for being low-power and area-efficient, crucial for digital IC design.
- Ideal for embedded systems where power and area are limited resources.

Floating-Point Model Using MATLAB

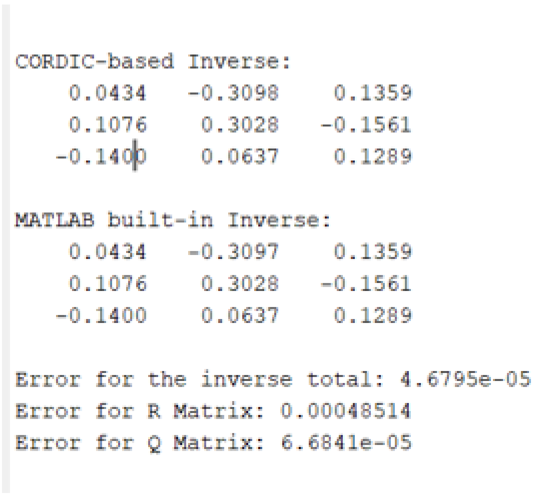
- The MATLAB code implements a floating-point model for performing 3x3 matrix inversion using QR decomposition combined with CORDIC operations.

Inputs



```
A =

    7.7191    7.6573    1.1351
    1.2609    3.8830    3.3741
    7.7647    6.4022    7.3259
```

Outputs



```
CORDIC-based Inverse:
    0.0434   -0.3098    0.1359
    0.1076    0.3028   -0.1561
   -0.1400    0.0637    0.1289

MATLAB built-in Inverse:
    0.0434   -0.3097    0.1359
    0.1076    0.3028   -0.1561
   -0.1400    0.0637    0.1289

Error for the inverse total: 4.6795e-05
Error for R Matrix: 0.00048514
Error for Q Matrix: 6.6841e-05
```

Fixed-Point Representation

- Inputs are converted into fixed-point format using a specified word length (e.g., 16-bit) and fraction length (e.g., 12-bit) for high precision.
- **Resource Efficiency:** Lower hardware resource consumption compared to floating-point arithmetic, making it suitable for embedded systems.
- **Deterministic Precision:** Fixed-point arithmetic guarantees predictable behavior and precision.

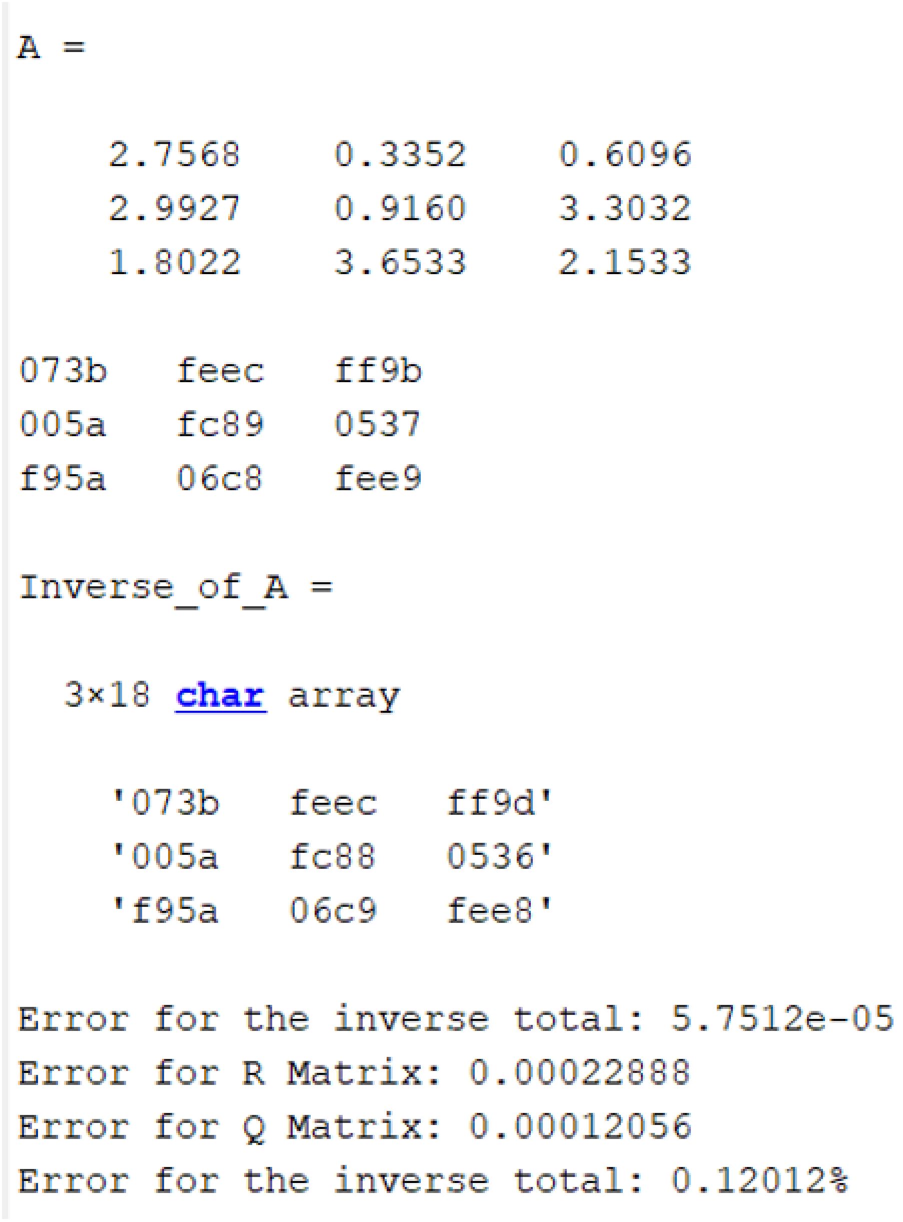
- **Scalability:** The CORDIC algorithm is iterative and scalable for various precision levels.

All matrices and intermediate calculations are performed using fixed-point arithmetic, defined by the following parameters:

- `WORDLEN = 16`: Total bit-width of the fixed-point representation.
- `FRACTION_WIDTH = 12`: Number of fractional bits.

The default scaling factor for inputs is `2^12` (i.e., 12 fractional bits). Outputs are scaled down accordingly to fit the defined format.

Outputs of Fixed Point

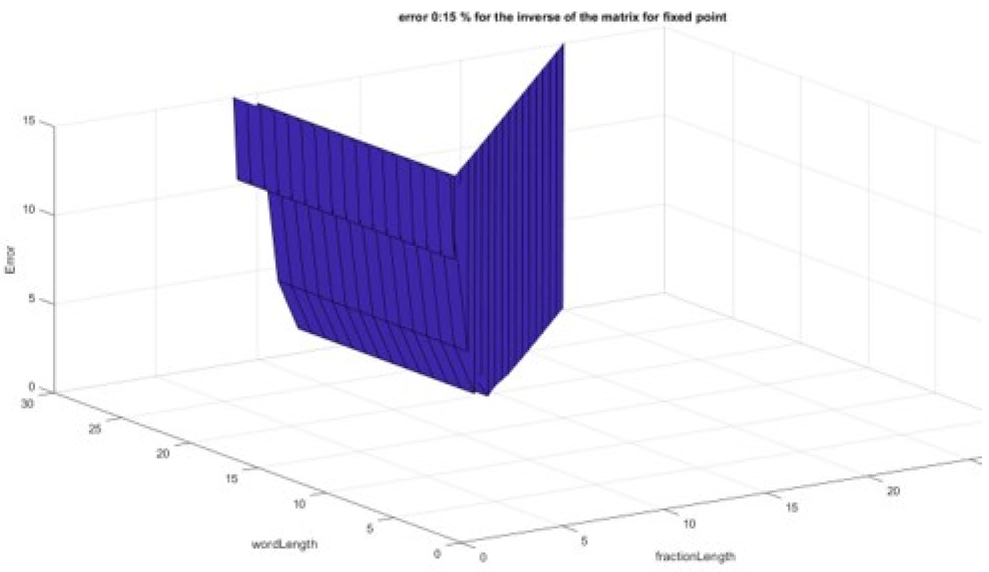


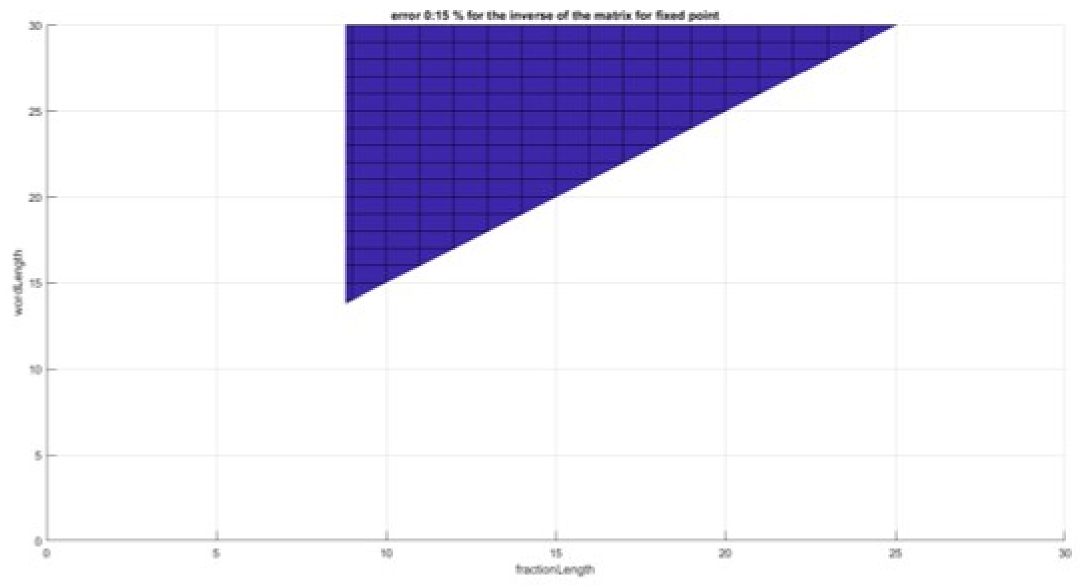
```
A =

    2.7568    0.3352    0.6096
    2.9927    0.9160    3.3032
    1.8022    3.6533    2.1533

073b    feec    ff9b
005a    fc89    0537
f95a    06c8    fee9

Inverse_of_A =

  3×18 char array

    '073b    feec    ff9d'
    '005a    fc88    0536'
    'f95a    06c9    fee8'

Error for the inverse total: 5.7512e-05
Error for R Matrix: 0.00022888
Error for Q Matrix: 0.00012056
Error for the inverse total: 0.12012%
```

Suitable Bit Width

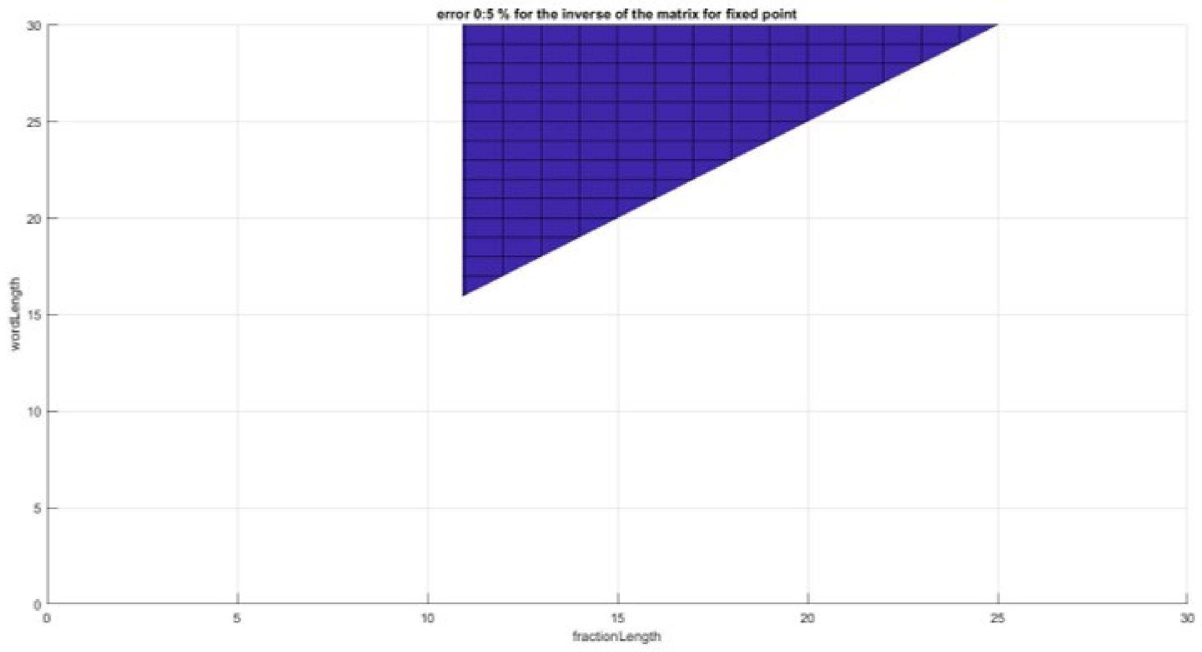
Alt Block

Error 1.5%


error 0:15 % for the inverse of the matrix for fixed point
Error
wordLength
fractionLength


error 0:15 % for the inverse of the matrix for fixed point
wordLength
fractionLength

Error 0.5%



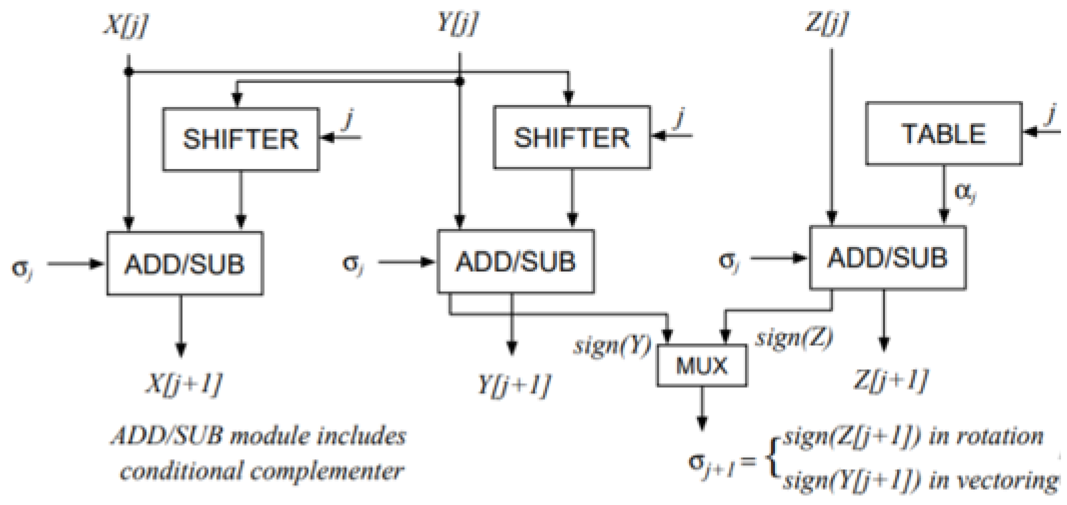
Modules and Functionality

1. **CORDIC Algorithm Implementation (`cordic`)**

What is cordic

- **CORDIC (COordinate Rotation DIgital Computer)** is a hardware-efficient iterative algorithm used in this project to perform various trigonometric, hyperbolic, and vector operations. For the 3x3 matrix inversion using QR decomposition, we applied CORDIC to compute rotations and vector operations needed to perform the **QR decomposition** in fixed-point precision. This allowed for efficient calculation of the Q and R matrices with reduced hardware complexity

Cordic diagram


X[j]
Y[j]
Z[j]
SHIFTER
j
SHIFTER
j
TABLE
j
α_j
σ_j
ADD/SUB
σ_j
ADD/SUB
σ_j
ADD/SUB
sign(Y)
MUX
sign(Z)
X[j+1]
Y[j+1]
Z[j+1]
ADD/SUB module includes conditional complementer
σ_{j+1} = sign(Z[j+1]) in rotation, sign(Y[j+1]) in vectoring

CORDIC General Functionality

- **CORDIC** is an algorithm designed to calculate a wide range of mathematical functions using only additions, subtractions, bit shifts, and look-up tables. It is especially useful for systems where multiplication and division operations are costly.

Cordic Blocks

- Rotational
- Vectoring
- Division

CORDIC Vectoring for Zeroing Elements

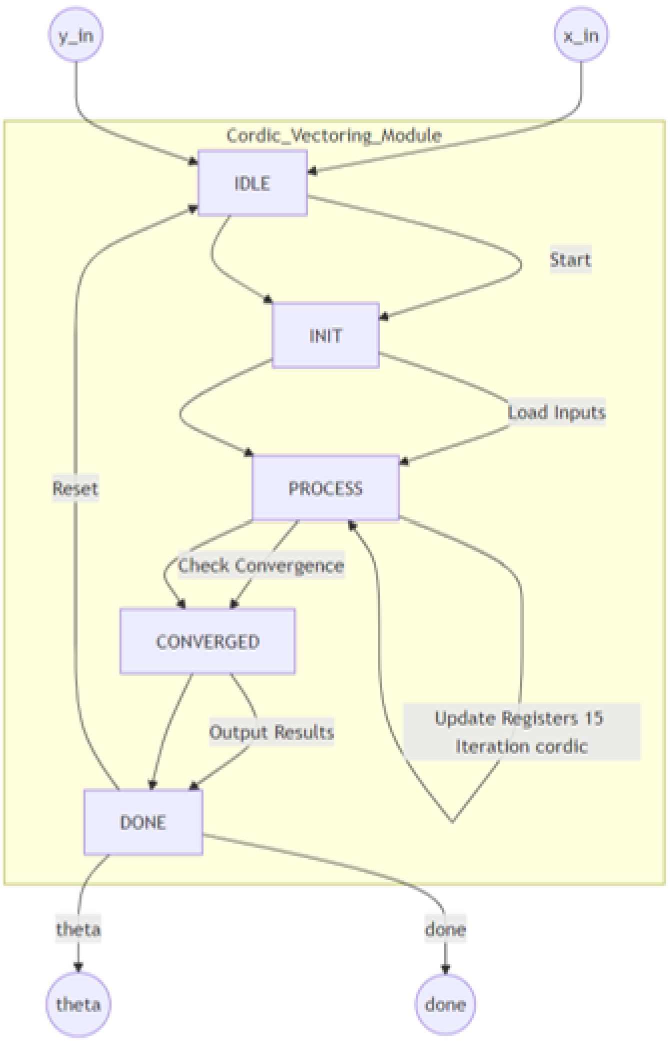
Vectoring Mode: The algorithm rotates the vector towards the x-axis by adjusting x and y coordinates. **Iterations:** 15 steps using shift-add operations with LUT for angle adjustments.

Result:

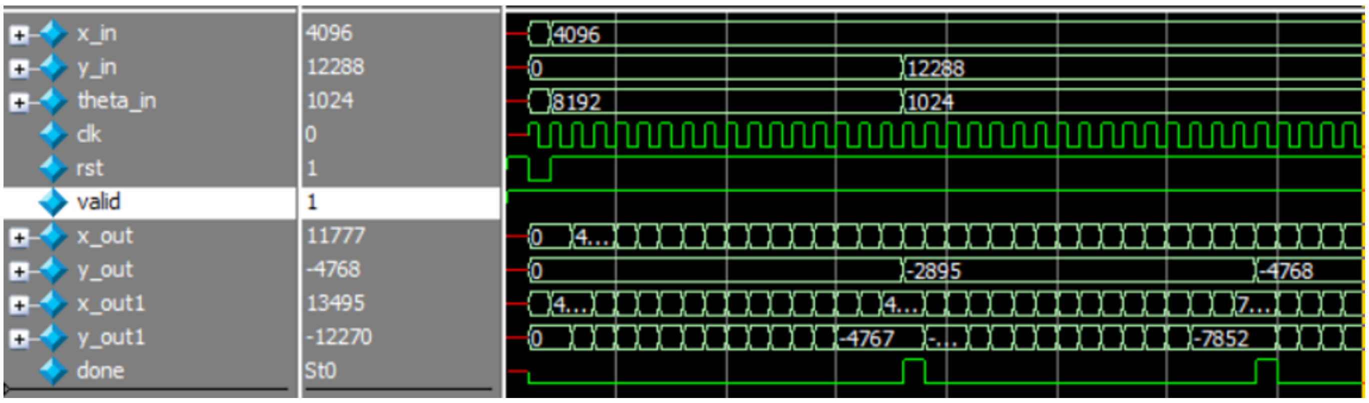
- Provides final magnitude and theta angle after N iterations.
- Fixed-point scaling using a precomputed constant (K).

Optimization

- Theta is Normalized by 2π , for higher accuracy.
- Working with the upper half space +X to improve the performance and fix negative X values Error

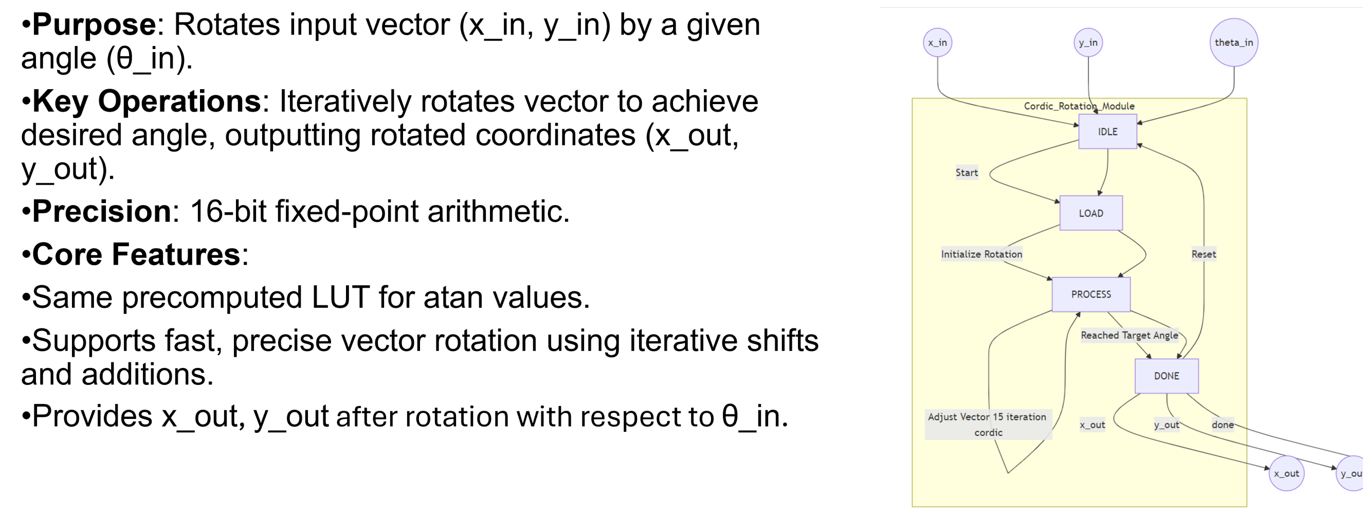


Wave Form



CORDIC Rotation for Matrix Updates

Purpose: Rotates input vector (x_in, y_in) by a given angle (θ_in). **Key Operations**: Iteratively rotates vector to achieve desired angle, outputting rotated coordinates (x_out, y_out). **Precision**: 16-bit fixed-point arithmetic. **Core Features**: Same precomputed LUT for atan values. Supports fast, precise vector rotation using iterative shifts and additions. Provides x_out, y_out after rotation with respect to θ_in.



•**Purpose**: Rotates input vector (x_in, y_in) by a given angle (θ_in).
•**Key Operations**: Iteratively rotates vector to achieve desired angle, outputting rotated coordinates (x_out, y_out).
•**Precision**: 16-bit fixed-point arithmetic.
•**Core Features**:
•Same precomputed LUT for atan values.
•Supports fast, precise vector rotation using iterative shifts and additions.
•Provides x_out, y_out after rotation with respect to θ_in.

CORDIC Division Algorithm

Initialization:

- Numerator (y) and Denominator (x) are set.
- Sign of numerator is tracked separately to apply at the end. **Iterative Process (CORDIC-like method)**:
- Perform N iterations to converge on the quotient.
- If y (denominator) is negative, adjust y and decrement the quotient.
- If y is positive, adjust y and increment the quotient.
- **Step size** scales with the iteration to converge faster. **Final State:** After N iterations, apply the sign to the result and set done. **Key Features:**
- **Word Length**: WORD_LENGTH = 16, allowing for signed 16-bit inputs.
- **Precision**: Uses 12 fractional bits.
- **Error Handling**: Detects zero numerator to prevent undefined division.

Enhancements to CORDIC Division Algorithm

Problem Addressed:

- **Negative Division:** Initially, the algorithm didn't handle negative denominators.

Solution Implemented:

- **Handling Negative Numbers:** Added support for division by negative numbers by extending the sign handling for both numerator and denominator. Now, the algorithm ensures correct results with both positive and negative inputs.

Error Signal out if the Matrix has No inverse and stop the Operations

Problem Addressed:

- **Output Limitation:** The quotient was capped at a maximum of 2 due to the algorithm's structure, leading to incorrect results in certain cases.

Solution Implemented:

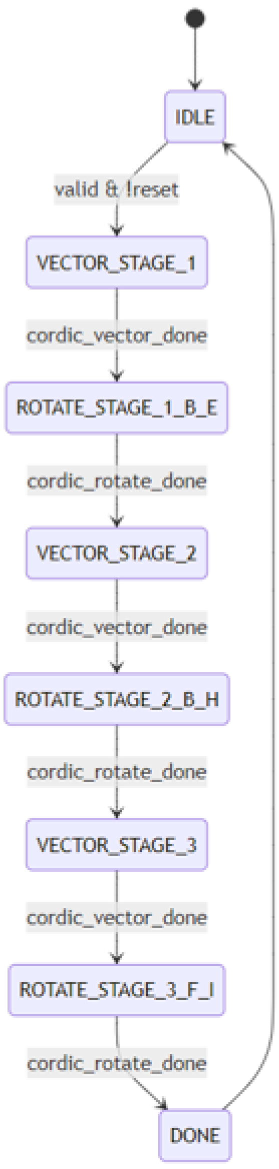
- Adjusted the algorithm to ensure proper scaling of the result by bit-shifting the numerator.
- This correction removes the limitation, allowing the quotient to reach beyond 2, ensuring accurate results for a wider range of input values.

```
z <= z + (16^' h1000 >>> (i));
step_size = (abs_x);
```

TO

```
z <= z + (16'h8000 >>> (i));
assign step_size = (abs_x <<< 3);
```

Cordic Top FSM



R Matrix

$$\begin{tikzcd}
A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \arrow[r] & A_1 = \begin{bmatrix} a' & b' & c' \\ 0 & e' & f' \\ g & h & i \end{bmatrix} \arrow[d] \\
A_3 = \begin{bmatrix} a'' & b'' & c'' \\ 0 & e'' & f'' \\ 0 & 0 & i'' \end{bmatrix} & A_2 = \begin{bmatrix} a'' & b'' & c'' \\ 0 & e' & f' \\ 0 & h' & i' \end{bmatrix} \arrow[l]
\end{tikzcd}$$

Q Matrix

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta_x & -sin\theta_x \\ 0 & sin\theta_x & cos\theta_x \end{pmatrix} \begin{pmatrix} cos\theta_y & 0 & -sin\theta_y \\ 0 & 1 & 0 \\ sin\theta_y & 0 & cos\theta_y \end{pmatrix} \begin{pmatrix} cos\theta_z & -sin\theta_z & 0 \\ sin\theta_z & cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

CORDIC FSM Module

- FSM Structure and Steps:
- IDLE: Initialization of inputs and variables
- VECTOR_STAGE_1: First vectoring stage for - matrix elements a and d
- ROTATE_STAGE_1_B_E: Rotating stage for matrix - elements b and e
- ROTATE_STAGE_1_C_F: Rotating stage for matrix - elements c and f
- VECTOR_STAGE_2: Second vectoring stage for a - and g, calculates sin/cos1
- ROTATE_STAGE_2_B_H: Rotating stage for b and h
- ROTATE_STAGE_2_C_I: Rotating stage for c and i
- VECTOR_STAGE_3: Third vectoring stage for e and - h, calculates sin/cos2
- ROTATE_STAGE_3_F_I: Rotating stage for f and i, - calculates sin/cos3
- DONE: Output results and final matrix inversion

Now , We have R and Q's matrices

*A=Q_3*Q_2*Q_1*R*

*A^(-1)= 〖 〖(Q〗 _3*Q_2*Q_1*R)〗 ^(-1)= 〖(Q *R)〗 ^(-1)*

*A^(-1)=R^(-1)*Q^(-1)=R^(-1)*Q^T*

Blocks we need? -Matrix Multiplier -Upper Triangle matrix Inversion

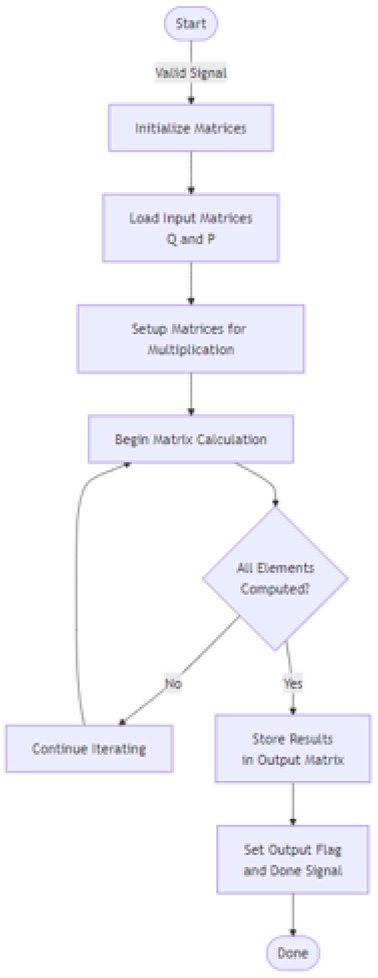
2. **Matrix Multiplication (`matrix_multiply`)**

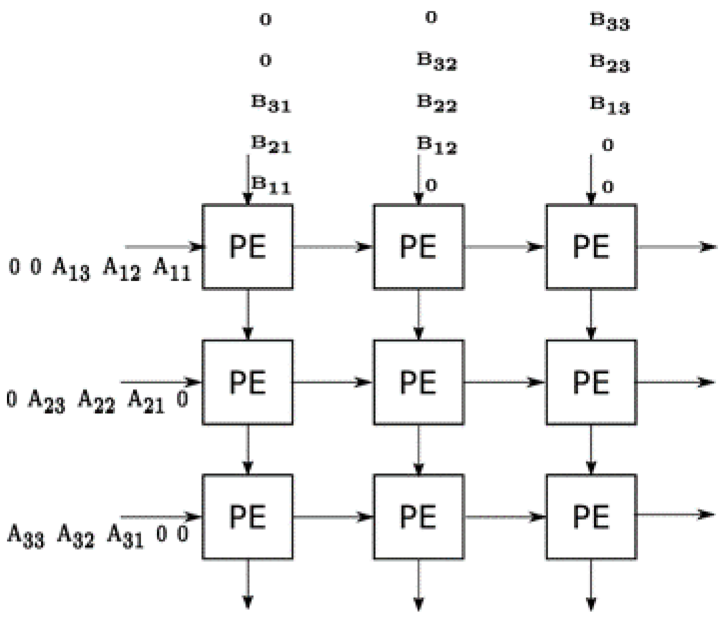
Basic Algorithm

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta_x & -sin\theta_x \\ 0 & sin\theta_x & cos\theta_x \end{pmatrix} \begin{pmatrix} cos\theta_y & 0 & -sin\theta_y \\ 0 & 1 & 0 \\ sin\theta_y & 0 & cos\theta_y \end{pmatrix} \begin{pmatrix} cos\theta_z & -sin\theta_z & 0 \\ sin\theta_z & cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- To calculate any element needed 3 Multiplier and 3 Adder We cannot meet timing

Systolic array algorithm


Start
Valid Signal
Initialize Matrices
Load Input Matrices Q and P
Setup Matrices for Multiplication
Begin Matrix Calculation
All Elements Computed?
No
Yes
Continue Iterating
Store Results in Output Matrix
Set Output Flag and Done Signal
Done



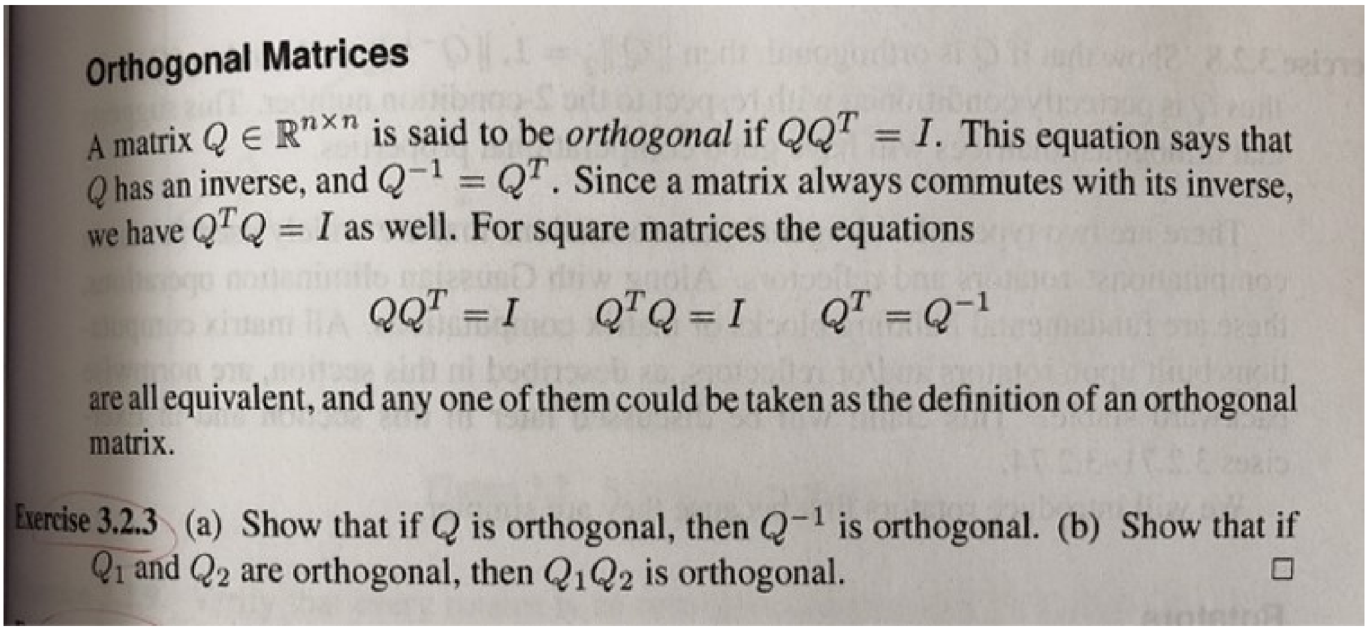
3. **Inversion of 3x3 Upper Triangular Matrix (`inv_3x3_upper_triangular`)**

$$\begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix} \Rightarrow \begin{pmatrix} \frac{1}{a} & \frac{-b}{a \cdot d} & \frac{be-cd}{a \cdot d \cdot f} \\ 0 & \frac{1}{d} & \frac{-e}{d \cdot f} \\ 0 & 0 & \frac{1}{f} \end{pmatrix}$$

4. **Rotation Matrices (`rotation_matrix1`, `rotation_matrix2`, `rotation_matrix3`)**

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta_x & -sin\theta_x \\ 0 & sin\theta_x & cos\theta_x \end{pmatrix} \begin{pmatrix} cos\theta_y & 0 & -sin\theta_y \\ 0 & 1 & 0 \\ sin\theta_y & 0 & cos\theta_y \end{pmatrix} \begin{pmatrix} cos\theta_z & -sin\theta_z & 0 \\ sin\theta_z & cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Orthonormal Matrix Q



Orthogonal Matrices

A matrix $Q \in \mathbb{R}^{n\times n}$ is said to be *orthogonal* if $QQ^T = I$. This equation says that Q has an inverse, and $Q^{-1} = Q^T$. Since a matrix always commutes with its inverse, we have $Q^TQ = I$ as well. For square matrices the equations

$$QQ^T = I \qquad Q^TQ = I \qquad Q^T = Q^{-1}$$

are all equivalent, and any one of them could be taken as the definition of an orthogonal matrix.

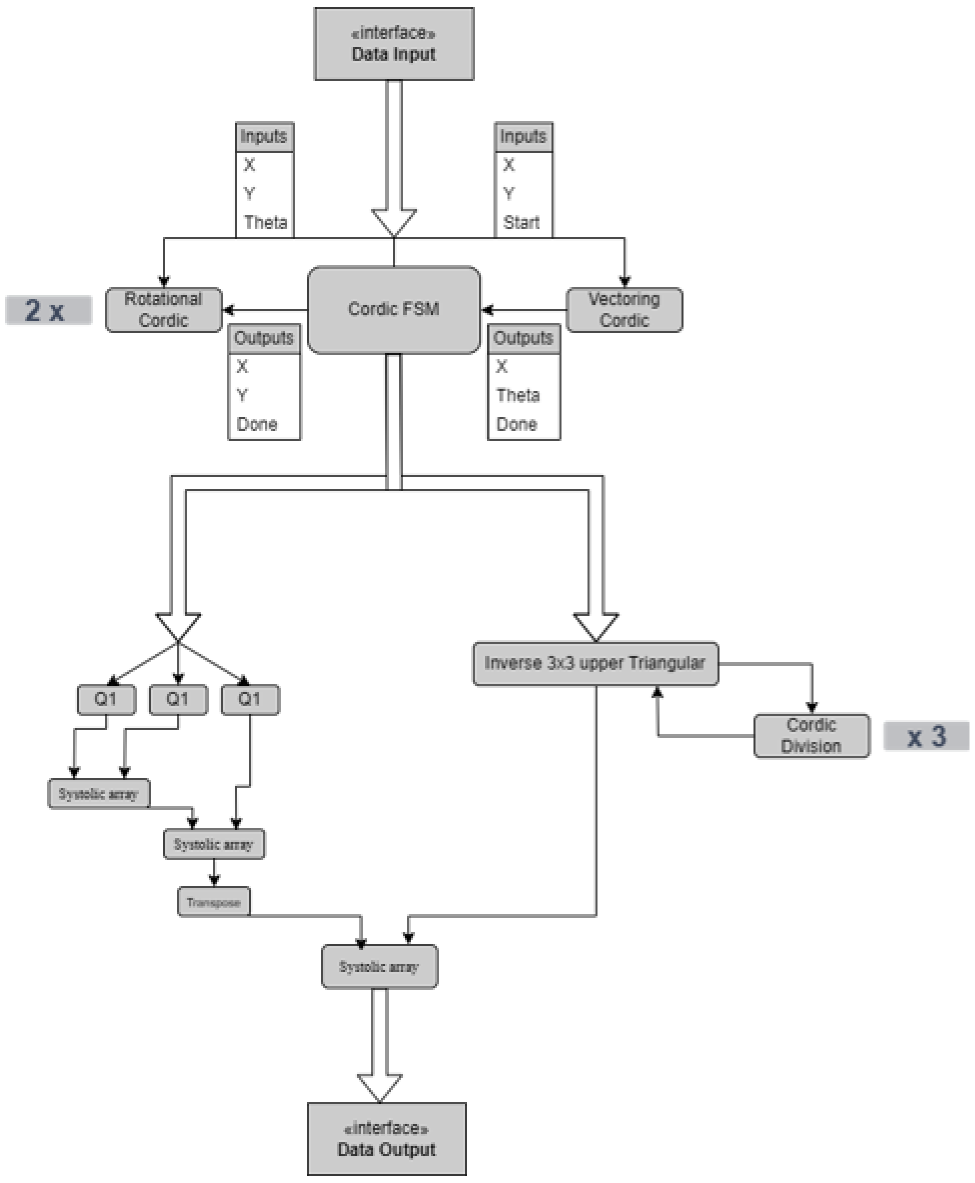
Exercise 3.2.3 (a) Show that if Q is orthogonal, then Q^{-1} is orthogonal. (b) Show that if Q_1 and Q_2 are orthogonal, then Q_1Q_2 is orthogonal. □

And Finally, we will Re use the Systolic Array and get A_inv

`〖A^(-1)= 〖 〖(Q〗 _3*Q_2*Q_1*R) 〗 ^(-1)=R^(-1)*Q〗 ^T=A_inv`

5. **Top Module (`matrix_operations_top`)**

Block Diagram



Description:

The `matrix_operations_top` module serves as the central controller for coordinating the different matrix operations. It accepts control signals and data inputs for the various matrix operations (e.g., matrix multiplication, inversion, rotation matrix generation) and selects the appropriate operation based on the mode input.

Inputs:

- `clk`: Clock signal.
- `rst`: Reset signal (active low).
- `mode`: Selects the matrix operation (e.g., `MULTIPLY`, `INVERSE`, `ROTATE`).
- `Q11, Q12, ..., Q33`: Elements of the first matrix (for multiplication mode).
- `P11, P12, ..., P33`: Elements of the second matrix (for multiplication mode).
- `a11, a12, ..., a33`: Elements of an upper triangular matrix (for inversion mode).
- `angle`: Input angle for rotation matrices.

Outputs:

- `R11, R12, ..., R33`: Resulting matrix elements.
- `valid`: Indicates that the operation has completed.
- `error`: Error flag indicating issues such as division by zero (for inversion mode).

Operation:

The top module uses an internal state machine to control the flow of operations:

- **Mode 1: Matrix Multiplication** - Selects `matrix_multiply` module.
- **Mode 2: Matrix Inversion** - Selects `inv_3x3_upper_triangular` module.
- **Mode 3: Rotation Matrix** - Uses `cordic` for trigonometric computations to construct the desired rotation matrix.

Testbench and Verification

Verification Goals

- Ensure the correct implementation of the matrix inversion using QR decomposition with CORDIC.
- Verify the functional correctness of the QR decomposition process (Q and R matrix generation).
- Confirm the accuracy of the CORDIC-based QR computations.
- Ensure the final matrix inversion is correct.

Key Features to Verify

QR Decomposition:

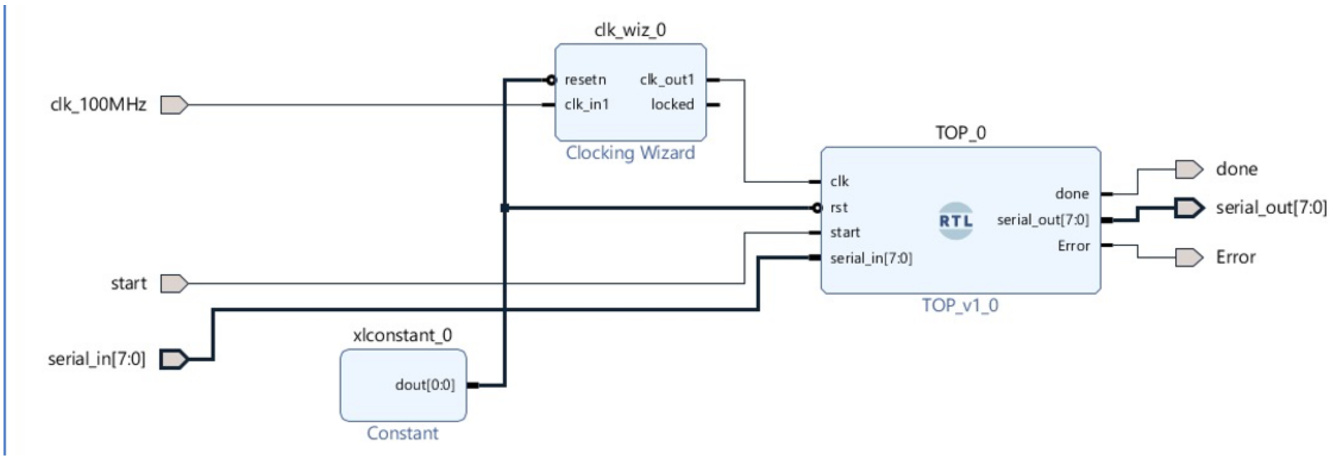
- Verify that the input 3x3 matrix is decomposed into an orthogonal matrix QQQ and an upper triangular matrix RRR.
- Ensure CORDIC is correctly used for rotation in the QR process. **Matrix Inversion:**
- Verify the computation of the inverse using R Inverse and Q matrices and transpose
- Check for correct matrix multiplication steps.
- Check the matrix cannot be inverted like (zero matrix or R matrix has element on diagonal equal zero)
- Check for correct matrix multiplication steps. **CORDIC Operations:**
- Confirm of the CORDIC algorithm.
- Verify the precision of rotation angles and scaling with the model

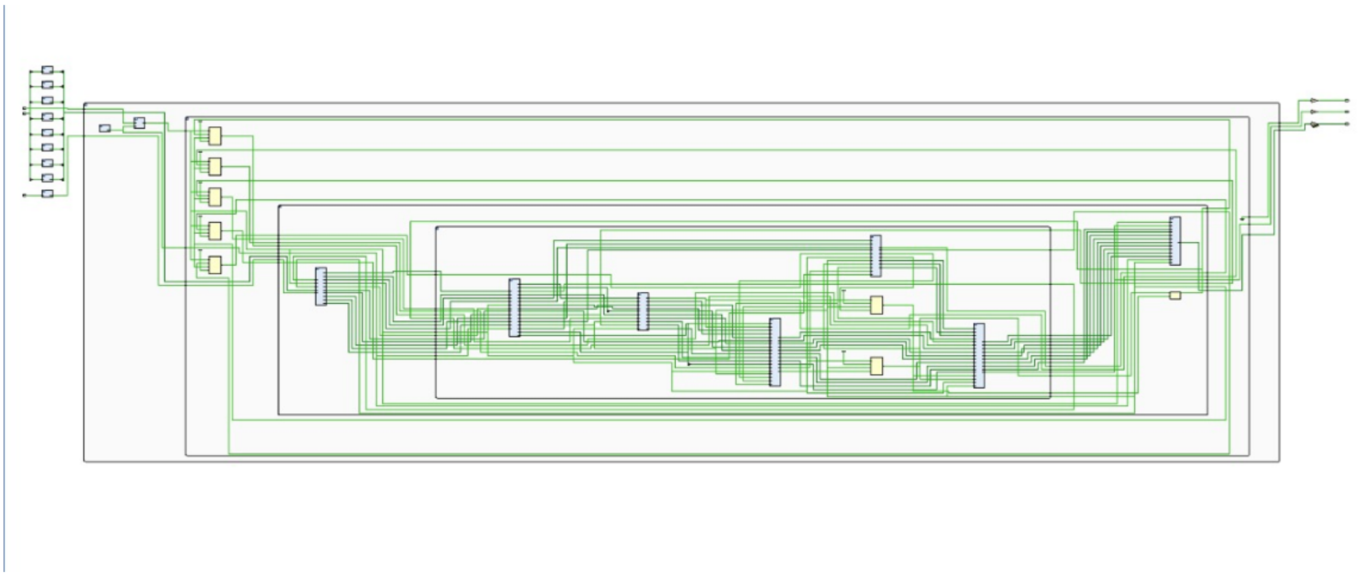
Verification Environment

Testbench Components: Driver: Feeds the input 3x3 matrix to the DUT (Device Under Test). **Monitor:** Monitors output matrices (Q, R, and the inverted matrix). **Checker:** Compares DUT output with the expected result from a golden model (MATLAB

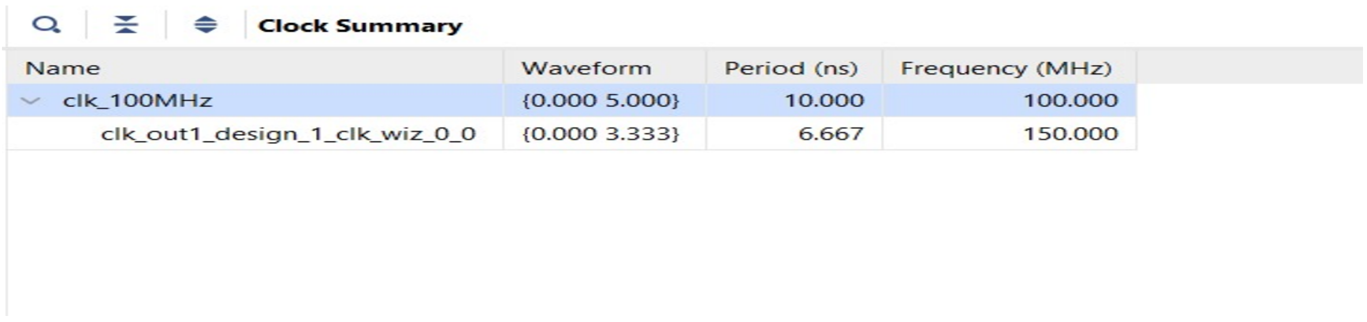
Each module has its own testbench, located in the `testbenches/` directory. The testbenches simulate various input cases and edge conditions. The `matrix_operations_top` testbench, `tb_matrix_operations_top.v`, verifies the integration and interaction of all modules.

Results of FPGA



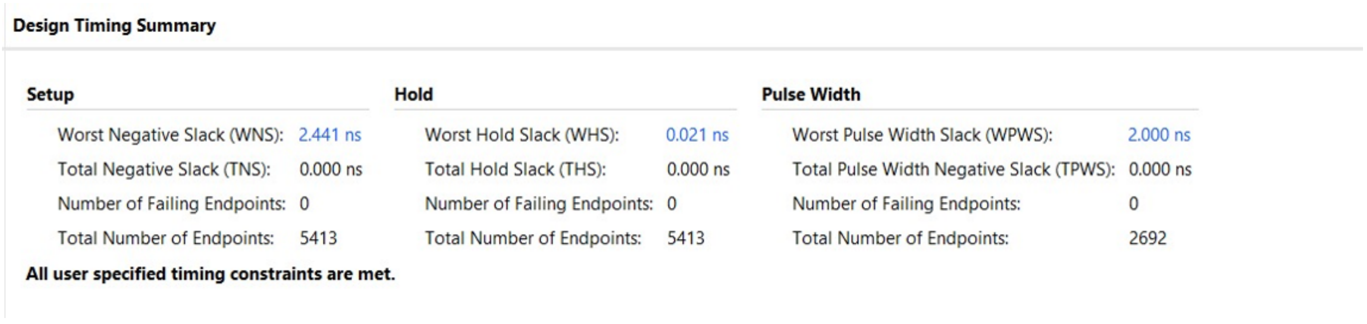


**Freqancy 150M



Clock Summary

Name	Waveform	Period (ns)	Frequency (MHz)
clk_100MHz	{0.000 5.000}	10.000	100.000
clk_out1_design_1_clk_wiz_0_0	{0.000 3.333}	6.667	150.000

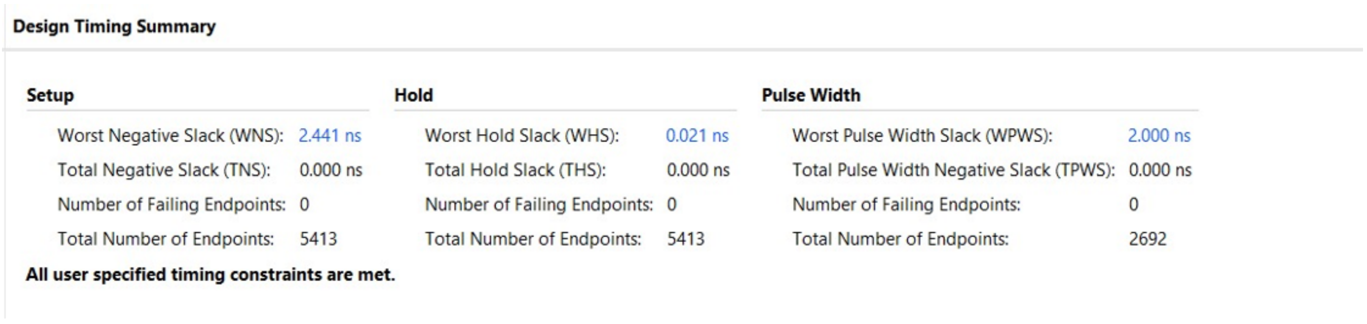


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.441 ns	Worst Hold Slack (WHS):	0.021 ns	Worst Pulse Width Slack (WPWS):	2.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5413	Total Number of Endpoints:	5413	Total Number of Endpoints:	2692

All user specified timing constraints are met.

Utilization

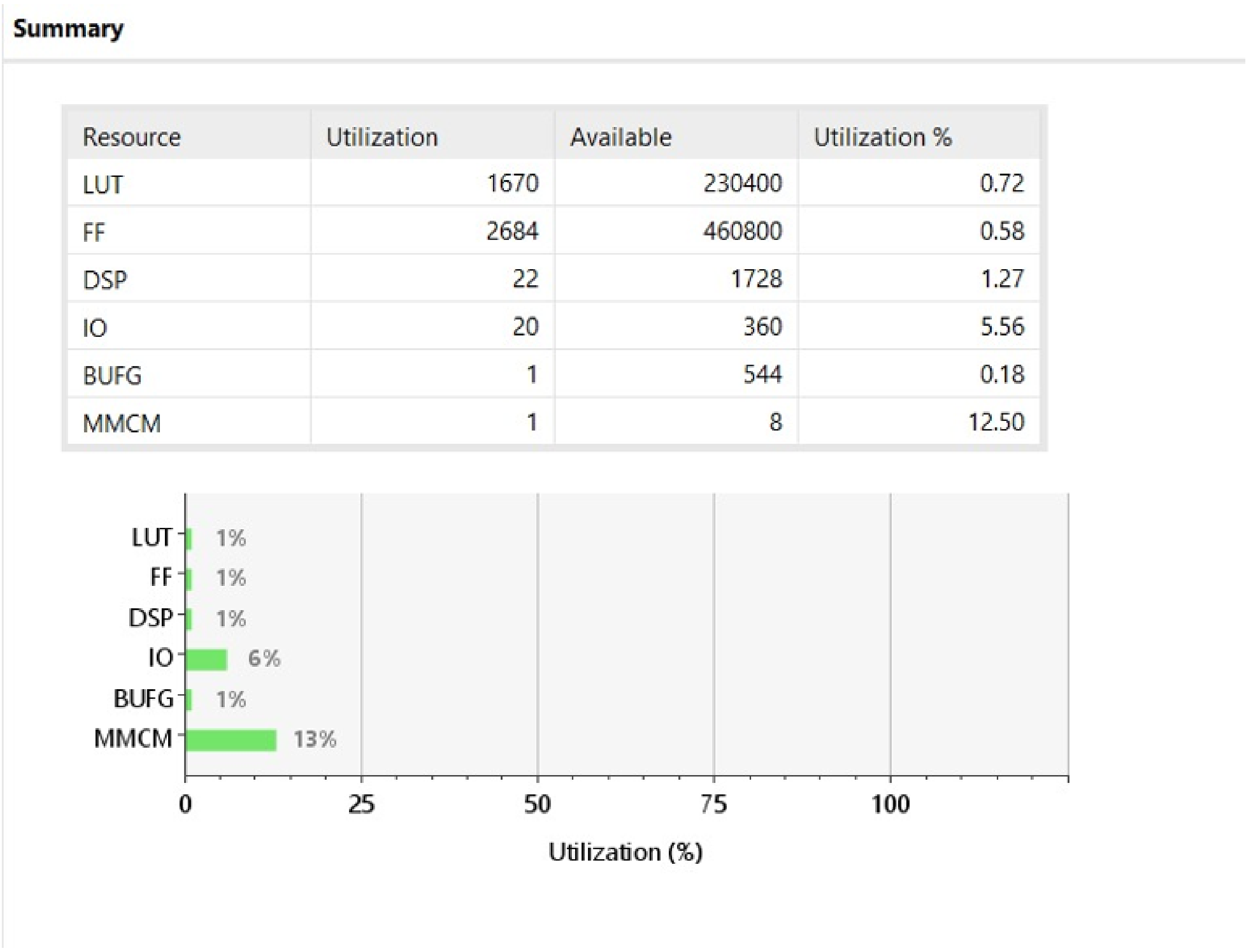


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.441 ns	Worst Hold Slack (WHS):	0.021 ns	Worst Pulse Width Slack (WPWS):	2.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5413	Total Number of Endpoints:	5413	Total Number of Endpoints:	2692

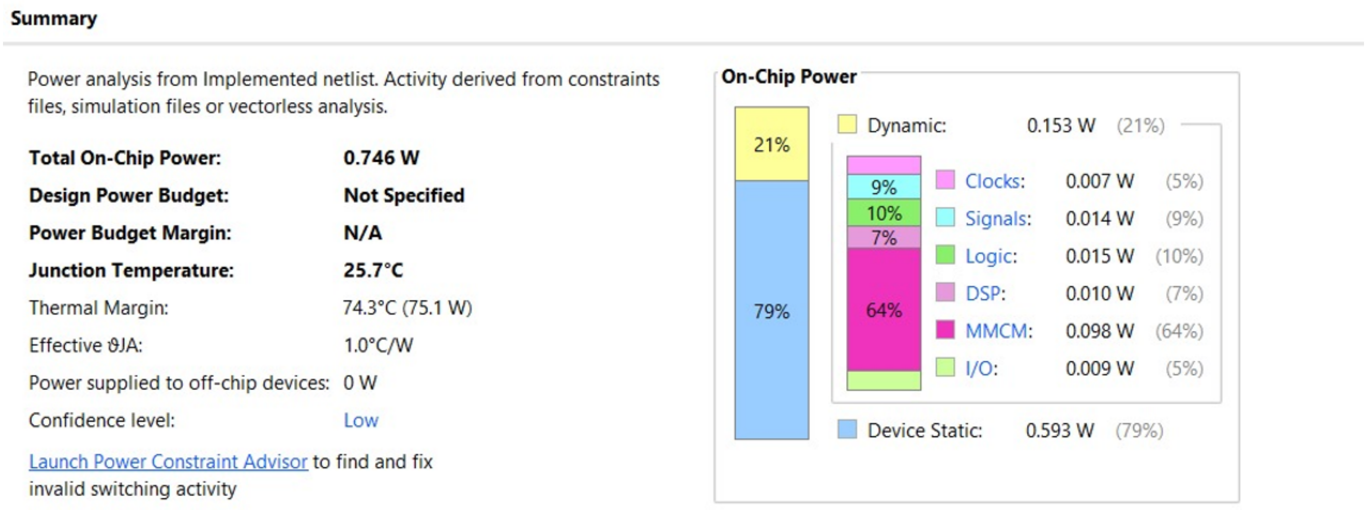
All user specified timing constraints are met.

Power



Summary

Resource	Utilization	Available	Utilization %
LUT	1670	230400	0.72
FF	2684	460800	0.58
DSP	22	1728	1.27
IO	20	360	5.56
BUFG	1	544	0.18
MMCM	1	8	12.50



Summary

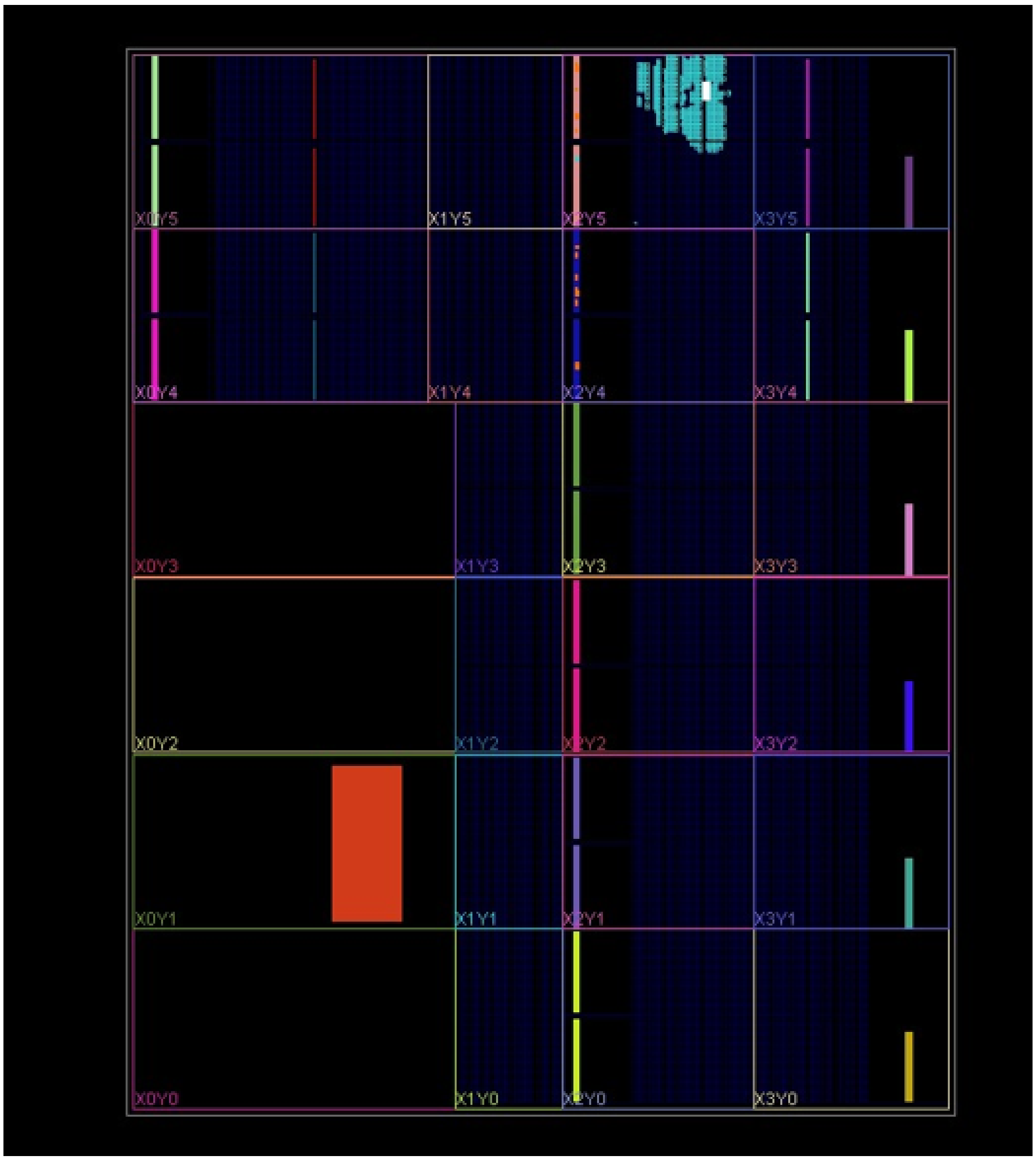
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.746 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.7°C**
Thermal Margin:	74.3°C (75.1 W)
Effective ϑJA:	1.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

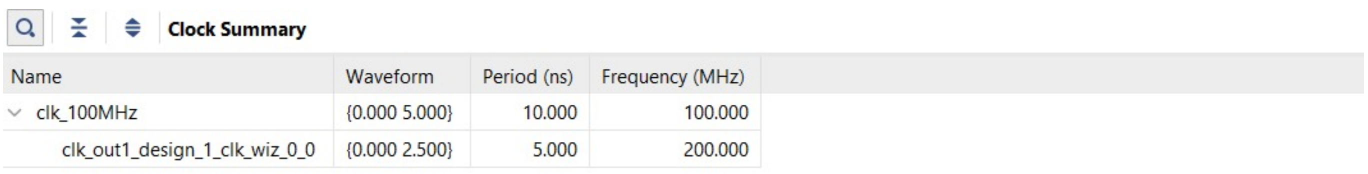
Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

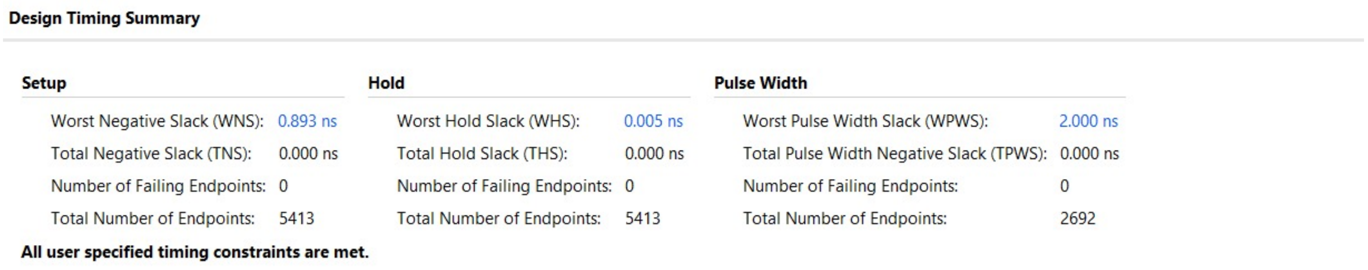
Dynamic:		0.153 W	(21%)
	Clocks:	0.007 W	(5%)
	Signals:	0.014 W	(9%)
	Logic:	0.015 W	(10%)
	DSP:	0.010 W	(7%)
	MMCM:	0.098 W	(64%)
	I/O:	0.009 W	(5%)
Device Static:		0.593 W	(79%)



Freqancy 200M


Clock Summary

Name	Waveform	Period (ns)	Frequency (MHz)
clk_100MHz	{0.000 5.000}	10.000	100.000
clk_out1_design_1_clk_wiz_0_0	{0.000 2.500}	5.000	200.000


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.893 ns	Worst Hold Slack (WHS):	0.005 ns	Worst Pulse Width Slack (WPWS):	2.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5413	Total Number of Endpoints:	5413	Total Number of Endpoints:	2692

All user specified timing constraints are met.

Future Enhancements

Extended Functional Verification using DPI-C & UVM

- Integrate the MATLAB model into SystemVerilog using **DPI-C** (Direct Programming Interface for C).
- Utilize the MATLAB model for co-simulation to verify RTL design accuracy.
- Develop a UVM (Universal Verification Methodology) testbench to cover edge cases and stress the design under randomized inputs.
- **Verification goals:** Check for design correctness, corner cases, and performance under varying inputs.

Increasing Matrix Size in RTL

- Expand the RTL to support larger matrix sizes (e.g., 4x4, 5x5, etc.) using the same systolic array structure.
- Investigate the effect of increasing matrix sizes on latency, resource utilization, and power consumption.