VLSI – I Final Project Report

DESIGN OF A CORDIC SVD PROCESSOR IP

Submitted to

Prof. Adnan Aziz
Department of Electrical and Computer Engineering,
University of Texas at Austin
December 2006

Project Members

Kapil Gulati (KG4434) Jasveen Kaur (JK5349

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Specification

1.1 Overview

CORDIC (COordinate Rotation DIgital Computer) is a simple and efficient algorithm to calculate hyperbolic and trigonometric functions. The modern CORDIC algorithm was first introduced by Volder[1] for the computation of trigonometric functions, multiplications, division and datatype conversions. Over the last decade, the CORDIC algorithm has been extended to varied mathematical functions like square roots [2], SVD [3] etc.

The CORDIC method provides an iterative scheme, consisting of simple addition and binary shift operations, to compute trigonometric values to any desired precision. The iterations in the CORDIC method can be pipelined to devise an efficient hardware unit that is capable of computing one sine and cosine every clock cycle.

Singular Value Decomposition (SVD) is an important algorithm in varied domains of signal processing. It is a generalized extension to the eigen-decomposition for non-square matrices and is hence of great importance, particularly for subspace based algorithms in signal processing. SVD is a complex algorithm with computational complexity ~$O(N^3)$ (for a NxN square matrix) and an efficient hardware implementation is desired for a real-time application. CORDIC arithmetic for SVD was developed by Cavallaro et.al. [3] that reduces the SVD to an iterative scheme that is twice as fast as the one based on conventional radix-2 addition oriented arithmetic.

1.2 Feature List

The features are listed below:

Interface

- AMBA APB bus [4] compliant

Mathematical Functions

- SVD (Singular Value Decomposition) of a 2x2 square matrix (with computation of all eigen-values, the left and the right eigenvectors)
- Sine / Cosine of an angle
- Tan^{-1} computation
- Support of the Generic Rotation mode
- Support of the Generic Vectoring mode

Number Format

- 32-bit fixed point 2's complement hardware implementation
- Programmable bit allocation for integer and fractional part, i.e. a generic 1.I.Q format number representation, where I, Q represent the number of bits for the integer part and the fractional part respectively
- Separate programmable number format for vector coordinates and vector phase

Input Range

- Angles in the range [$-\pi$, π] through coarse rotation
- Vector coordinates in the range [2^{-31}, 2^{31}] by appropriate programming of the format registers

The CORDIC SVD IP supports the computation of many mathematical functions apart for SVD (like sine, cosine and inverse tangent). It also supports generic rotation and vectoring mode operation of a CORDIC and hence can be programmed to compute any mathematic function that requires a single set of CORDIC-iteration cycles [2] (e.g. arcsine, arccosine etc). This gives a high flexibility to the programmer to configure the IP for a wide-range of mathematical functions possible to be computed by CORDIC arithmetic.

The SVD of higher dimension matrices (more than 2x2) can be implanted by CORDIC arithmetic as a mesh of 2x2 CORDIC SVD processors. Hence, this IP forms the basic building block for CORDIC SVD processors of higher dimension matrices.

1.3 System Block Diagram

The top-level system block diagram of the CORDIC SVD is as follows:

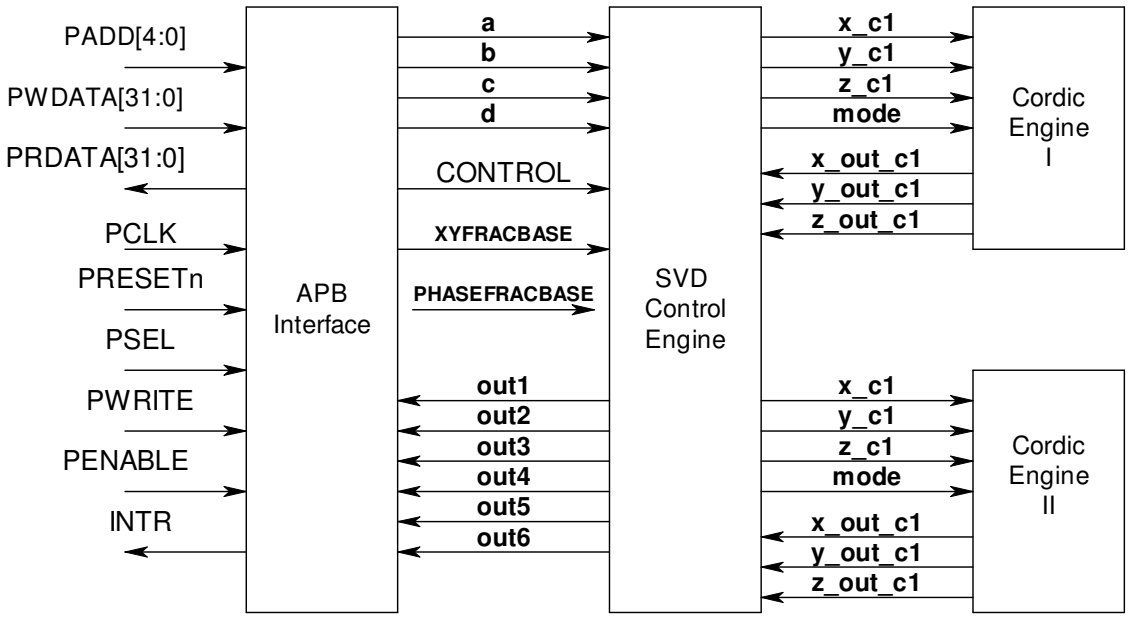

Figure 1: Block Diagram of a CORDIC SVD IP

1.4 Performance Targets

The performance goals for the CORDIC SVD IP are as mentioned in Table 1.

Target	**Performance**
Process	TSMC 0.18 um CMOS
Frequency	100 MHz
Area	3500000 square microns
Interface	AMBA APB compatible

Table 1: Performance Goals

To increase the performance of the IP, we have implemented Kogge-Stone structural adder. However the current performance is limited by time taken for the multiplication of two 32-bit 2's complement numbers. Hence, the performance can be considerably increased by choosing an efficient structural multiplier from the technology library.

1.5 System Interface

The CORDIC SVD IP is compatible with the AMBA APB interface [4]. The Advanced Peripheral Bus (APB) is a part of the Advanced Microcontroller Bus

Architecture (AMBA) hierarchy of busses and it is optimized for minimal power consumption and reduced interface complexity.

The IO ports are listed in Table [2],

Port Names	Input / Output	Description	Width
PRESETn	Input	Active-low Synchronous reset signal	1
PCLK	Input	System clock	1
PSEL	Input	IP Select Singal 0: IP not selected 1: IP selected	1
PADDR[5:0]	Input	Address	6
PENABLE	Input	Strobe signal for APB access 0: No transfer 1: Transfer ongoing	1
PWRITE	Input	Write / nRead signal 0: Read 1: Write	1
PWDATA[31:0]	Input	Write Data:	32
PRDATA[310]	Output	Read Data	32
INT	Output	Interrupt signal indicating that cordic operation is complete	1

Table 2: System Interface (I/O Port List)

2 Design Document

2.1 Tools

- Technology
  TSMC 0.18 um CMOS process library (HT018)
- Tools
  Compiler: VCS 7.0.1
  Synthesis: Synopsys Design Vision 2004.06-SP2

2.2 CORDIC SVD IP Block Diagram

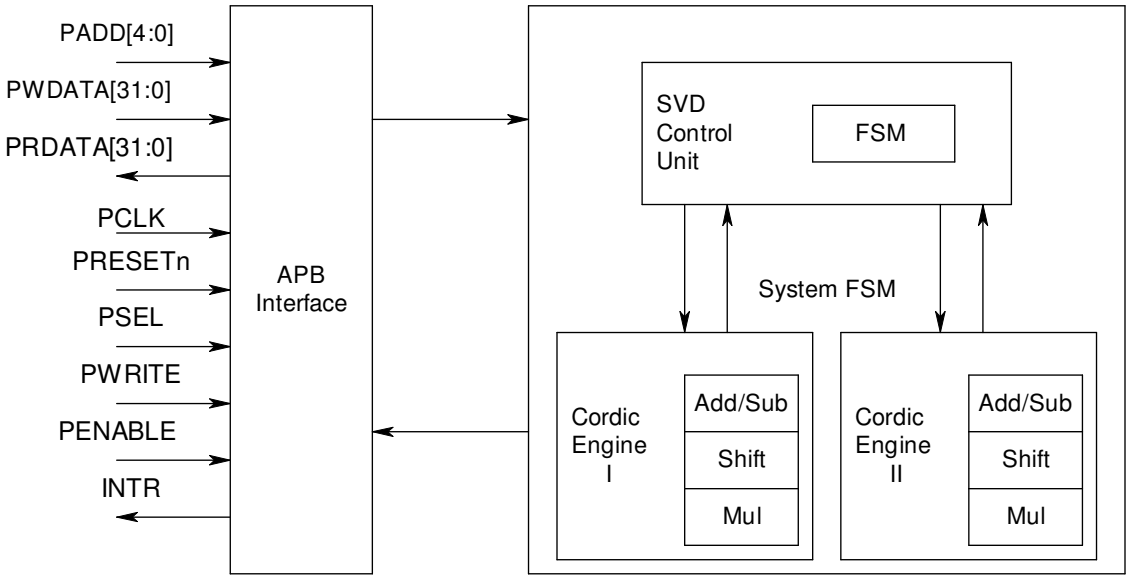

Figure 2: System Block Diagram

2.3 CORDIC Arithmetic

2.3.1 Concept of CORDIC

Andraka [2] presents an excellent summary of the CORDIC arithmetic and most of the discussion in this section is related to his work. All trigonometric functions can be computed or derived from functions using vector rotations. The CORDIC

Algorithms can provide the calculation of vector rotation and inverse tangent. The algorithm, credited to Volder[1], is derived from the general rotation transform:

$$x' = x\cos\phi - y\sin\phi$$
$$y' = y\cos\phi + x\sin\phi$$

which rotates a vector in a Cartesian plane by the angle ϕ. These can be arranged so that:

$$x' = \cos\phi.[x - y\tan\phi]$$
$$y' = \cos\phi.[y + x\tan\phi]$$

If the rotation angles are restricted such that $\tan(\phi) = \pm 2^{-i}$, the multiplication by the tangent term is reduced to simple shift operation. Arbitrary angles of rotation are obtainable by performing a series of successively smaller elementary rotations. If the decision at each iteration, I, is which direction to rotate rather than whether or not to rotate, then the $\cos(\delta_i)$ term becomes a constant. The iterative equation can now be expressed as,

$$x_{i+1} = K_i[x_i - y_i.d_i.2^{-i}]$$
$$y_{i+1} = K_i[y_i - x_i.d_i.2^{-i}]$$

where:

$$K_i = \cos(\tan^{-1} 2^{-i}) = 1/\sqrt{1 + 2^{-2i}}$$
$$d_i = \pm 1$$

The 2^{-i} multiplication can be implemented by shifter operation. The product of $K_i's$ approaches 0.6073 as the number of iterations goes to infinity. Therefore, the rotation algorithm has a gain, A_n of approximately 1.647.
The CORDIC Iteration equations implemented in hardware are:

$$X(i+1) = X(i) - \sigma(2^{-i}Y(i))$$
$$Y(i+1) = Y(i) + \sigma(2^{-i}X(i))$$
$$Z(i+1) = Z(i) - \sigma_i e(i)$$
$$\sigma_i = -1 \; or \; 1$$
$$R = \Pi(1 + \tan^2(e(i))^{1/2}$$

The 2^{-i} multiplication can be implemented by shifter operation. The compensation of R is addressed by putting 1/R (1/1.6467...) as the starting magnitude for sin()/cos() calculation. The R will converge into 1.6467 as the iteration number increases.

2.3.2 Sine and Cosine using CORDIC

The rotational mode CORDIC operation can simultaneously compute the sine and cosine of the input angle. Setting the y component of the input vector to zero reduces the rotation mode result to,

$$x_n = A_n \cdot x_0 \cos z_0$$
$$y_n = A_n \cdot x_0 \sin z_0$$

By setting x_0 equal to $1/A_n$, the rotation produces the un-scaled sine and cosine of the angle argument z_0. The CORDIC technique performs the multiply as part of the rotation operation, and therefore eliminates the need for a pair of explicit multipliers. The output of the CORDIC rotator is scaled by the rotator gain.

2.3.3 General Vector Rotation

The rotation mode CORDIC rotator is also useful for performing general vector rotations. For general rotation, a 2 dimensional input vector is presented to the rotator inputs. The rotator rotates the vector through vector through the desired angle. The output is scaled by the CORDIC rotator gain, which must be accounted for elsewhere in the system.

The equations used for this are:

$$x_{i+1} = K_i[x_i - y_i.d_i.2^{-i}]$$
$$y_{i+1} = K_i[y_i - x_i.d_i.2^{-i}]$$

where,

$$K_i = \cos(\tan^{-1} 2^{-i}) = 1/\sqrt{1+2^{-2i}} \qquad d_i = \pm 1$$

2.3.4 Arctangent

The arctangent, $\theta = \tan^{-1}(y/x)$, is directly computed using the vectoring mode CORDIC rotator if the angle accumulator is initialized with zero. The argument must be provided as a ratio expressed as a vector (x, y). Presenting the argument as a ratio has the advantage of being able to represent infinity. Since

the arctangent result is taken from the angle accumulator, the CORDIC rotator growth does not affect the result.

$$z_n = z_0 + \tan^{-1}(y_0 / x_0)$$

2.3.5 Coarse Rotation

The CORDIC rotation and vectoring algorithms are limited to rotation angles between $-\pi/2$ and $\pi/2$. For composite rotation angles larger than $\pi/2$, an additional rotation is required. Volder[1] describes an initial rotation of $\pm\pi/2$. This gives the correction iteration for inverse tan mode as,

$$x' = -d \cdot y$$
$$y' = d \cdot x$$
$$z' = z + d \cdot \pi/2$$
$$d = +1 \; if \;\; y < 0, -1 \;\; otherwise$$

The correction equation in rotation mode is:

$$x' = d \cdot y$$
$$y' = -d \cdot x$$
$$z' = z + d \cdot \pi/2$$
$$d = +1 \; if \;\; z < 0, -1 \;\; otherwise$$

2.3.6 Basic CORDIC Structure

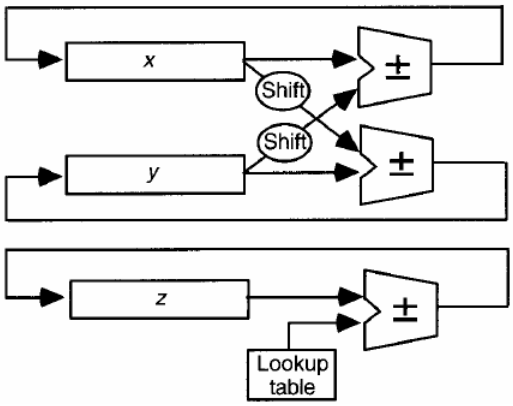

Figure 3: Basic CORDIC hardware

As shown above, CORDIC iteration requires 2 shifters, 1 table lookup and 3 adders. For n bits of precision, n iterations are needed. Therefore it results in the delay of O(n).

2.4 Singular Value Decomposition (SVD)

2.4.1 Concept of SVD

A singular value decomposition of a $P \times P$ matrix M is

$$M = U\Sigma V^T,$$

where U and V are orthogonal matrices and Σ is a diagonal matrix of the singular values. An efficient implementation of the SVD computation was developed by Brent, R. et.al. [5], in which matrix M is divided into 2×2 sub-matrices. This yields a scalable architecture with an array mesh of 2×2 sub-matrix SVD processors. There are two types of data flowing in this array. The rotation angles generated by the diagonal processors flow in a systolic manner along the rows and columns of the array. Matrix data elements are exchanged diagonally, after the diagonal neighbor has received and applied the necessary rotation angles. This leads to “waves” of activity diagonally away from the main array diagonal.

2.4.2 CORDIC SVD

A 2×2 SVD can be described as

$$R(\theta_l)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix},$$

where θ_l and θ_r are the left and right rotation angles respectively. The rotation matrix is,

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

and the input matrix is

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The angles θ_l and θ_r are found by the direct two-angle method by computing the inverse tangents of the data elements of M. Given M as defined above, θ_{SUM} and θ_{DIFF} are

$$\theta_{SUM} = (\theta_l + \theta_r) = \tan^{-1}(\frac{c+b}{d-a}),$$

$$\theta_{DIFF} = (\theta_r - \theta_l) = \tan^{-1}(\frac{c-b}{d+a})$$

The sum and difference angles yield the two angles, θ_l and θ_r, which can be applied to the two-sided rotation module to diagonalize M.

Thus, the general algorithm for a 2×2 CORDIC SVD processor would be:

CORDIC SVD ():
begin
    Use CORDIC angle-solver module to
        find rotation angles;
    Use CORDIC rotation module to
        transform the 2×2 matrix;
end

2.4.3 Parallel Diagonalization SVD

The Parallel Diagonalization Method, is based on determining θ_{SUM} and θ_{DIFF} directly. This results in reduction in time and area necessary for a 2×2 SVD processor as shown below

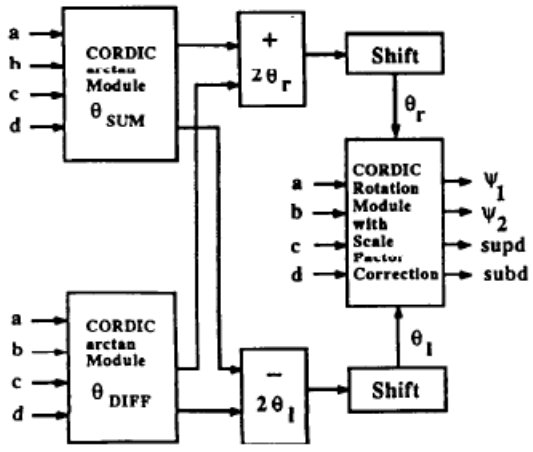

Figure 4: CORDIC SVD Parallel Diagonization Method [2]

The algorithm applying the parallel diagonalization method becomes

CORDIC SVD Parallel():
begin
parallel do $\{b+c, c-b, d-a, d+a\}$,
parallel do begin
Find $\theta_{SUM} = (\theta_r + \theta_l)$;
Find $\theta_{DIFF} = (\theta_r - \theta_l)$;
end;
parallel do Separate θ_r , θ_l ;
Apply θ_r , θ_l using CORDIC two-sided Rotation module;
parallel find sine/cosine of θ_r , θ_l using CORDIC processors
end

2.5 Sub Block Description

2.5.1 APB Interface

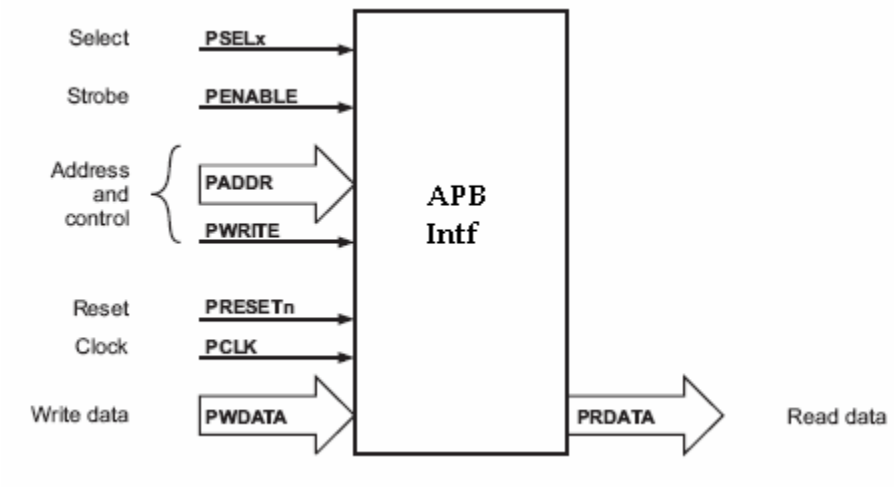

Figure 5: APB Interface Block Diagram[4]

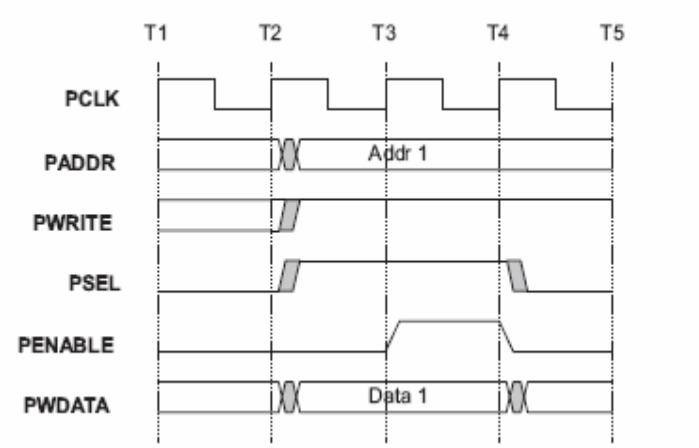

Figure 6: Write Transfer

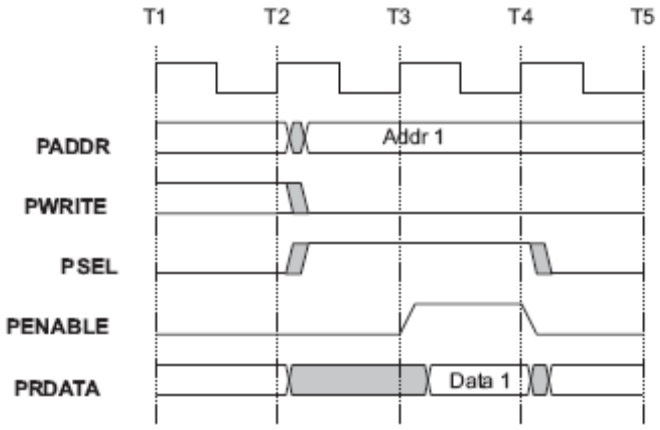

Figure 7: Read Transfer

Name	Description
PCLK	The main clock of CORDIC IP. All the transfers are done in the rising edge of the clock.
PRESETn	Bus reset signal, active low signal.
PADDR [5:0]	Address bus
PWDATA[31:0]	Write data
PRDATA[31:0]	Read data
PWRITE	WRITE/nREAD signal
PENABLE	Strobe signal for a APB access
PSEL	APB select signal
INT	CORDIC operation completion interrupt. active high signal

Table 3: CORDIC SVD Signal Description

2.5.2 CONTROL UNIT

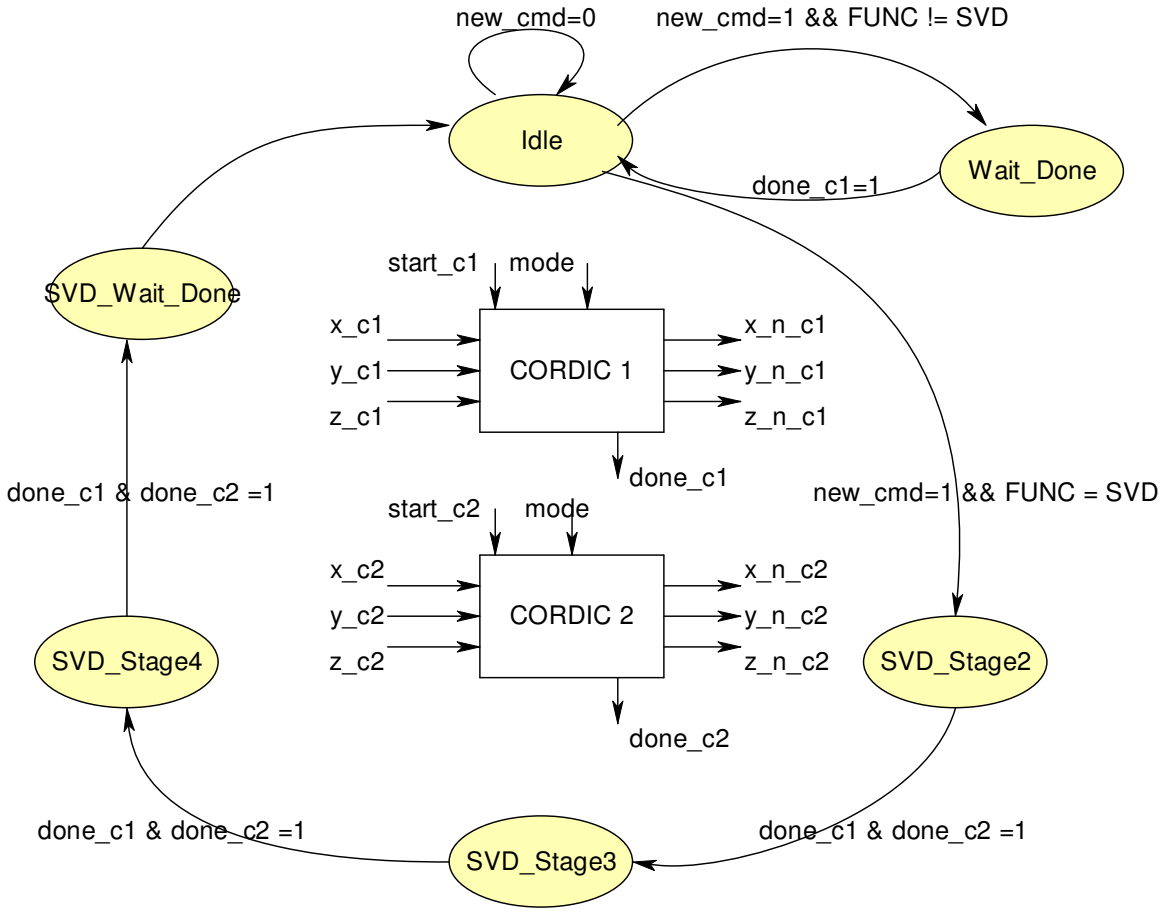

Figure 8: Control Unit FSM

FSM Description

FSM_Idle State

This is the default idle state of the FSM. It represents that the CORDIC processors are currently idle and ready to accept any new command.

FSM_Wait_Done State

This state is reached from the FSM_Idle state when a new command comes and the function is not SVD. It represents the execution of all single stage (32 iteration cycles) cordic operation for functions like sine/cosine, inverse tangent, generic vectoring/rotation mode.

The input to the CORDIC processor (CORDIC 1 only) is dependent on the function being executed. The following table lists the inputs stimulus to the CORDIC processor.

Note that only CORDIC 1 is used since these functions require a single stage CORDIC operation that comprises of 32 iteration cycles.

Function	Desired outputs	CORDIC processor inputs
Sin/Cos	$\cos(\theta), \sin(\theta)$	$x = 1$ $y = 0$ $z = \theta$ mode = rotation
Inverse Tangent	$\tan^{-1}(a)$	$x = 1$ $y = a$ $z = 0$ mode = vectoring
Generic Rotation	x_n, y_n, z_n	$x = x_0$ $y = y_0$ $z = z_0$ mode = rotation
Generic Vectoring	x_n, y_n, z_n	$x = x_0$ $y = y_0$ $z = z_0$ mode = vectoring

Table 4: CORDIC Processor inputs for different functions

FSM_SVD_Stage_2 State

This stage is reached if the CORDIC IP is programmed to compute the SVD of a matrix. The SVD of a matrix is computed in 4 stages as shown in the following figure,

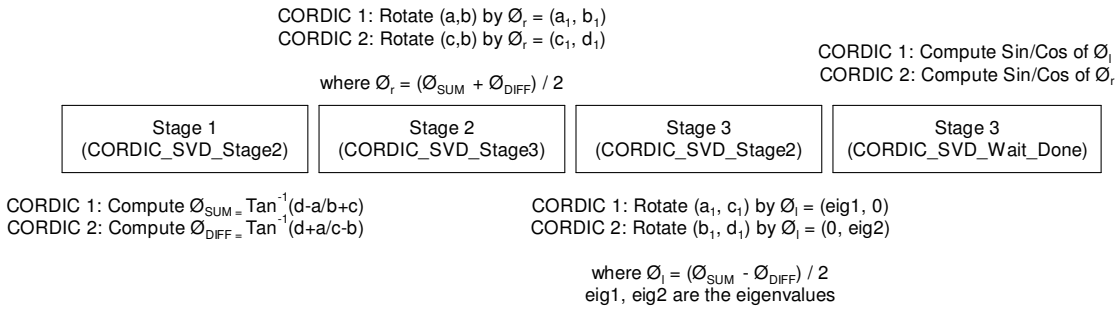

Figure 9: IP Stages to compute the SVD

In this stage the CORDIC processors are configured to compute θ_{SUM} and θ_{DIFF} respectively. The inputs to the CORDIC processors are as follows:

CORDIC 1	CORDIC 2
$x = d - a$	$x = d + a$
$y = b + c$	$y = c - b$
$z = 0$	$z = 0$
mode = vectoring	mode = vectoring
Output:	Output:
$z_n = \theta_{SUM} = \tan^{-1}(\frac{d-a}{b+c})$	$z_n = \theta_{DIFF} = \tan^{-1}(\frac{d+a}{c-b})$

Table 5: CORDIC Processor configuration in CORDIC_SVD_STAGE2

FSM_SVD_Stage_3 State

As shown in the figure above, Stage3 and Stage4 are used to diagonalize the input matrix. In this state, the matrix vectors are rotated by $\theta_r = (\theta_{SUM} + \theta_{DIFF})/2$. The following table lists the configuration of the 2 CORDIC processors

CORDIC 1	CORDIC 2
$x = a$	$x = c$
$y = b$	$y = d$
$z = \theta_r = (\theta_{SUM} + \theta_{DIFF})/2$	$z = \theta_r = (\theta_{SUM} + \theta_{DIFF})/2$
mode = rotation	mode = rotation
Output	
$M_1 = M \times R(\theta_r) = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix}$, *where* $R(\theta_r)$ *is the rotation matrix*	

Table 6: CORDIC Processor configuration in CORDIC_SVD_STAGE3

FSM_SVD_Stage_4 State

In this stage, the output of the previous stage is diagonalized by $\theta_l = (\theta_{SUM} - \theta_{DIFF})/2$. At the end of this stage, the eignevalues of the input matrix have been computed as shown the next table. The following table lists the configuration of the 2 CORDIC processors

CORDIC 1	CORDIC 2
$x = a_1$	$x = a_1$
$y = c_1$	$y = d_1$
$z = \theta_l = (\theta_{SUM} - \theta_{DIFF})/2$	$z = \theta_l = (\theta_{SUM} - \theta_{DIFF})/2$
mode = rotation	mode = rotation

Output
$R^T(\theta_l) \times M_1 = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix}$, *where* $R(\theta_l)$ *is the rotation matrix*

Table 7: CORDIC Processor configuration in CORDIC_SVD_STAGE4

FSM_SVD_Wait_Done State

This is the final stage for the SVD computation. In this stage the sine/cosine of θ_r and θ_l are found that completely represent the left and the right singular vectors. The output registers out1, out2, out3, out4, out5 and out6 are updated on the completion of this stage. The FSM returns to the IDLE state and is ready to accept new inputs.

CORDIC 1	CORDIC 2
$x = 1$	$x = 1$
$y = 0$	$y = 0$
$z = \theta_r$	$z = \theta_l$
mode = rotation	mode = rotation
Output	Output
$x_n = \cos(\theta_r)$	$x_n = \cos(\theta_l)$
$y_n = \sin(\theta_r)$	$y_n = \sin(\theta_l)$

Table 8: CORDIC Processor configuration in CORDIC_SVD_WAIT_DONE

2.5.3 CORDIC ENGINE

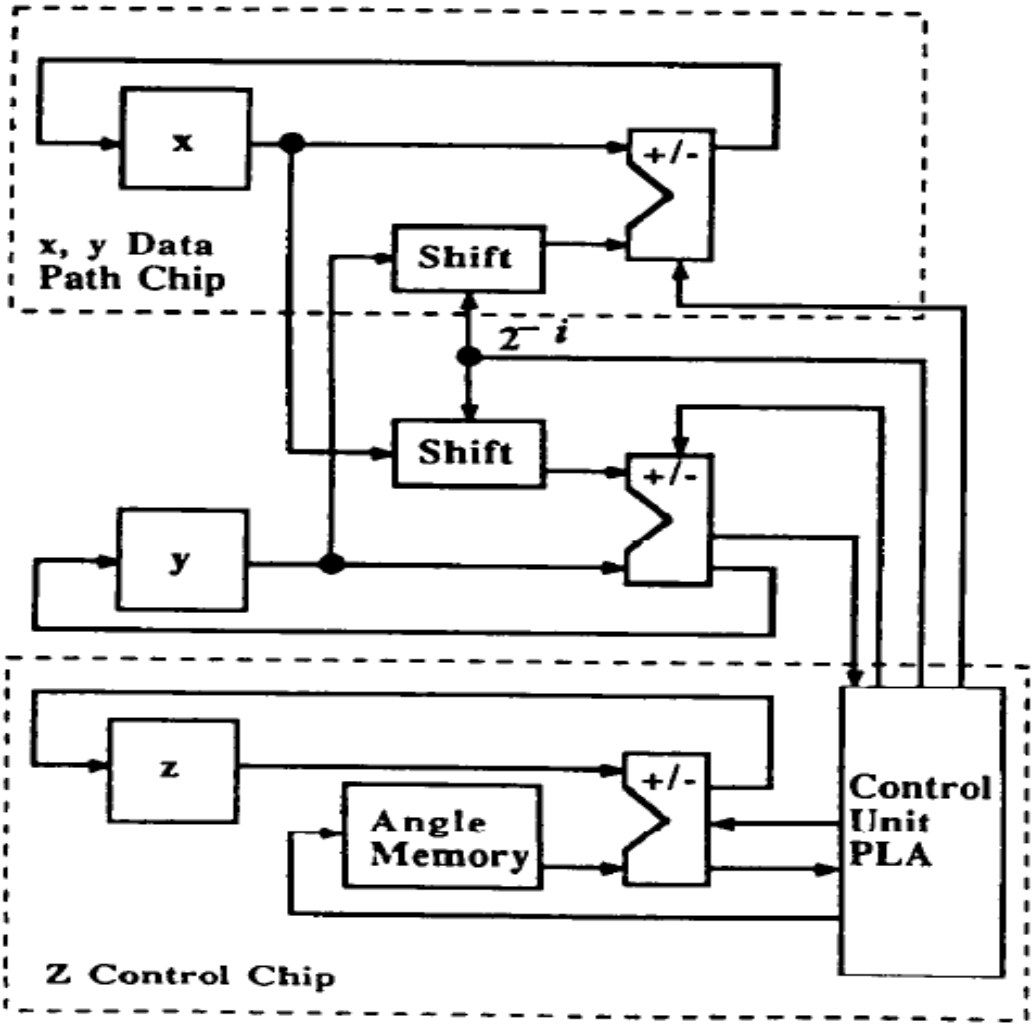

Figure 10: CORDIC Engine [3]

The CORDIC Engine takes three basic inputs x, y and z along with programmable values for data formats and a control signals indicating the start and completion of CORDIC processor. Internally adders, shifters and final stage multiplier are used to perform the computations.

2.5.4 Other Modules

In addition to the above three main modules we also have a system FSM and a top level.

System FSM

The top level FSM has two states SYSFSM_IDLE and SYSFSM_BUSY. It moves from one state to the other and passes the top level inputs in programmable registers a,b,c,d to the control logic and helps pre-compute the sift values based on the programmable data format by the user.

TOP Level Wrapper

A top level module is developed to help integrate the APB Interface with the system FSM.

3 User Document

3.1 Features

The CORDIC SVD Co-processor has the following features:

- AMBA APB bus compliant
- Support of the following mathematic computations
    - SVD (Singular Value Decomposition) of a 2x2 square matrix (with computation of all eigen-values, the left and the right eigenvectors)
    - Sine / Cosine of an angle
    - Tan^{-1} computation
- Support of the Generic Rotation mode
- Support of the Generic Vectoring mode
- 32-bit fixed point 2's complement hardware implementation
- Programmable bit allocation for integer and fractional part, i.e. a generic 1.I.Q format number representation, where I, Q represent the number of bits for the integer part and the fractional part respectively
- Separate programmable number format for vector coordinates and vector phase
- Angles in the range [$-\pi$, π] through coarse rotation
- Vector coordinates in the range [2^{-31}, 2^{31}] by appropriate programming of the format registers

3.2 Description

The CORDIC SVD IP is a programmable CORDIC processor that is capable of computation of many mathematical functional (SVD, Sin, Cos, Inverse Tangent) by CORDIC arithmetic. Beyond specific functions, it also supports the generic rotation and vectoring mode defined in context of CORDIC arithmetic [2] and hence can be programmed to compute of other mathematical functions (e.g. arcsine, arccosine, vector magnitude, some linear functions, hyperbolic functions). This can be done by programming the appropriate inputs to the CORDIC processor. Most of the input / output relationships to compute these functions has been discussed in [2].

The IP supports programmable input/output number format to provide flexibility in the required range on numbers and accuracy of the output. It also has separate programmable number formats for the vector co-ordinates (x, y coordinate input to the CORDIC co-processor) and the phase component (z input to the CORDIC processor). This enables varied and disjoint optimization in vector co-ordinates and phase components.

The CORDIC SVD IP integrates two-cordic co-processors which are used in parallel for the computation of the SVD. The IP is capable of computing the eigen-values, right and the left eigen-vectors. For other mathematical functions (sine, cosine, etc) only a single cordic co-processor is used.

3.3 Physical Specifications

Parameter	Details
Process	TSMC 0.18 um CMOS
Max Operating frequency	100 Mhz
Area	3297653.25 square microns
Power	1.2227 mW (as reported by Synopsys DC)
Vdd	Global operating voltage 2.35 V

Table 9: Physical Specifications

3.4 Block Diagram

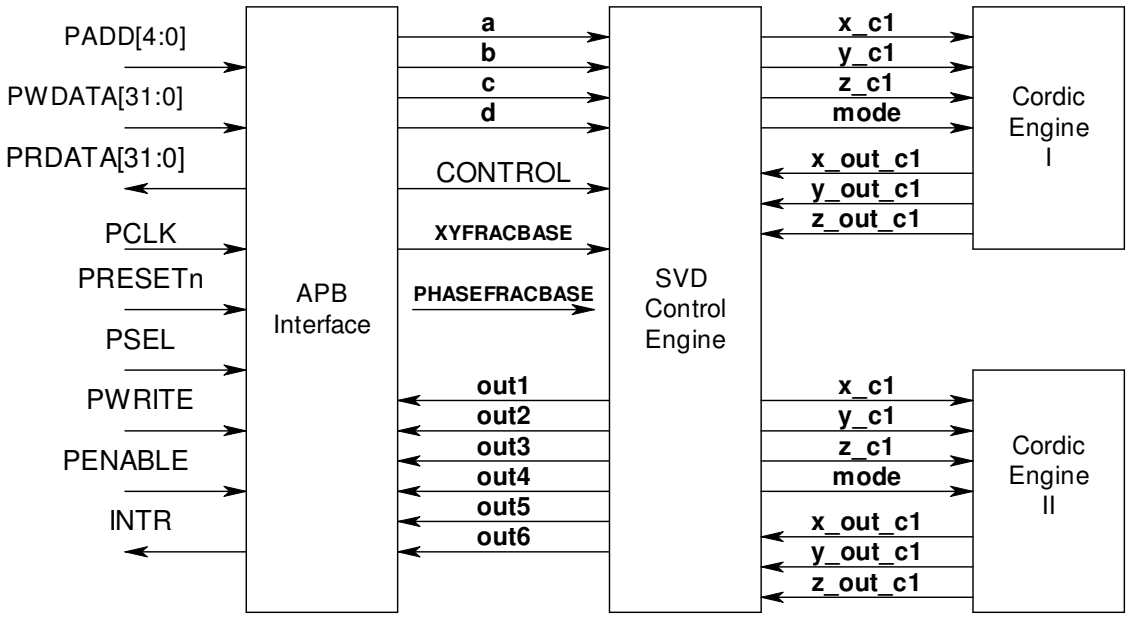

Figure 11: CORDIC IP Block Diagram

3.5 Signal Description

Port Names	Input / Output	Description	Width
PRESETn	Input	Active-low Synchronous reset signal	1
PCLK	Input	System clock	1
PSEL	Input	IP Select Singal 0: IP not selected 1: IP selected	1
PADDR[5:0]	Input	Address	6
PENABLE	Input	Strobe signal for APB access 0: No transfer 1: Transfer ongoing	1
PWRITE	Input	Write / nRead signal 0: Read 1: Write	1
PWDATA[31:0]	Input	Write Data:	32
PRDATA[310]	Output	Read Data	32
INT	Output	Interrupt signal indicating that cordic operation is complete	1

Table 10: Signal Description

3.6 Register Summary

The following table presents a summary of the registers present in the IP. It is followed by the full specification of all the registers.

Summary of Programmable Registers

Register Name	Address	General Description
CORDIC.CONTROL	0x00	Controls the operation of the CORDIC IP, like function selection, interrupt enable, trigger to start the cordic operation
CORDIC.PROG_A	0x04	Input 1 to the CORDIC co-processors
CODRIC.PROG_B	0x08	Input 2 to the CORDIC co-processors
CORDIC.PROG_C	0x0C	Input 3 to the CORDIC co-processors
CORDIC.PROG_D	0x10	Input 4 to the CORDIC co-processors
CORDIC.OUT1	0x14	Output 1 from the CORDIC IP
CORDIC.OUT2	0x18	Output 2 from the CORDIC IP
CORDIC.OUT3	0x1C	Output 3 from the CORDIC IP
CORDIC.OUT4	0x20	Output 4 from the CORDIC IP
CORDIC.OUT5	0x24	Output 5 from the CORDIC IP
CORDIC.OUT6	0x28	Output 6 from the CORDIC IP
CORDIC.XYFRACBASE	0x30	Number of Bits for fractional part of x,

		y coordinates
CORDIC.PHASEFRACBASE	0x34	Number of Bits for fractional part of the phase component

3.6.1 CORDIC.CONTROL Register

Address Offset 0x00
Description This register allows to control the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:6	Reserved	Write 0's for future compatibility. Reads return 0	R	0x000000
5	Interrupt Enable	0: Interrupt Disable 1: Interrupt Enable The CORIC IP generates an interrupt to indicate the end of CORIDC operation if this bit is set to 1	R/W	0x0
4	CORDIC Done	Indicates the end of CORDIC operation 0: CORDIC IP is Idle / Busy 1: CORDIC IP has completed the last operation	R	0x0
3:1	Function	CORDIC Funtion 000: SIN/COS 001: INVERSE TAN 010: GENERIC ROTATION MODE 011: SVD 100: GENRIC VECTORING MODE others: not supported	R/W	0x0
0	Start CORDIC	Write 1 to start the CORDIC operation. This bit is cleared once the CORDIC operation is started by the hardware.	R/W	0x0

3.6.2 CORDIC.PROG_A Register

Address Offset 0x04
Description This register contains the INPUT 1 to the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	PROG_A	INPUT 1 to the CORIC IP	R/W	0x00000000

3.6.3 CORDIC.PROG_B Register

Address Offset 0x08
Description This register contains the INPUT 2 to the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	PROG_B	INPUT 2 to the CORIC IP	R/W	0x00000000

3.6.4 CORDIC.PROG_C Register

Address Offset 0x0C
Description This register contains the INPUT 3 to the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	PROG_C	INPUT 3 to the CORIC IP	R/W	0x00000000

3.6.5 CORDIC.PROG_D Register

Address Offset 0x10
Description This register contains the INPUT 4 to the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	PROG_D	INPUT 4 to the CORIC IP	R/W	0x00000000

3.6.6 CORDIC.OUT1 Register

Address Offset 0x14
Description This register contains the OUTPUT 1 of the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	OUT1	OUTPUT 1 of the CORIC IP Valid after the CONTOL.DONE does high	R/W	0x00000000

3.6.7 CORDIC.OUT2 Register

Address Offset 0x18
Description This register contains the OUTPUT 2 of the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	OUT2	OUTPUT 2 of the CORIC IP Valid after the CONTOL.DONE does high	R/W	0x00000000

3.6.8 CORDIC.OUT3 Register

Address Offset 0x1C
Description This register contains the OUTPUT 3 the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	OUT3	OUTPUT3 of the CORIC IP Valid after the CONTOL.DONE does high	R/W	0x00000000

3.6.9 CORDIC.OUT4 Register

Address Offset 0x20
Description This register contains the OUTPUT 4 of the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	OUT4	OUTPUT 4 of the CORIC IP Valid after the CONTOL.DONE does high	R/W	0x00000000

3.6.10 CORDIC.OUT5 Register

Address Offset 0x24
Description This register contains the OUTPUT 5 of the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	OUT4	OUTPUT 5 of the CORIC IP Valid after the CONTOL.DONE does high	R/W	0x00000000

3.6.11 CORDIC.OUT6 Register

Address Offset 0x28
Description This register contains the OUTPUT 6 of the CORDIC IP
Type R/W

Bits	Field Name	Description	Type	Reset
31:0	OUT4	OUTPUT 6 of the CORIC IP Valid after the CONTOL.DONE does high	R/W	0x00000000

3.6.12 CORDIC.XYFRACBASE Register

Address Offset 0x2C
Description This register determined the number format for x,y coordinates
Type R/W

Bits	Field Name	Description	Type	Reset
31:5	Reserved	Write 0's for future compatibility. Reads return 0	R	0x0000000
4:0	XYFRACBASE	Number of bits used to represent the fractional part for the x, y vector co-ordinates Assuming that this field is programmed as N, then the number format would be 1.I.Q == 1.(31-N).N where 1 : Sign Bit (31-N) : Number of bits for integer part N : Number of bits for the fractional part Default Format: 1.15.16 (as per reset values)	R/W	0x10

3.6.13 CORDIC.PHASEFRACBASE Register

Address Offset 0x30
Description This register determined the number format for phase components
Type R/W

Bits	Field Name	Description	Type	Reset
31:5	Reserved	Write 0's for future compatibility. Reads return 0	R	0x0000000
4:0	PHASEFRACBASE	Number of bits used to represent the fractional part for the phase components Assuming that this field is programmed as N, then the number format would be 1.I.Q == 1.(31-N).N where 1 : Sign Bit (31-N) : Number of bits for integer part N : Number of bits for the fractional part Default Format: 1.3.28 (as per reset values)	R/W	0x1C

3.7 Data Format

The CORDIC IP supports programmable data format for both the x,y vector co-ordinates and the phase components. This is done by programming the following registers

Register	Address	To Program
CORDIC.XYFRACBASE	0x2C	x,y vector co-ordinates
CORDIC.PHASEFRACBASE	0x30	phase component

The hardware implementation represents the data as 32-bit fixed point 2's complement numbers. A configurable data format can hence be written as:
1.I.Q, where
- The most significant bit represents the sign of the number
- I is the number of bits used to represent the integer part
- Q is the number of bits used to represent the fractional part

Such that 1 + I + Q = 32. This is achieved by programming Q (= number of bits used to represent the fractional part) in the registers mentioned above. The reset values of the aforementioned registers enable the IP to be used in a default format as listed in the following table:

Component	**Data Format**	**Programming Register**	**Default Format (at reset)**
x,y co-ordinates	1.I.Q	CORDIC.XYFRACBASE	1.15.16
Phase	1.I.Q	CORDIC.PHASEFRACBASE	1.3.28

Table 11: Data Format

3.8 Programming Guide

A complete guide to programming the CORDIC-IP has been presented in this section. This enables a quick look-up for the user to program the CORDIC-IP for the desired functionality.

3.8.1 To Calculate SVD of a 2x2 Matrix

The Singular Value Decomposition (SVD) of a 2x2 square matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ can be written as follows:

$$R(\theta_l)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix},$$

where,

- ψ_1, ψ_2 are the eigen-values
- $R(\theta_l) = \begin{bmatrix} \cos(\theta_l) & \sin(\theta_l) \\ -\sin(\theta_l) & \cos(\theta_l) \end{bmatrix}$ are the left singular vectors
- $R(\theta_r) = \begin{bmatrix} \cos(\theta_r) & \sin(\theta_r) \\ -\sin(\theta_l) & \cos(\theta_r) \end{bmatrix}$ are the right singular vectors

Thus the Singular value decomposition is completely determined by the following parameters:
$\psi_1, \psi_2, \cos(\theta_l), \sin(\theta_l), \cos(\theta_r), \sin(\theta_r)$

The CORDIC SVD IP takes can be programmed to compute these 6 parameters for a 2x2 input matrix. The following table lists the steps needed to program to the CORDIC SVD IP to compute the SVD of an input matrix (using default data format).

SVD of a 2x2 input matrix using default data format

Step 1	Write $M(1,1) \times 2^{16} = a \times 2^{16}$ to CORDIC.PROG_A register Note that we have multiplied by 2^{16} to convert to the default data format of 1.15.16 for x, y co-ordinates
Step 2	Write $M(1,2) \times 2^{16} = b \times 2^{16}$ to CORDIC.PROG_B register
Step 3	Write $M(2,1) \times 2^{16} = c \times 2^{16}$ to CORDIC.PROG_C register
Step 4	Write $M(2,2) \times 2^{16} = d \times 2^{16}$ to CORDIC.PROG_D register
Step 5	Write 0x00000007 to CORDIC.CONTROL register Note: this step writes FUNCTION = SVD (bits 3:1) and also triggers the start of the CORDIC processing (bit 0)
Step 6	Wait for INT output to go high. This indicates the completion of the CORDIC operation. This can also be done by polling the Bit 4 of CORDIC.CONTOL to check for the 'CORDIC Done' field Total number of cycles taken to compute SVD = 32 x 4 = 128
Step 7	Read the outputs in any desired order. The outputs are available as follows: CORDIC.OUT1 == ψ_1 CORDIC.OUT2 == ψ_2 CORDIC.OUT3 == $\cos(\theta_r)$ CORDIC.OUT4 == $\sin(\theta_r)$ CORDIC.OUT5 == $\cos(\theta_l)$ CORDIC.OUT6 == $\sin(\theta_l)$ Output format: 1.15.16 (for all outputs)

Step 8	Write 0x00000010 to CORDIC.CONTROL register to clear the CORDIC Done bit field

Table 12: Programming steps to compute SVD

3.8.2 Sine/Cosine of an input angle

To compute the Sine/Cosine of an angle θ in the range $[-\pi, \pi]$, following steps need to be performed

Sine/Cosine of an angle using default data format	
Step 1	Write $\theta \times 2^{28}$ to CORDIC.PROG_A register Note that we have multiplied by 2^{28} to convert to the default data format of 1.3.28 for phase component
Step 2	Write 0x00000001 to CORDIC.CONTROL register Note: this step writes FUNCTION = SINCOS (bits 3:1) and also triggers the start of the CORDIC processing (bit 0)
Step 3	Wait for INT output to go high. This indicates the completion of the CORDIC operation. This can also be done by polling the Bit 4 of CORDIC.CONTOL to check for the 'CORDIC Done' field Total number of cycles to compute Sin/Cos = 32
Step 4	Read the outputs in any desired order. The outputs are available as follows: CORDIC.OUT1 == $\cos(\theta)$ CORDIC.OUT2 == $\sin(\theta)$ Output format: 1.15.16 for both outputs
Step 5	Write 0x00000010 to CORDIC.CONTROL register to clear the CORDIC Done bit field

Table 13: Programming steps to compute Sin/Cos

3.8.3 Inverse Tangent

To compute the inverse tangent of a number, following steps need to be performed

Inverse Tangent of a number [y = $\tan^{-1}$(x)]	
Step 1	Write $x \times 2^{16}$ to CORDIC.PROG_A register Note that we have multiplied by 2^{16} to convert to the default data format of 1.3.28 for phase component
Step 2	Write 0x00000003 to CORDIC.CONTROL register Note: this step writes FUNCTION = SINCOS (bits 3:1) and also triggers the start of the CORDIC processing (bit 0)
Step 3	Wait for INT output to go high. This indicates the completion of the CORDIC operation. This can also be done by polling the Bit 4 of CORDIC.CONTOL to check for the 'CORDIC Done' field Total number of cycles to compute $\tan^{-1}$(x) = 32
Step 4	Read the outputs in any desired order. The outputs are available as follows: CORDIC.OUT1 == $\tan^{-1}(x)$

	Output format: 1.3.28 (default phase data format)
Step 5	Write 0x00000010 to CORDIC.CONTROL register to clear the CORDIC Done bit field

Table 14: Programming steps to compute $\tan^{-1}(x)$

3.8.4 Generic CORDIC Rotation mode

The CORDIC iteration cycles can be represented as follows:

$X(i+1) = X(i) - \sigma(2^{-i} Y(i))$

$Y(i+1) = Y(i) + \sigma(2^{-i} X(i))$

$Z(i+1) = Z(i) - \sigma_i e(i)$

$\sigma_i = -1$ or 1 based on mode of operation

The selection of σ_i is based on the mode of operation of the CORDIC processors, i.e. rotation mode, vectoring mode. As shown in [2], based on the inputs provided to the CORDIC processors, numerous mathematical functions can be calculated using CORDIC arithmetic. The Generic Rotation mode can be used to compute many functions like sine, cosine, vector rotation, vector magnitude etc.

Generic Rotation mode (using default data format)

Step 1	Write $X_0 \times 2^{16}$ to CORDIC.PROG_A register Note that we have multiplied by 2^{16} to convert to the default data format of 1.15.16 for x, y co-ordinates
Step 2	Write $Y_0 \times 2^{16}$ to CORDIC.PROG_B register Note that we have multiplied by 2^{16} to convert to the default data format of 1.15.16 for x, y co-ordinates
Step 3	Write $Z_0 \times 2^{28}$ to CORDIC.PROG_C register Note that we have multiplied by 2^{28} to convert to the default data format of 1.3.28 for phase component
Step 4	Write 0x00000005 to CORDIC.CONTROL register Note: this step writes FUNCTION = SVD (bits 3:1) and also triggers the start of the CORDIC processing (bit 0)
Step 6	Wait for INT output to go high. This indicates the completion of the CORDIC operation. This can also be done by polling the Bit 4 of CORDIC.CONTOL to check for the 'CORDIC Done' field Total number of iterations = 32
Step 7	Read the outputs in any desired order. The outputs are available as follows: CORDIC.OUT1 == X_n CORDIC.OUT2 == Y_n CORDIC.OUT3 == Z_n Output format: 1.15.16 (for OUT1, OUT2) : 1.3.28 (for OUT3) Note that the suffix n represent the values of X, Y, Z after 32 cordic iteration cycles.

Step 8	Write 0x00000010 to CORDIC.CONTROL register to clear the CORDIC Done bit field

Table 15: Programming steps for Generic Rotation mode

3.8.5 Generic CORDIC Vectoring mode

This programming order is similar to the Generic CORDIC Vectoring mode. However, the correct function should be programmed in CORDIC.CONTROL register to enable this mode of operation. Hence, in Step 4, 0x00000011 should be written to the CORIDC.CONTROL register.

3.8.6 To Calculate SVD of a 2x2 Matrix with user defined data formats

The CORDIC SVD IP can be configured as per the desired data format for the x,y co-ordinates and the phase components. Programmable data format provides an increased range in the input data and can be effectively used as per the accuracy requirement of the output. For example, if the elements of the matrix (whose SVD has to be calculated) have high integral parts, then the data format can be programmed with higher bits to represent the integral part. Similarly, if the elements of the matrix are all fractional numbers, then more bits could be allocated for the fractional part.
Hence we increase the input range and the associated accuracy in the output.

Let us assume that the following configuration is required

Component	Format
x,y	1.(31-B_1).B_1
Phase z	1.(31-B_2).B_2

This can be achieved by the following programming steps:

SVD of a 2x2 input matrix with user defined data formats

Step 1	Write B_1 to CORDIC.XYFRACBASE register
Step 2	Write B_2 to CORDIC.PHASEFRACBASE register
Step 3	Write $M(1,1)\times 2^{B_1} = a\times 2^{B_1}$ to CORDIC.PROG_A register Note that we have multiplied by 2^{B_1} to convert to the default data format of 1.(31-B_1).B_1 for x, y co-ordinates
Step 4	Write $M(1,2)\times 2^{B_1} = b\times 2^{B_1}$ to CORDIC.PROG_B register
Step 5	Write $M(2,1)\times 2^{B_1} = c\times 2^{B_1}$ to CORDIC.PROG_C register
Step 6	Write $M(2,2)\times 2^{B_1} = d\times 2^{B_1}$ to CORDIC.PROG_D register
Step 7	Write 0x00000007 to CORDIC.CONTROL register Note: this step writes FUNCTION = SVD (bits 3:1) and also triggers the start of the CORDIC processing (bit 0)
Step 8	Wait for INT output to go high. This indicates the completion of the CORDIC

	operation. This can also be done by polling the Bit 4 of CORDIC.CONTOL to check for the 'CORDIC Done' field Total number of cycles taken to compute SVD = 32 x 4 = 128
Step 9	Read the outputs in any desired order. The outputs are available as follows: CORDIC.OUT1 == ψ_1 CORDIC.OUT2 == ψ_2 CORDIC.OUT3 == $\cos(\theta_r)$ CORDIC.OUT4 == $\sin(\theta_r)$ CORDIC.OUT5 == $\cos(\theta_l)$ CORDIC.OUT6 == $\sin(\theta_l)$ Output format: 1.(31-B_1).B_1 (for all outputs since all are x,y co-ordinate components)
Step 10	Write 0x00000010 to CORDIC.CONTROL register to clear the CORDIC Done bit field

Table 16: Programming steps to compute SVD with user defined data formats

Note that in this example (i.e. SVD function), all outputs belong to the x,y coordinate formatting. The effect of changing the phase format is internal to the processing of the SVD. For other functions, like inverse tangent, generic rotation/vectoring mode, effect of both the data format parameters can be seen in the output.

4 Testing

4.1 Test Inputs

4.1.1 MATLAB Modeling Strategy

We created a MATLAB model for SVD implementation using a CORDIC Engine. The code was written to emulate the exact function of the verilog code. According to the results a verilog testbench was generated to test the functionality of the actual verilog code for all the corner cases and to compare the results.

This approach has significant benefits. First benefit is time saving. Running time for MATLAB model is significantly shorter than Verilog, so we could save a lot of developing and debugging time. Second benefit is easy debugging. MATLAB model emulates Verilog code using logical operations and mathematical functions. Therefore, we can easily check intermediate values without looking at waveforms. Moreover, we can compare expected values for arbitrary input values using mathematical library; this helps reduce debugging time.

4.2 Test Coverage

4.2.1 General Coverage Strategy

It's impossible to test all possible combination inputs for this IP since each input register is 32-bit. Even though some of the inputs are not defined, testing for all valid inputs is also not feasible. Therefore, we chose representative inputs for each component.

The test inputs were used to test the following modes

- SVD
- Sin/Cos
- Tan-1
- Generic Rotation
- Generic Vectoring

Within these modes we tested for different combinations of the format for x,y and phase components. We also tested for angles in all the 4 quadrants and the SVD for both positive and negative valued matrices.

We chose inputs to cover representative angles and magnitude. We also considered small angles because their intermediate values change frequently;

they might accumulate large errors if we ignored or missed small fraction of errors. Specific testcases for generic vectoring and rotating mode were not developed they were tested by implication in the SVD Mode.

4.2.2 Representative Tests Description

Test 1: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase = 16

Test 2: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase = 20

Test 3: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase = 24

Test 4: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase = 12

Test 5: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase = 8

Test 6: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase = 4

Test 7: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase = 0

Test 8: SVD Mode Test with input matrix $M = \begin{bmatrix} 1 & 5 \\ 2 & 7 \end{bmatrix}$ XYFractionalBase = 16

Checks for the transpose of the initial sample matrix

Test 9: SVD Mode Test with input matrix $M = \begin{bmatrix} -1 & -2 \\ -5 & -7 \end{bmatrix}$ XYFractionalBase = 16

Checks for negative values of the initial sample matrix

Test 10: Sin/Cos Mode Test with input angle 60° (Quadrant 1)

Test 11: Sin/Cos Mode Test with input angle 120° (Quadrant 2)

Test 12: Sin/Cos Mode Test with input angle -60° (Quadrant 4)

Test 13: Sin/Cos Mode Test with input angle -120° (Quadrant 3)

Test 14: Sin/Cos Mode Test with input angle 0°

Test 15: Sin/Cos Mode Test with input angle 90°

Test 16: Sin/Cos Mode Test with input angle 45° PhaseBase = 20

Test 17: InvTan Mode Test with input value 1 (To yield 45°)

Test 18: InvTan Mode Test with input value 32767 (Large Value to yield 90°)

Test 19: InvTan Mode Test with input value -6550 (Large Negative Value to yield -90°)

Test 20: InvTan Mode Test with input value 0 (To yield 0°)

Test 21: Generic Rotation Mode Test with input vector (1,1) rotated by 45°

Test 22: Generic Rotation Mode Test with input vector (1,1) rotated by 120°

Test 23: Generic Vectoring Mode Test with input vector (1,1) , phase 45°

Test 24: Generic Vectoring Mode Test with input vector (1,1) , phase 120°

4.2.3 Representative Tests Inputs/Outputs

4.2.3.1 Mode - SVD

#	Inputs					
	a[31:0]	**b[31:0]**	**c[31:0]**	**d[31:0]**	**xyfrac**	**pfrac**
1.	32h'0001000	32h'0002000	32h'0005000	32h'0007000	5h'10	5'h1c
2.	32h'0010000	32h'0020000	32h'0050000	32h'0070000	5h'14	5'h1c
3.	32h'0100000	32h'0200000	32h'0500000	32h'0700000	5h'18	5'h1c
4.	32h'0000100	32h'0000200	32h'0000500	32h'0000700	5h'0c	5'h1c
5.	32h'0000010	32h'0000020	32h'0000050	32h'0000070	5h'08	5'h1c
6.	32h'0000000	32h'0000000	32h'0000000	32h'0000000	5h'04	5'h1c
7.	32h'0000001	32h'0000002	32h'0000005	32h'0000007	5h'00	5'h1c
8.	32h'0001000	32h'0005000	32h'0002000	32h'0007000	5h'10	5'h1c
9.	32h'ffff0000	32h'fffe0000	32h'fffb0000	32h'fff90000	5h'10	5'h1c

Table 17: SVD MODE Inputs (Test1 - Test9)

Outputs					
out1	**out2**	**out3**	**out4**	**out5**	**out6**
32h'ffffa98d	32h'0008e1a1	32h'0000d1c3	32'h000092be	32'h0000f7ee	32'h00003fc0
32h'fffa987f	32h'008e1ba6	32'h000d1c2d	32'h00092c07	32'h000f7eef	32'h0003fc1a
32h'ffa987df	32h'08e1bbe1	32'h00d1c2c9	32'h0092c0b2	32'h00f7eeff	32'h003fc1c4
32h'fffffa98	32h'00008e0f	32h'00000d1b	32'h0000092a	32'h00000f7f	32'h000003fb
32h'ffffffa6	32h'000008e2	32h'000000d1	32'h00000092	32'h000000f7	32'h0000003f
32h'ffffffa	32'h00000079	32'h0000000c	32'h00000008	32'h0000000e	32'h00000003
32h'0000000	32h'00000000	32h'00000000	32h'00000000	32h'00000000	32h'00000000
32h'ffffa987	32'0008e19e	32'h0000f7ee	32h'00003fc0	32'h0000d1c3	32h'000092be
32h'ffffa98d	32'0008e1a	32'hffff2e3c	32'hffff6d42	32'h0000f7ee	32'h00003fc0

Table 18: SVD MODE Outputs (Test 1 - Test 9)

4.2.3.2 Mode – SINE/COSINE

#	Inputs			Outputs	
	a[31:0]	**xyfrac**	**pfrac**	**Out1**	**Out2**
10.	32'h10c15238	5'h10	5'h1c	32'h0008000	32'h0000ddb1
11.	32'h2182a470	5'h10	5'h1c	32'hffff7fff	32'h0000ddb2
12.	32'hefbeadc8	5'h10	5'h1c	32'h000086dd	32'hffff2699
13.	32'hde7d5b90	5'h10	5'h1c	32'hffff8002	32'hffff224b
14.	32'h00000000	5'h10	5'h1c	32'h0000ffff	32'h00000000
15.	32'h1921FB54	5'h10	5'h1c	32'h00000000	32'h0000ffff
16.	32'h000C90FE	5'h10	5'h14	32'h0000b504	32'h0000b503

Table 19: SIN/COS Input-Ouput (Test 10 - Test 16)

4.2.3.3 Mode – INVTAN

#	Inputs			Outputs
	a	xyfrac	Pfrac	out3
17.	32'h00010000	5'h10	5'h1c	32'h19220dc8
18.	32'h7fff0000	5'h10	5'h1c	32'h0c13ae58
19.	32'he66a0000	5'h10	5'h1c	32'he6dea4c2
20.	32'h00000000	5'h10	5'h1c	32'hffffed8e

Table 20: INVTAN Mode - Inputs/Outputs (Test 17 - Test 20)

4.2.3.4 Mode – Generic Rotation

#	Inputs			Outputs	
	a[31:0]	b[31:0]	c[31:0]	out1	out2
21.	32'h00010000	32'h00010000	32'h19220dc8	32'hfffeffff	32'h0000fffa
22.	32'h00010000	32'h00010000	32'h2182a470	32'hfffea24d	32'h00005db2

Table 21: Generic Rotation Mode - Inputs/Outputs (Test 21 - Test 22)

4.2.3.5 Mode – Generic Vectoring

#	Inputs			Outputs		
	a[31:0]	b[31:0]	c[31:0]	out1	out2	out3
23.	32'h00010000	32'h00010000	32'h19220dc8	32'h00016a08	32'h00000000	32'h25b31bba
24.	32'h00010000	32'h00010000	32'h2182a470	32'h00016a08	32'h00000000	32'h2e13b262

Table 22: Generic Vectoring Mode - Inputs/Outputs (Test 23 - Test 24)

4.2.4 Representative Test Results from MATLAB

#	Outputs					
	eig1	eig2	sin_thr	cos_thr	sin_thl	cos_thl
1.	FFFFA984	0008E216	0000D1C7	000092C4	0000F7F4	00003FC3

2.	FFFA9847	008E2163	000D1C6F	00092C3A	000F7F3F	0003FC31
3.	FFA98473	08E2162E	00D1C6F3	0092C39E	00F7F3EA	003FC309
4.	FFFFFA98	FFFFFA98	00000D1C	0000092C	00000F7F	000003FC
5.	FFFFFFAA	000008E2	000000D2	00000093	000000F8	00000040
6.	FFFFFFFB	0000008E	0000000D	00000009	0000000F	00000004
7.	100000000	00000009	00000001	00000001	00000001	00000000
8.	FFFFA984	0008E216	0000F7F4	00003FC3	0000D1C7	000092C4
9.	FFFE3377	0008E216	0000F7F4	00003FC3	FFFF2E39	FFFF6D3C

Table 23: Output Data from MATLAB (Test 1 - Test 9)

4.2.5 Error Computation

The percentage error was calculated by comparing the verilog simulation results with the MATLAB Results. The calculation was performed using the first 9 sample sets of data as reference. Square Error Value for SVD Mode is approximately $\approx 10^{-8}$.

Since CORDIC arithmetic is itself an approximation, we performed an analysis of the error in computation of basic mathematical functions – sin, cos, inverse tangent. These results have been presented in the next section and have meaningful interpretation.

For detailed verilog testbench refer to Appendix A.
For the MATLAB Code refer to Appendix B.

4.3 Verilog Simulation Results using VCS

4.3.1 Single SVD

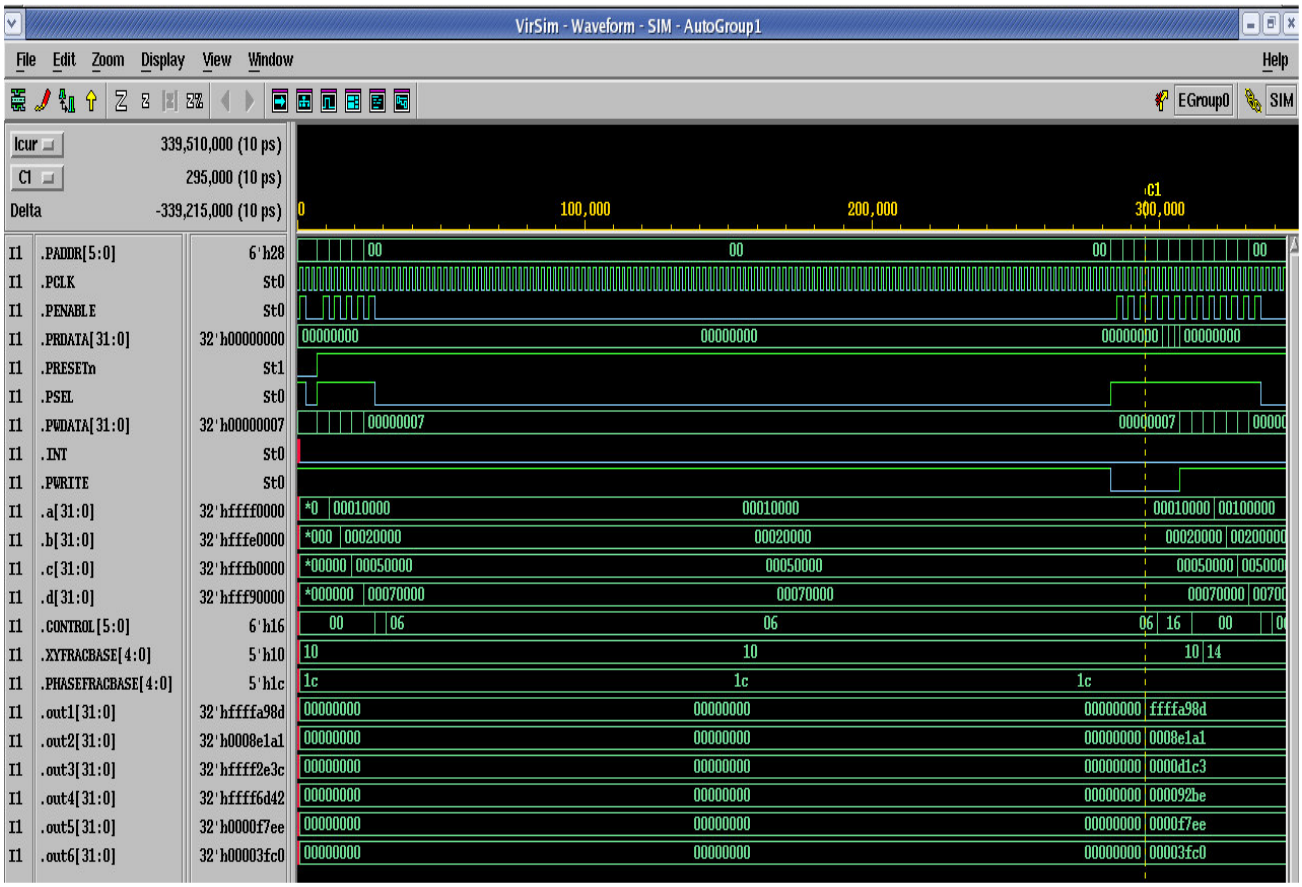

Figure 12: Waveform for SVD Mode Inputs a =1, b=2, c=5, d =7 (Test 1)

The above figure shows the waveform corresponding to Test 1

Test with input matrix $M = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$ XYFractionalBase =16

Outputs are the eigen-values as out1 (eig1) = 32h'ffffa98d, out2 (eig2) = 32h'0008e1a1. This corresponds to decimal values -0.3378 and 8.8832

The resultant orthogonal matrix obtained is

$$\begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix} = \begin{bmatrix} -0.3378 & 0 \\ 0 & 8.8832 \end{bmatrix}$$

4.3.2 SVD for multiple XY fractional base

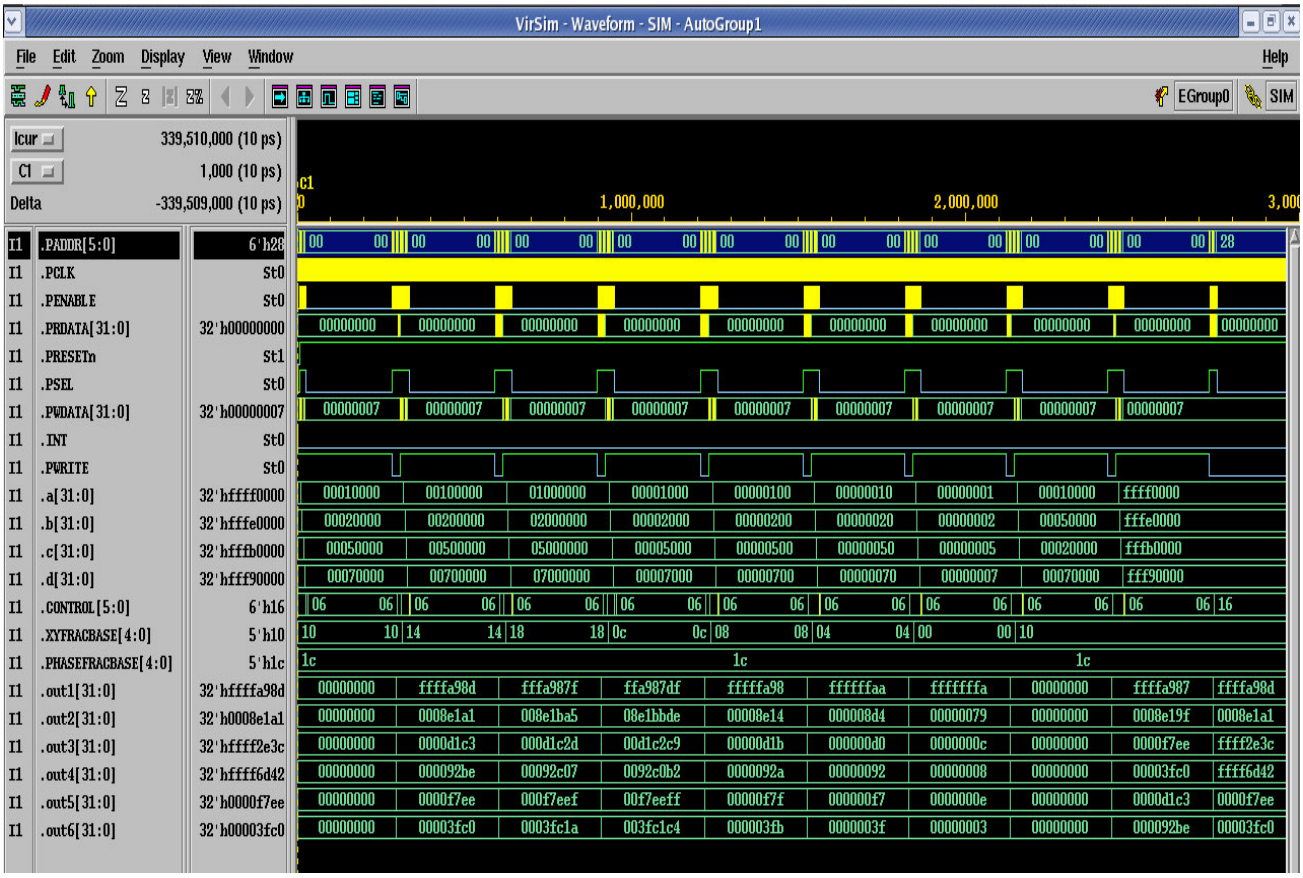

Figure 13: Waveform for SVD Mode (Test 1 - Test 9)

The above figure shows the waveform corresponding to the Test 2 – Test 9 which shows different input matrices to the SVD IP along with different programmable value for the XY fractional base.

4.3.3 SIN()/COS() Mode

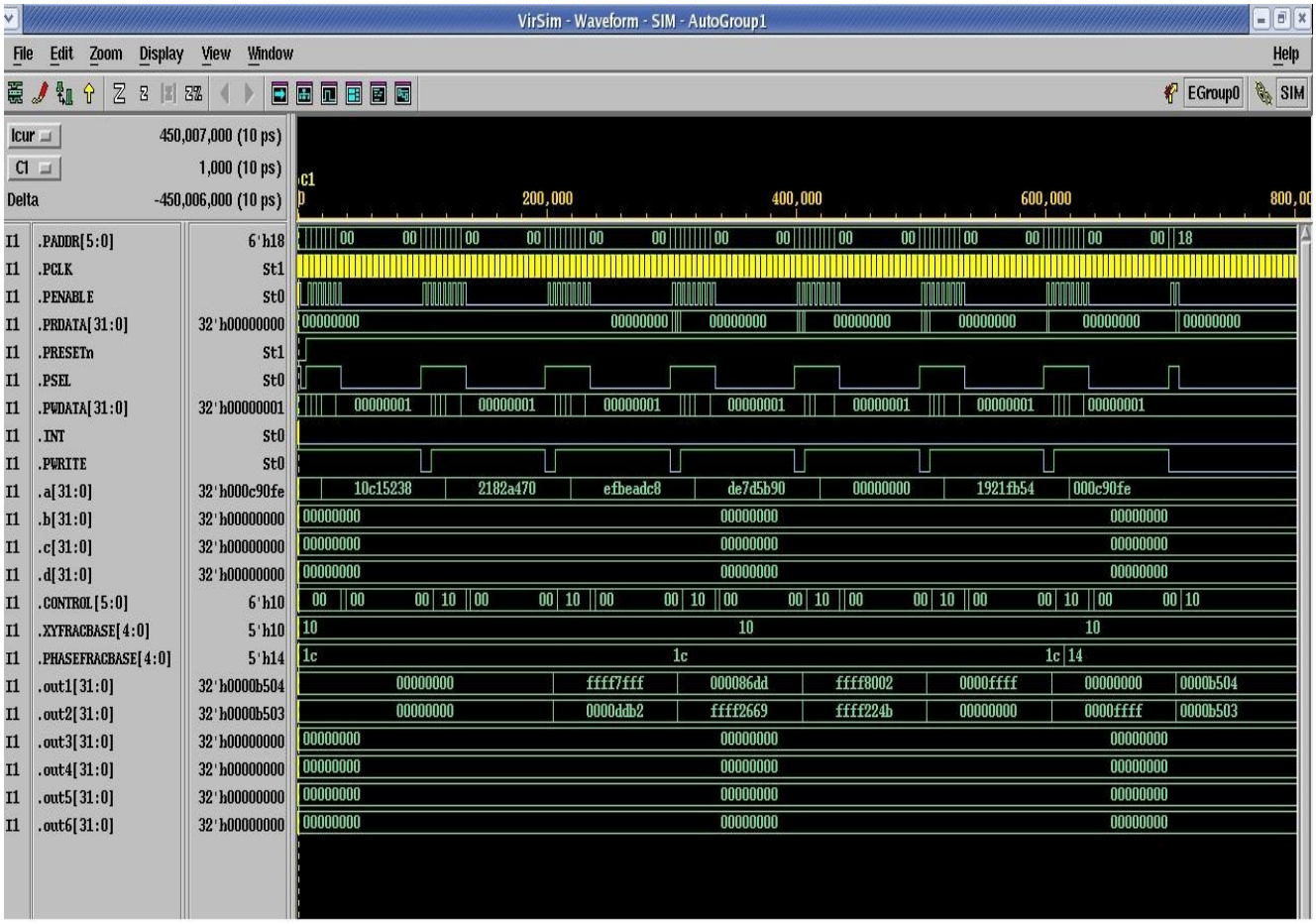

Figure 14: Waveform for SIN/COS Mode (Test 10 - Test 16)

The above figure shows the waveform corresponding to the Test 10 – Test 16 which shows different input values one in each quadrant along with programmable fractional Phase along with corresponding outputs and the signals of the top level APB Interface.

4.3.4 INVTAN Mode

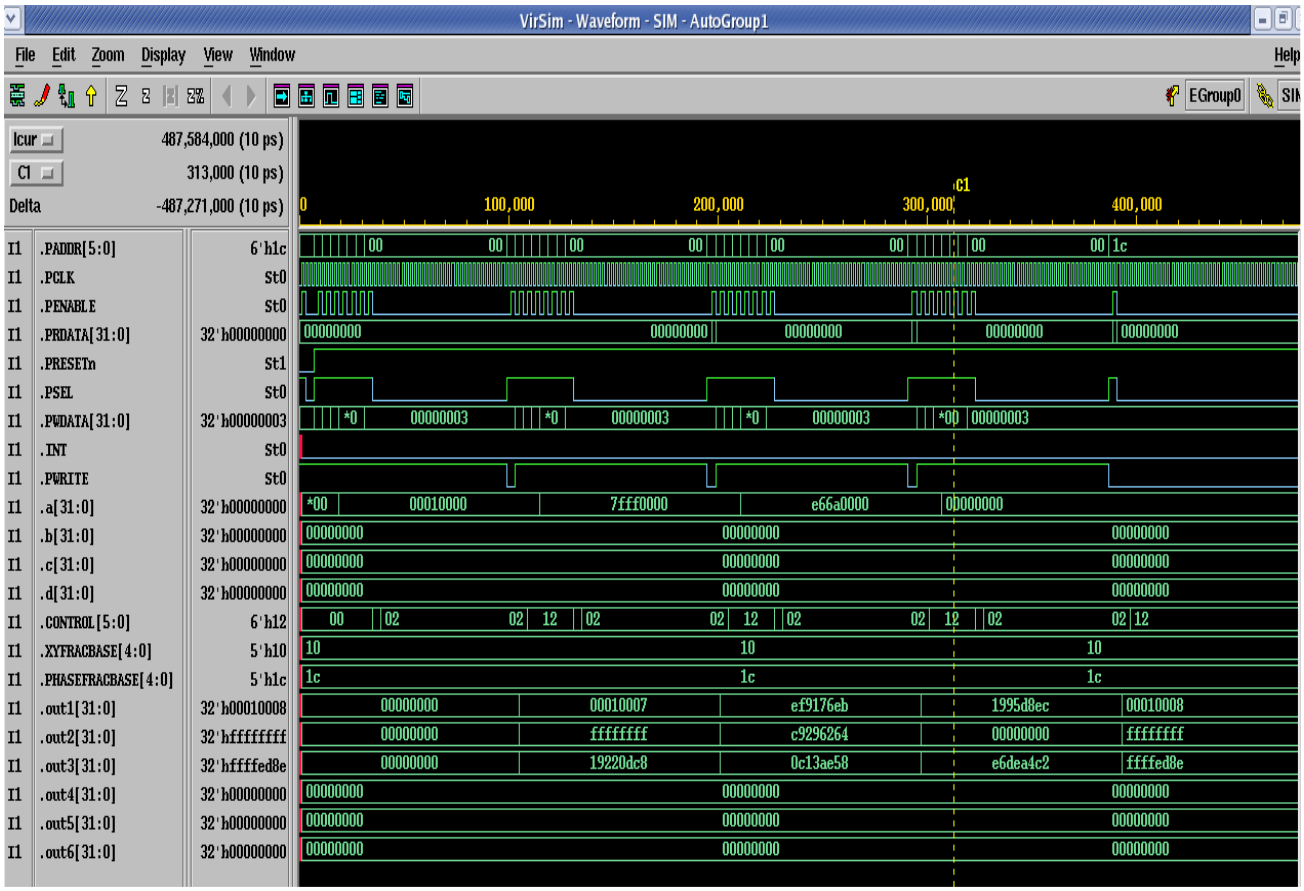

Figure 15: Waveform for Inverse Tan Mode (Test 17 - Test 20)

The above figure shows the waveform corresponding to the Test 17 – Test 20 which shows different input values for calculating inverse tan along with the corresponding outputs and the top level signals of the APB interface

4.3.5 Generic Rotating and Vectoring Mode

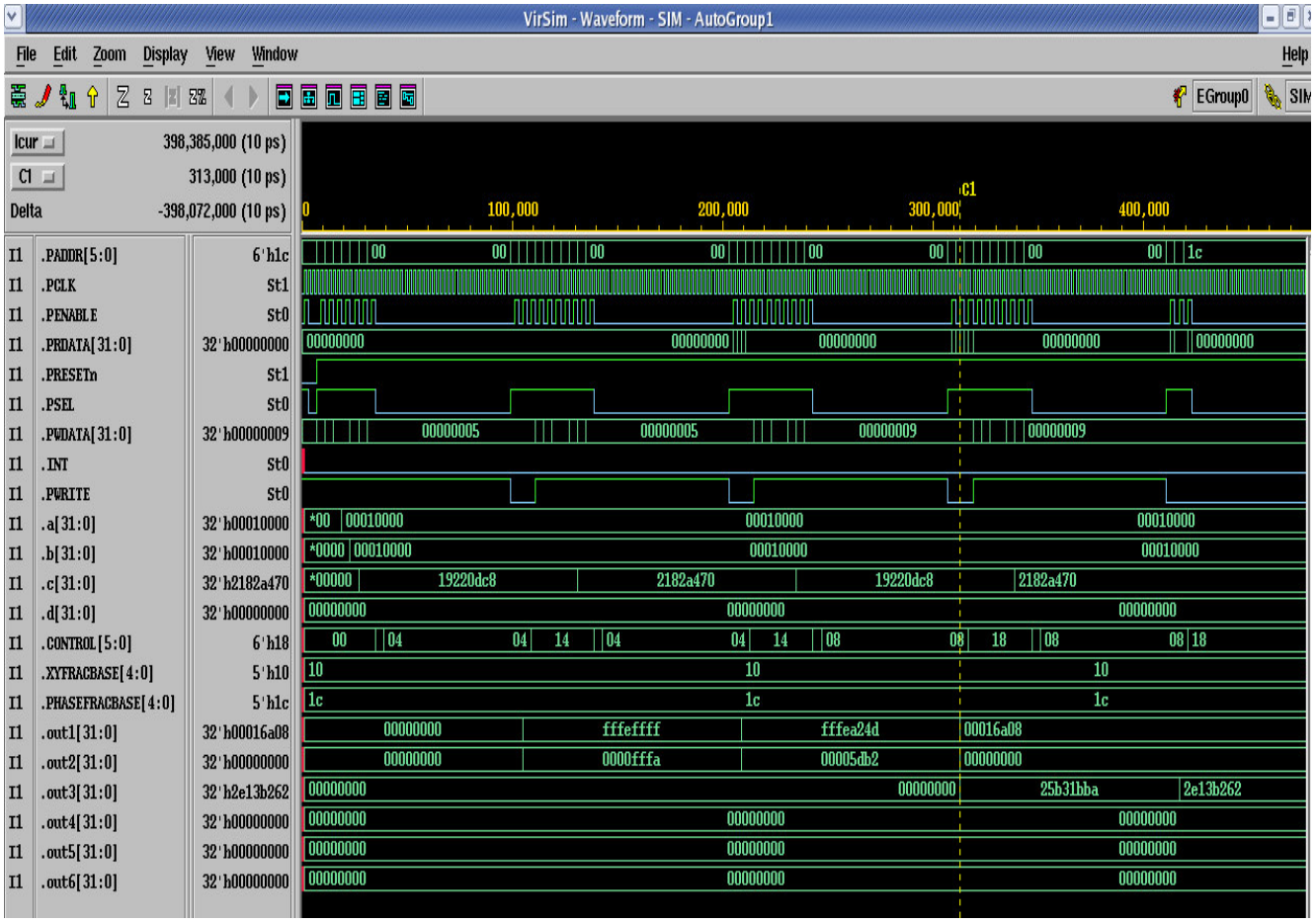

Figure 16: Waveform for Generic Vectoring / Rotation Mode (Test 21 - Test 24)

The above figure shows the waveform corresponding to the Test 21 – Test 24 which shows different input values for rotating and vectoring mode and the corresponding inputs along with the top level signals of the APB Interface.

4.4 MATLAB Results for Error Analysis

4.4.1 Calculation of $\sin(\theta)$

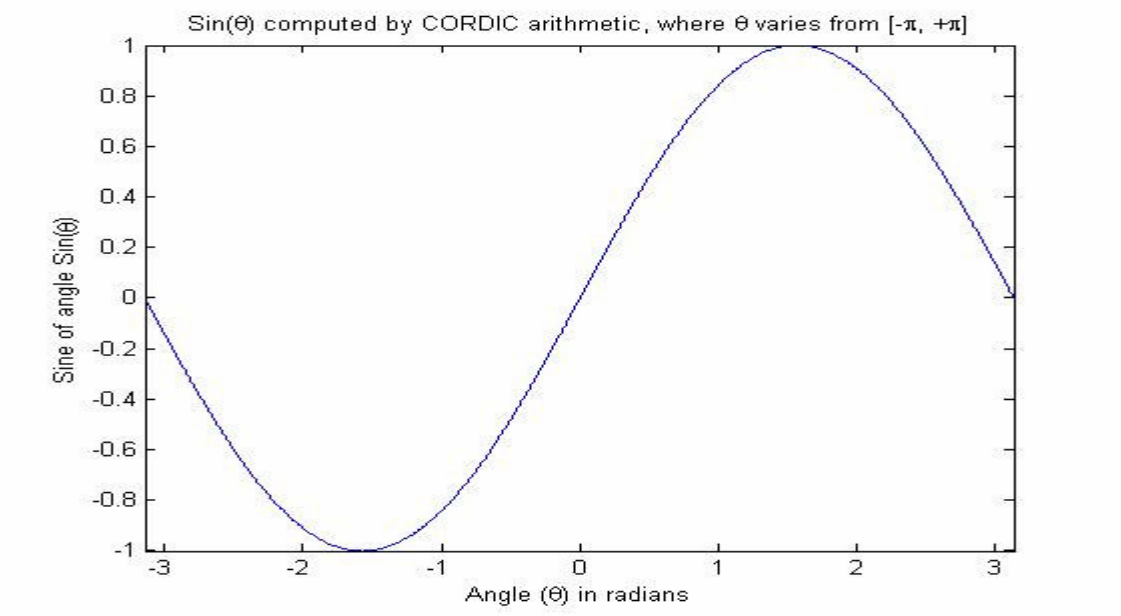

Figure 17: Sin calculation from -pi, pi by CORDIC arithmetic

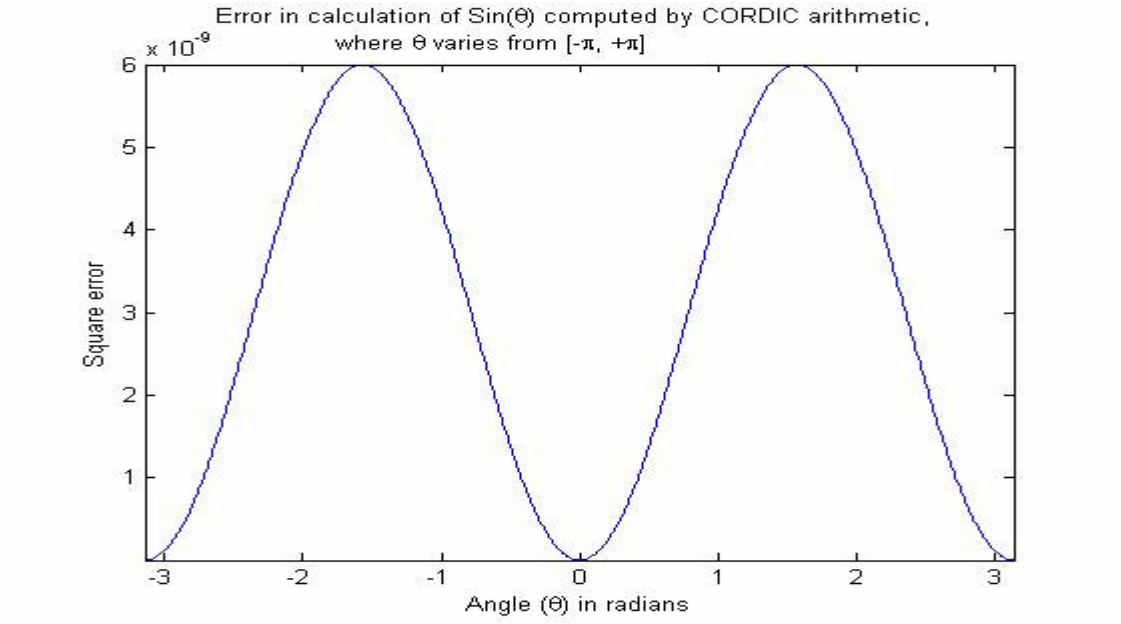

Figure 18: Error Calculation for Sin from -pi, pi in CORDIC arithmetic

The Error Calculation graph suggests that the maximum error of the order of 10^{-9} occurs at the values $\pm\pi$. The error peaks for rotation angles of $\pm\pi/2$ a full rotation is required as these times. Hence the CORDIC arithmetic will present the maximum error.

4.4.2 Calculation of $\cos(\theta)$

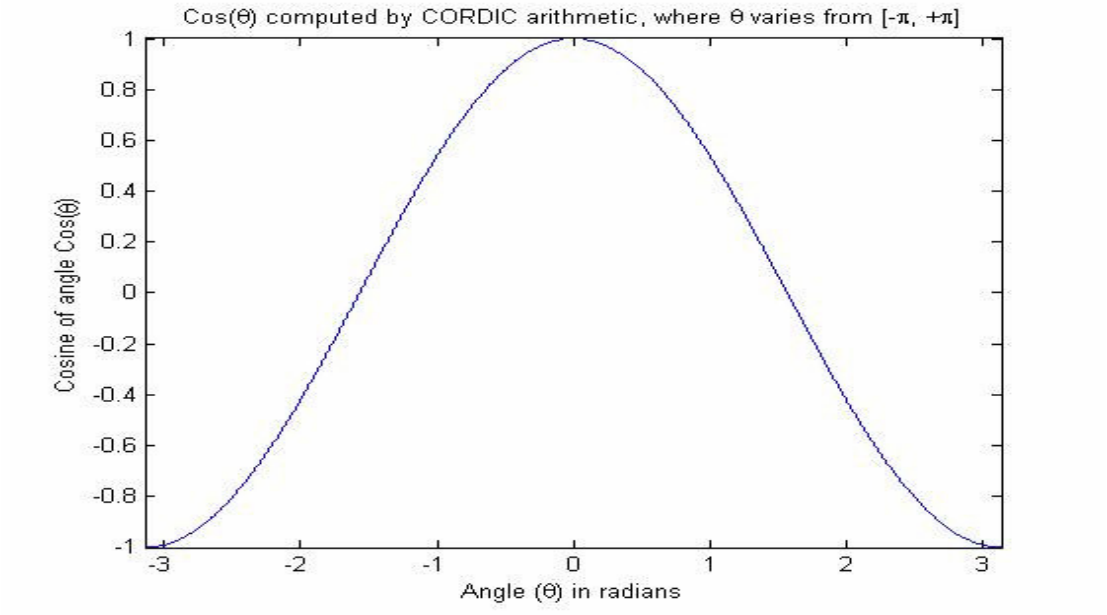

Figure 19: Cosine calculation from -pi, pi by CORDIC arithmetic

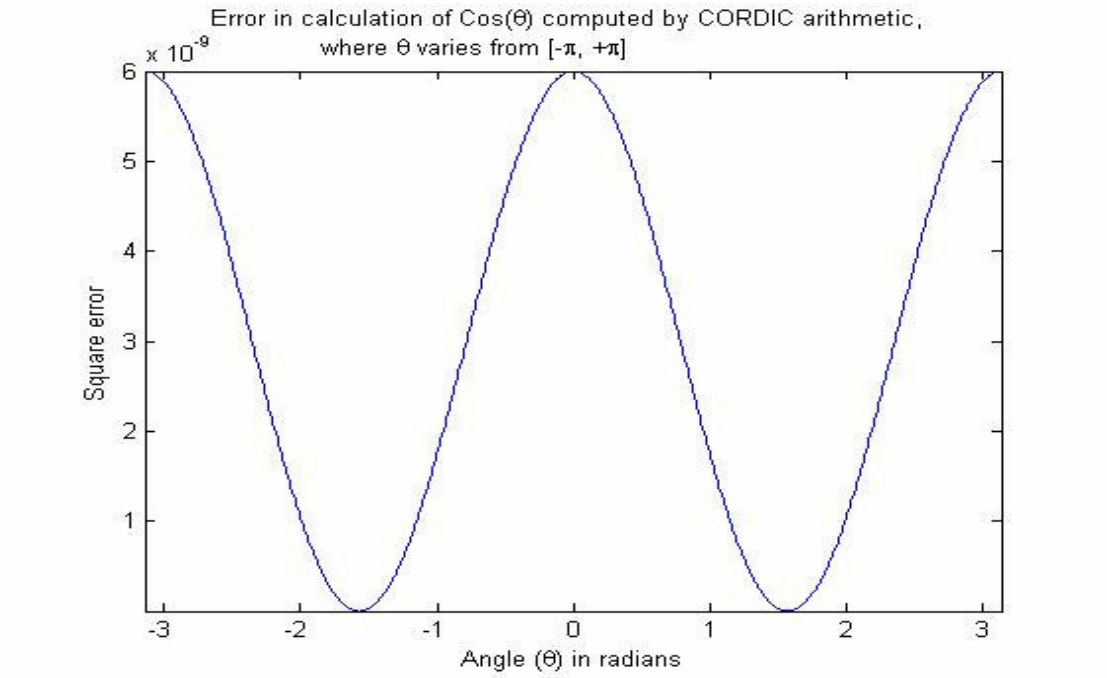

Figure 20: Error Calculation for Cosine from -pi, pi for CORDIC arithmetic

The Error Calculation graph suggests that the maximum error of the order of 10^{-9} occurs at the values $\pm\pi$, 0. This is because computing the output values ± 1 use the complete rotation cycles.

4.4.3 Calculation of $\tan^{-1}$

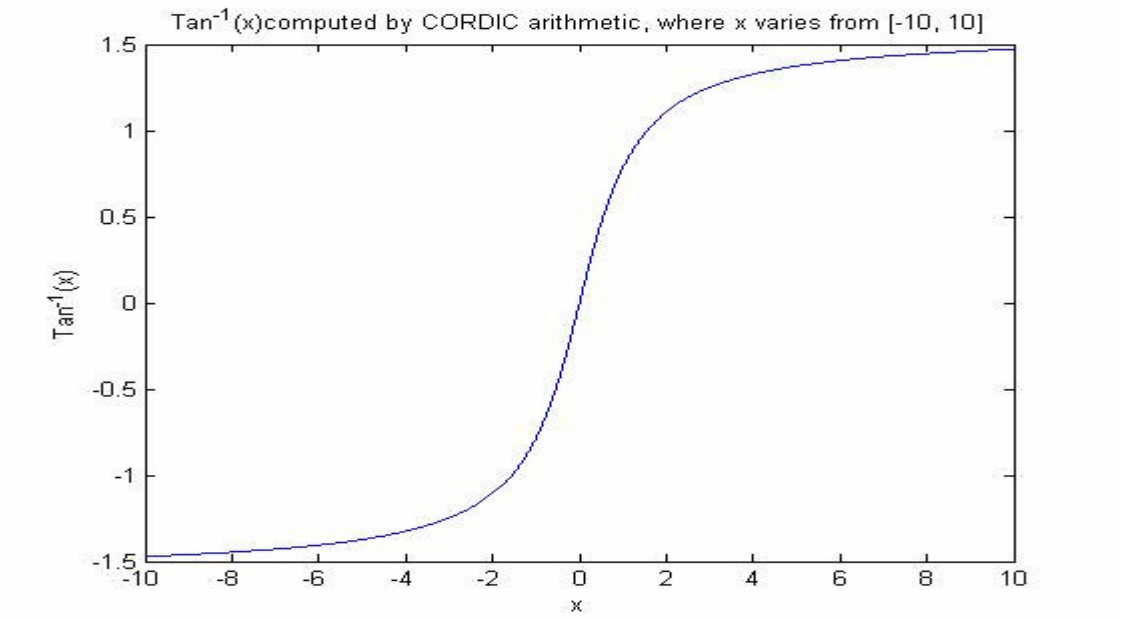

Figure 21: Inverse Tan calculation from -10, 10 by CORDIC arithmetic

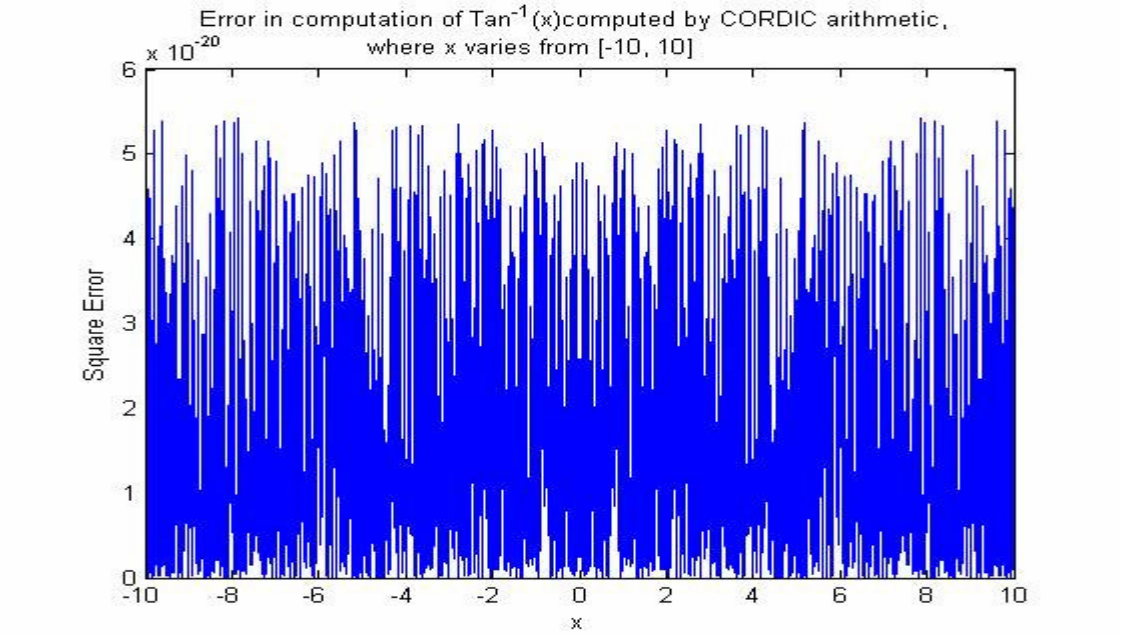

Figure 22: Error Calculation for Inverse Tan from -10, 10 from CORDIC arithmetic

As expected, the error for calculation of inverse tangent is unpredictable since the CORDIC engine is working in the vectoring mode. However, bounds for the error can be found and is expected to be lower that that observed in rotation mode. This can also be seen in the above figure where the error is of the order of 10^{-20}.

5 Optimization

5.1 Optimization Overview

Normal SVD computation for a given Matrix would require 8n cycles using a single CORDIC engine with n bit accuracy. This causes a huge delay. Optimal number of CORDIC engine instances were used so as to achieve parallel processing. Structural RTL was used wherever possible in order to optimize the delay though logic blocks.

5.1.1 Timing Optimization

To increase the frequency for the worst case path, the following schemes were implemented.

- Gate-level optimized Adder

Although the synthesis tool (Design Vision) supports the carry-look-ahead adder by default, a parallel-prefix tree adder "Kogge-Stone" was implemented by writing the structural RTL.
Delay for a 32 bit Carry Look Ahead Adder: $delay \approx 2\log_2 n$
Delay for a 32 bit Kogge Stone Adder: $delay \approx \log_2 n$
This helps in reducing the delay through the critical path since the CORDIC engine fir the SVD uses Adders at various stages.

- Parallel Diagonalization Method

Two CORIC Engine Instances were used to achieve parallelization wherever possible. The θ_{SUM} and θ_{DIFF} are computed in parallel in 32 cycles denoted by T_1.by using two CORDIC modules. The separate θ_l and θ_r are computed in parallel using an adder followed by a shifter. Finally the two sided CORDIC rotation is performed which takes 3 CORDIC 32 clock cycles; denoted by T_2, T_3 and T_4 respectively. The total time for a CORDIC 2x2 SVD, T_{CSVD}, is
$T_{CSVD} = T_1 + T_2 + T_3 + T_4$

5.1.2 Area Optimization

The area of the architecture is approximately twice that of the cordic engine, the interface logic including the programmable registers, the control logic and the overhead area which includes the glue logic.

$$A_{CSVD} = 2 \cdot A_{CORDIC} + A_{PROG\_REG} + A_{CONTROL} + A_{OH}$$

5.1.3 Implementation Versions

The CORDIC SVD IP was developed in steps. There are 5 different versions we had implemented. Table 5.1 shows the different versions.

Version Number	SIN/COS TAN-1	Generic Rotation Mode	Generic Vectoring Mode	SVD	Structural Adder	Coarse Rotation
1.	+	-	-	-	-	-
2.	+	-	-	+	-	-
3.	+	+	+	+	-	-
4.	+	+	+	+	+	-
5.	+	+	+	+	+	+

Table 24: Comparison of difference of each SVD CORDIC IP version

5.2 Bugs and Fixes

Phase Default Format Issue

In some cases while computing the value for θ_r and θ_l using θ_{SUM} and θ_{DIFF} the computed angle could exceed π and the default phase format we used was 1.2.29 which had to be changed to 1.3.28 to support the case when the sum or difference values were above π.

Coarse Rotation Issue

Coarse Rotation algorithm was not implemented initially and was added in the final version to make the CORDIC engine work for all values between $-\pi$ to $+\pi$.

CONTROL Register Programming

CONTROL Register was initially designed to have 5 bits which included 2 bits for function. An additional generic vectoring mode was added in version 3 which had to change the CONTROL register size to 6 bits and 3 bits for the function. This change had to accommodated in the rest of the code else to avoid incorrect mode programming.

5.3 Results

5.3.1 Synthesis

Tool Used - *Synopsys Design Vision 2004.06-SP2.*
All modules were synthesized without any errors

5.3.2 Timing Analysis

Tool Used - *Synopsys Design Vision 2004.06-SP2*

The timing for the worst case path delay was reported as 9940.08 ps. Hence, the current design can operate at a frequency of 100MHz.
The delay is mainly through the multiplier block with the synthesis tool auto-generates as an Array Multiplier the delay for which is huge. This delay can be reduced considerably using a Tree Multiplier (Wallace or Dadda). In the interest of time the structural RTL for this was not implemented. With the use of an efficient multiplier in the system the operating frequency can go up to 400 MHz.
For detailed report on Timing Analysis refer to Appendix C.1

5.3.3 Area Analysis

Tools Used - *Synopsys Design Vision 2004.06-SP2*
Silicon Ensemble Software in Cadence

The approximate area computed using Synopsys Design Vision is 3296853.25 square microns. The APR tool was used to get the actual layout. For detailed report on Area Analysis refer to Appendix C.2.

5.3.4 Power Analysis

Tool Used - *Synopsys Design Vision 2004.06-SP2*

The power number was computed as 1.2227 mW. For detailed report on Power Analysis refer to Appendix C.3

6 Conclusion

In order to meet the specification of SVD CORDIC IP, we considered the design in various stages. The first stage was to get the basic CORDIC Engine to function in the Sin()/Cos() and InvTan Mode. Next the SVD Mode was implemented with used the CORDIC engine for computation. In order to develop a top level IP which performed all basic CORDIC Operations along with the SVD a generic rotation and vectoring mode were added and made accessible at the top level. The flexibility for the user to program the SVD CORDIC IP with programmable bit allocation for the integer and fractional part was done. Finally the complete system was integrated to give a performance of over 100MHz with and area of 3500000 square microns.

In the SVD CORDIC IP the maximum delay is mainly through the multiplier block with the synthesis tool auto-generates as an Array Multiplier the delay for which is huge. This delay can be reduced considerably using a Tree Multiplier (Wallace or Dadda). In the interest of time the structural RTL for this was not implemented. With the use of an efficient multiplier in the system the operating frequency can go up to 400 MHz. An overall optimization of logic for the multiplier would be taken into consideration for future work.

We developed all SVD CORDIC Verilog RTL from scratch. During the course of the project several problems were encountered and handled efficiently. One of the big problems was related to the manipulation of sign bit and shifter in order to provide the flexible formatting option. After solving the problems, we have known the real meaning of 2's complement number world. This project gave us an opportunity to understand better both the of the world of VLSI and Computer Arithmetic. It also gave us a deep insight into number systems and a complete mathematical IP Design.

7 References

[1] Jack E. Volder, The CORDIC Trigonometric Computing Technique*, IRE Transactions on Electronic Computers*, September 1959

[2] R. Andraka, "A Survey of CORDIC Algorithms for FPGAs," in *Proceedings of the 1998 ACM/SIGDA Sbcth International Symposium on Field Programmable Gate Arrays (FPGA '98)*, Monterey, CA, Feb. 22-24, 1998, pp. 191-200.

[3] J.R. Cavallaro and F.T. Luk, "CORDIC Arithmetic for a SVD Processor", *Journal for Parallel and Distributed Computing,* vol. 5, pp. 271-290, 1988

[4] AMBA Specifications (Rev 2.0), *http://www.gaisler.com/doc/amba.pdf,* (accessed 20th November, 2006)

[5] R. P. Brent, F. T. Luk, and C. F. Van Loan, "Computation of the singular value decomposition using mesh-connected processors," *Journal for VLSI Computer Systems*, vol. 1, no. 3, pp. 243-270, 1985

[6] J.R. Cavallaro and F.T. Luk, "Architectures for a CORDIC SVD processor", *SPIE Proceedings Vol. 698*. Edited by William J. Miceli. Bellingham, WA: Society for Photo-Optical Instrumentation Engineers, 1986., p.45

[7] Nariankadu D. Hemkumar, "A Systolic VLSI Architecture for Complex SVD", *Masters Thesis*, Rice University, Houston, Texas, 1991

[8] Inwook Kong, Jungho Jo, "Design of High-Speed CORDIC IP", *VLSI-I Project Final Report*, Submitted to Prof Adnan Aziz, 2005 (http://www.ece.utexas.edu/~adnan/grad-vlsi-06/cordic_sp2.tar.gz)

8 Appendix A – Verilog Codes

Main.v : Top Level Wrapper

```
module main(PRESETn, PCLK, PSEL, PADDR, PENABLE, PWRITE, PWDATA, PRDATA, INT);


input PRESETn, PCLK;                    // Reset and clock
input PSEL, PENABLE, PWRITE;
input [5:0] PADDR;
input [31:0] PWDATA;


output INT;
output [31:0] PRDATA;


wire sys_cordic_done, clear_control_bit;
wire [31:0] out1, out2, out3, out4, out5, out6;
wire [5:0] CONTROL;
wire [31:0] a, b, c, d;

wire [4:0] XYFRACBASE, PHASEFRACBASE;


intf apb_intf(.PRESETn(PRESETn), .PCLK(PCLK), .PSEL(PSEL), .PADDR(PADDR),
.PENABLE(PENABLE), .PWRITE(PWRITE), .PWDATA(PWDATA), .PRDATA(PRDATA), .INT(INT),
              .sys_cordic_done(sys_cordic_done), .clear_control_bit(clear_control_bit),
              .cordic_out1(out1), .cordic_out2(out2), .cordic_out3(out3),
.cordic_out4(out4), .cordic_out5(out5), .cordic_out6(out6),
              .CONTROL(CONTROL),
              .PROG_A(a), .PROG_B(b), .PROG_C(c), .PROG_D(d),
             .XYFRACBASE(XYFRACBASE), .PHASEFRACBASE(PHASEFRACBASE));

sysfsm cordic_sysfsm (.rst_n(PRESETn), .clk(PCLK), .func(CONTROL[3:1]), .a(a), .b(b),
.c(c), .d(d), .new_cmd(CONTROL[0]),
               .clear_control_bit(clear_control_bit), .sys_cordic_done(sys_cordic_done),
.out1(out1), .out2(out2), .out3(out3), .out4(out4), .out5(out5), .out6(out6),
.XYFRACBASE(XYFRACBASE), .PHASEFRACBASE(PHASEFRACBASE));

endmodule
```

intf.v : AMBA Interface

```
module intf(PRESETn, PCLK, PSEL, PADDR, PENABLE, PWRITE, PWDATA, PRDATA, INT,
            sys_cordic_done, clear_control_bit,
            cordic_out1, cordic_out2, cordic_out3, cordic_out4, cordic_out5, cordic_out6,
            CONTROL,
            PROG_A, PROG_B, PROG_C, PROG_D,
            XYFRACBASE, PHASEFRACBASE);

input PRESETn, PCLK;                  // Reset and clock
input PSEL, PENABLE, PWRITE;
input [5:0] PADDR;
input [31:0] PWDATA;
input sys_cordic_done, clear_control_bit;
input [31:0] cordic_out1, cordic_out2, cordic_out3, cordic_out4, cordic_out5,
cordic_out6;

output INT;
output [31:0] PRDATA;
output [5:0] CONTROL;
output [31:0] PROG_A, PROG_B, PROG_C, PROG_D;
output [4:0] XYFRACBASE, PHASEFRACBASE;

// Registers
reg [5:0] CONTROL;     // 0x0
```

```
reg [31:0] PROG_A;     // 0x4
reg [31:0] PROG_B;     // 0x8
reg [31:0] PROG_C;     // 0xC
reg [31:0] PROG_D;     // 0x10
reg [31:0] OUT1;       // 0x14
reg [31:0] OUT2;       // 0x18
reg [31:0] OUT3;       // 0x1C
reg [31:0] OUT4;       // 0x20
reg [31:0] OUT5;       // 0x24
reg [31:0] OUT6;       // 0x28
// Format dependent factors
reg [4:0] XYFRACBASE;         // 0x2C
reg [4:0] PHASEFRACBASE;      // 0x30

always @(posedge PCLK)
begin
        if (~PRESETn)
                begin
                // Control Register is done in a different process
                PROG_A          <= 32'h0;      // 0x04
                PROG_B          <= 32'h0;      // 0x08
                PROG_C          <= 32'h0;      // 0x0C
                PROG_D          <= 32'h0;      // 0x10
                // Skipping output registers
                // Output Registers are read only registers
                XYFRACBASE      <= 5'h10;      // 0x2C        // Reset Value = 16 (i.e. 16
Frac Bits)
                PHASEFRACBASE   <= 5'h1C;      // 0x30        // Reset Value = 28 (i.e. 28
Frac Bits)
                end
        else
                begin
                if (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b1)
                        begin
                        case(PADDR)
                        6'h04: PROG_A          <= PWDATA;     // 0x04
                        6'h08: PROG_B          <= PWDATA;     // 0x08
                        6'h0C: PROG_C          <= PWDATA;     // 0x0C
                        6'h10: PROG_D          <= PWDATA;     // 0x10
                        // Skipping output registers as they are read only
                        6'h2C: XYFRACBASE      <= PWDATA;     // 0x2C
                        6'h30: PHASEFRACBASE          <= PWDATA;     // 0x30
                        endcase
                        end
                end
end


always @(posedge PCLK)
begin
        if (~PRESETn)
                CONTROL  <= 6'h00;
        else
                begin
                if (clear_control_bit == 1'b1)
                        CONTROL[0] <= 1'b0;
                else if (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b1 && PADDR ==
6'h00)
                        CONTROL[0] <= PWDATA[0];

                 if (sys_cordic_done  == 1'b1)
                          CONTROL[4] <= 1'b1;
                else if (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b1 && PADDR ==
6'h00)
                        CONTROL[4] <= PWDATA[4];

                if (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b1 && PADDR == 6'h00)
                        begin
                        CONTROL[3:1] <= PWDATA[3:1];
                        CONTROL[5]   <= PWDATA[5];
                        end
```

```
                end
end

always @(posedge PCLK)
begin
        if (~PRESETn)
                begin
                 OUT1   <= 32'h0;
                 OUT2   <= 32'h0;
                 OUT3   <= 32'h0;
                 OUT4   <= 32'h0;
                 OUT5   <= 32'h0;
                 OUT6   <= 32'h0;
                end
        else if (sys_cordic_done == 1'b1)
                begin
                OUT1    <= cordic_out1;
                OUT2    <= cordic_out2;
                OUT3    <= cordic_out3;
                OUT4    <= cordic_out4;
                OUT5    <= cordic_out5;
                OUT6    <= cordic_out6;
                end
end



assign PRDATA = (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h00)?
{27'h0, CONTROL} :
                 (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h04)?
PROG_A  :
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h08)?
PROG_B  :       // 0x8
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h0C)?
PROG_C  :       // 0xC
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h10)?
PROG_D  :       // 0x10
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h14)?
OUT1    :       // 0x14
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h18)?
OUT2    :       // 0x18
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h1C)?
OUT3    :       // 0x1C
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h20)?
OUT4    :       // 0x20
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h24)?
OUT5    :       // 0x24
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h28)?
OUT6    :       // 0x28
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h2C)?
{27'h0, XYFRACBASE}:          // 0x2C
                (PSEL == 1'b1 && PENABLE == 1'b1 && PWRITE == 1'b0 && PADDR == 6'h30)?
{27'h0, PHASEFRACBASE} : 32'h0;       // 0x30



assign INT = (CONTROL[4] & CONTROL[5]);


endmodule
```

Sysfsm.v : System FSM module

```
module sysfsm (rst_n, clk, func, a, b, c, d, new_cmd, clear_control_bit, sys_cordic_done,
out1, out2, out3, out4, out5, out6,
              XYFRACBASE, PHASEFRACBASE);

input rst_n, clk;                     // Reset and Clock
input [2:0] func;
```

```
input [31:0] a, b, c, d;
input new_cmd;

output clear_control_bit, sys_cordic_done;
output [31:0] out1, out2, out3, out4, out5, out6;

input [4:0] XYFRACBASE, PHASEFRACBASE;


`define SYSFSM_IDLE    1'h0
`define SYSFSM_BUSY    1'h1

reg sysfsm_state;
reg clear_control_bit, sys_cordic_done, start_cordic, update_tan_table;
reg [2:0] cordic_func;
reg [31:0] cordic_a, cordic_b, cordic_c, cordic_d;

reg [31:0] SCALE_FACTOR, XYBASEONE;
reg [31:0] INVTAN0, INVTAN1, INVTAN2, INVTAN3, INVTAN4, INVTAN5, INVTAN6, INVTAN7,
INVTAN8, INVTAN9, INVTAN10;
reg [31:0] INVTAN11, INVTAN12, INVTAN13, INVTAN14, INVTAN15, INVTAN16, INVTAN17,
INVTAN18, INVTAN19, INVTAN20;
reg [31:0] INVTAN21, INVTAN22, INVTAN23, INVTAN24, INVTAN25, INVTAN26, INVTAN27,
INVTAN28, INVTAN29, INVTAN30, INVTAN31;

wire cordic_done;

always @(posedge clk)
begin
        if (~rst_n)
                begin
                sysfsm_state <= `SYSFSM_IDLE;
                cordic_func  <= 3'h0;
                cordic_a     <= 32'h0;
                cordic_b     <= 32'h0;
                cordic_c     <= 32'h0;
                cordic_d     <= 32'h0;
                sys_cordic_done   <= 1'b0;
                clear_control_bit <= 1'b0;
                start_cordic      <= 1'b0;
                update_tan_table  <= 1'b0;
                end
        else
                begin
                case (sysfsm_state)
                `SYSFSM_IDLE:
                        begin
                        sys_cordic_done  <= 1'b0;

                        if (new_cmd == 1'b1)
                                begin
                                update_tan_table  <= 1'b1;    // To update the Tan Table
and other format dependent constants
                                clear_control_bit <= 1'b1;    // To indicate that the
commnand has been registered
                                cordic_func       <= func;
                                cordic_a          <= a;
                                cordic_b          <= b;
                                cordic_c          <= c;
                                cordic_d          <= d;
                                start_cordic      <= 1'b1;
                                sysfsm_state      <= `SYSFSM_BUSY;
                                end
                        end

                `SYSFSM_BUSY:
                        begin
                        clear_control_bit     <= 1'b0;
                        start_cordic          <= 1'b0;
                        update_tan_table        <= 1'b0;
                        if (cordic_done == 1'b1)
```

```
                                begin
                                sys_cordic_done <= 1'b1;
                                if (new_cmd == 1'b1)
                                        begin
                                        update_tan_table  <= 1'b1;
                                        clear_control_bit <= 1'b1;
                                        cordic_func       <= func;
                                        cordic_a          <= a;
                                        cordic_b          <= b;
                                        cordic_c          <= c;
                                        cordic_d          <= d;
                                        start_cordic      <= 1'b1;
                                        sysfsm_state      <= `SYSFSM_BUSY;
                                        end

                                else
                                        begin
                                        sysfsm_state    <= `SYSFSM_IDLE;
                                        end
                                end
                        else
                                begin
                                sys_cordic_done <= 1'b0;
                                end

                        end

                endcase
                end

end

always @(posedge clk)
begin
        if (~rst_n)
                begin
                 SCALE_FACTOR    <= 32'h0;                    // 0.6073 * 2^B1
                 XYBASEONE       <= 32'h0;                              // 1 * 2^B1
                 // Registers holding inverse tan table INVTANi = tan^{-1} (2^{-i})
                 // Reset Values based on 2Q29 format. i.e. dec2hex(round(2^B2 *
(inv_tan(2^(-i))))
                 INVTAN0         <= 32'h0;
                 INVTAN1         <= 32'h0;
                 INVTAN2         <= 32'h0;
                 INVTAN3         <= 32'h0;
                 INVTAN4         <= 32'h0;
                 INVTAN5         <= 32'h0;
                 INVTAN6         <= 32'h0;
                 INVTAN7         <= 32'h0;
                 INVTAN8         <= 32'h0;
                 INVTAN9         <= 32'h0;
                 INVTAN10        <= 32'h0;
                 INVTAN11        <= 32'h0;
                 INVTAN12        <= 32'h0;
                 INVTAN13        <= 32'h0;
                 INVTAN14        <= 32'h0;
                 INVTAN15        <= 32'h0;
                 INVTAN16        <= 32'h0;
                 INVTAN17        <= 32'h0;
                 INVTAN18        <= 32'h0;
                 INVTAN19        <= 32'h0;
                 INVTAN20        <= 32'h0;
                 INVTAN21        <= 32'h0;
                 INVTAN22        <= 32'h0;
                 INVTAN23        <= 32'h0;
                 INVTAN24        <= 32'h0;
                 INVTAN25        <= 32'h0;
                 INVTAN26        <= 32'h0;
                 INVTAN27        <= 32'h0;
                 INVTAN28        <= 32'h0;
                 INVTAN29        <= 32'h0;
```

```
                INVTAN30        <= 32'h0;
                INVTAN31        <= 32'h0;
              end
       else
              begin
              if (update_tan_table == 1'b1)
                     begin
                      SCALE_FACTOR    <= (32'h26DD3B6A >> (5'h1E - XYFRACBASE));
                      XYBASEONE       <= (32'h00000001 << XYFRACBASE);
                      // Registers holding inverse tan
                      // Reset Values based on 2Q29 fo
                      INVTAN0              <= (32'h3243F6A9 >> (5'h1E -
PHASEFRACBASE));
                      INVTAN1              <= (32'h1DAC6705 >> (5'h1E -
PHASEFRACBASE));
                      INVTAN2              <= (32'h0FADBAFD >> (5'h1E -
PHASEFRACBASE));
                      INVTAN3              <= (32'h07F56EA7 >> (5'h1E -
PHASEFRACBASE));
                      INVTAN4              <= (32'h03FEAB77 >> (5'h1E -
PHASEFRACBASE));
                      INVTAN5              <= (32'h01FFD55C >> (5'h1E -
PHASEFRACBASE));
                      INVTAN6              <= (32'h00FFFAAB >> (5'h1E -
PHASEFRACBASE));
                      INVTAN7              <= (32'h007FFF55 >> (5'h1E -
PHASEFRACBASE));
                      INVTAN8              <= (32'h003FFFEB >> (5'h1E -
PHASEFRACBASE));
                      INVTAN9              <= (32'h001FFFFD >> (5'h1E -
PHASEFRACBASE));
                      INVTAN10     <= (32'h00100000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN11     <= (32'h00080000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN12     <= (32'h00040000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN13     <= (32'h00020000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN14     <= (32'h00010000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN15     <= (32'h00008000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN16     <= (32'h00004000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN17     <= (32'h00002000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN18     <= (32'h00001000 >> (5'h1E - PHASEFRACBASE));
                      INVTAN19     <= (32'h00000800 >> (5'h1E - PHASEFRACBASE));
                      INVTAN20     <= (32'h00000400 >> (5'h1E - PHASEFRACBASE));
                      INVTAN21     <= (32'h00000200 >> (5'h1E - PHASEFRACBASE));
                      INVTAN22     <= (32'h00000100 >> (5'h1E - PHASEFRACBASE));
                      INVTAN23     <= (32'h00000080 >> (5'h1E - PHASEFRACBASE));
                      INVTAN24     <= (32'h00000040 >> (5'h1E - PHASEFRACBASE));
                      INVTAN25     <= (32'h00000020 >> (5'h1E - PHASEFRACBASE));
                      INVTAN26     <= (32'h00000010 >> (5'h1E - PHASEFRACBASE));
                      INVTAN27     <= (32'h00000008 >> (5'h1E - PHASEFRACBASE));
                      INVTAN28     <= (32'h00000004 >> (5'h1E - PHASEFRACBASE));

                      INVTAN29     <= (32'h00000002 >> (5'h1E - PHASEFRACBASE));

                      INVTAN30     <= (32'h00000001 >> (5'h1E - PHASEFRACBASE));
                      INVTAN31     <= (32'h00000001 >> (5'h1E - PHASEFRACBASE));
                     end
              end

end

control cordic_control(.rst_n(rst_n), .clk(clk), .cordic_start(start_cordic),
.cordic_func(cordic_func), .a(cordic_a), .b(cordic_b), .c(cordic_c), .d(cordic_d),
.out1(out1), .out2(out2), .out3(out3), .out4(out4), .out5(out5), .out6(out6),
.write_op(cordic_done), .XYFRACBASE(XYFRACBASE), .PHASEFRACBASE(PHASEFRACBASE),
.SCALE_FACTOR(SCALE_FACTOR), .XYBASEONE(XYBASEONE), .INVTAN0(INVTAN0), .INVTAN1(INVTAN1),
.INVTAN2(INVTAN2), .INVTAN3(INVTAN3), .INVTAN4(INVTAN4), .INVTAN5(INVTAN5),
.INVTAN6(INVTAN6), .INVTAN7(INVTAN7), .INVTAN8(INVTAN8), .INVTAN9(INVTAN9),
.INVTAN10(INVTAN10), .INVTAN11(INVTAN11), .INVTAN12(INVTAN12), .INVTAN13(INVTAN13),
.INVTAN14(INVTAN14), .INVTAN15(INVTAN15), .INVTAN16(INVTAN16), .INVTAN17(INVTAN17),
.INVTAN18(INVTAN18), .INVTAN19(INVTAN19), .INVTAN20(INVTAN20), .INVTAN21(INVTAN21),
.INVTAN22(INVTAN22), .INVTAN23(INVTAN23), .INVTAN24(INVTAN24), .INVTAN25(INVTAN25),
```

```
.INVTAN26(INVTAN26), .INVTAN27(INVTAN27), .INVTAN28(INVTAN28), .INVTAN29(INVTAN29),
.INVTAN30(INVTAN30), .INVTAN31(INVTAN31));

endmodule
```

Control.v: CORDIC co-processor controller

```
module control (rst_n, clk, cordic_start, cordic_func, a, b, c, d, out1, out2, out3,
out4, out5, out6, write_op,
               XYFRACBASE, PHASEFRACBASE,
               SCALE_FACTOR, XYBASEONE,
                INVTAN0, INVTAN1, INVTAN2, INVTAN3, INVTAN4, INVTAN5, INVTAN6, INVTAN7,
INVTAN8, INVTAN9, INVTAN10,
                INVTAN11, INVTAN12, INVTAN13, INVTAN14, INVTAN15, INVTAN16, INVTAN17,
INVTAN18, INVTAN19, INVTAN20,
                INVTAN21, INVTAN22, INVTAN23, INVTAN24, INVTAN25, INVTAN26, INVTAN27,
INVTAN28, INVTAN29, INVTAN30, INVTAN31);

input rst_n, clk;                    // Reset and Clock
input [31:0] a, b, c, d;             // Inputs
input cordic_start;
input [2:0] cordic_func;
output [31:0] out1, out2, out3, out4, out5, out6;
output write_op;

input [4:0] XYFRACBASE, PHASEFRACBASE;
input [31:0] SCALE_FACTOR, XYBASEONE;
input [31:0] INVTAN0, INVTAN1, INVTAN2, INVTAN3, INVTAN4, INVTAN5, INVTAN6, INVTAN7,
INVTAN8, INVTAN9, INVTAN10;
input [31:0] INVTAN11, INVTAN12, INVTAN13, INVTAN14, INVTAN15, INVTAN16, INVTAN17,
INVTAN18, INVTAN19, INVTAN20;
input [31:0] INVTAN21, INVTAN22, INVTAN23, INVTAN24, INVTAN25, INVTAN26, INVTAN27,
INVTAN28, INVTAN29, INVTAN30, INVTAN31;

`define FSM_Idle      3'h0
`define FSM_WAIT_DONE   3'h1
`define FSM_SVD_STAGE2  3'h2
`define FSM_SVD_STAGE3  3'h3
`define FSM_SVD_STAGE4  3'h4
`define FSM_SVD_WAIT_DONE 3'h5

`define FUNC_SINCOS  3'h0
`define FUNC_INVTAN  3'h1
`define FUNC_VECROT  3'h2
`define FUNC_SVD     3'h3
`define FUNC_GEN_VEC 3'h4

reg [2:0] fsm_state;
reg [31:0] x_c1, y_c1, z_c1, x_c2, y_c2, z_c2;
reg [31:0] out1, out2, out3, out4, out5, out6;
reg [31:0] eig1, eig2, theta_r, theta_l;
reg start_c1, start_c2, mode_c1, mode_c2, write_op;

wire [31:0] sig_theta_r, sig_theta_l;
wire [31:0] temp_theta_r, temp_theta_l;
wire [31:0] x_n_c1, y_n_c1, z_n_c1, x_n_c2, y_n_c2, z_n_c2;
wire done_c1, done_c2;

always @(posedge clk)
begin
       if (~rst_n)
               begin
               fsm_state      <= `FSM_Idle ;
               // Cordic Processor 1
               x_c1           <= 32'h0;
               y_c1           <= 32'h0;
               z_c1           <= 32'h0;
               mode_c1        <= 1'b0;
               start_c1       <= 1'b0;
```

```
                // Cordic Processor 2
                x_c2            <= 32'h0;
                y_c2            <= 32'h0;
                z_c2            <= 32'h0;
                mode_c2         <= 1'b0;
                start_c2        <= 1'b0;
                // Answers
                write_op        <= 1'b0;
                out1            <= 32'h0;
                out2            <= 32'h0;
                out3            <= 32'h0;
                out4            <= 32'h0;
                out5            <= 32'h0;
                out6            <= 32'h0;
                // Intermediate for SVD
                theta_r         <= 32'h0;
                theta_l         <= 32'h0;
                eig1            <= 32'h0;
                eig2            <= 32'h0;
                end
        else
                begin
                case(fsm_state)
                `FSM_Idle:
                        begin
                        write_op <= 1'b0;             // Pull down write signal
                        if (cordic_start == 1'b1)
                                begin
                                if (cordic_func == `FUNC_SINCOS)
                                        begin
                                        x_c1      <= XYBASEONE ;       // x0 = 1 for sin/cos
calculations
                                        y_c1      <= 32'h0;            // y0 = 0 for sin/cos
calculations
                                        z_c1      <= a;                // Will Compute y =
Sin(a), x = Cos(a)
                                        mode_c1   <= 1'b0;             // Rotation Mode
                                        start_c1  <= 1'b1;
                                        fsm_state <= `FSM_WAIT_DONE;
                                        end
                                else if (cordic_func == `FUNC_INVTAN)
                                        begin
                                        x_c1      <= XYBASEONE;               // x0 = 1 for
inv_tan calculation
                                        y_c1      <= a;                // will compute z =
inv_tan(a)
                                        z_c1      <= 32'h0;            // z0 = 0 for inv_tan
calculation
                                        mode_c1   <= 1'b1;             // Vectoring mode
                                        start_c1  <= 1'b1;
                                        fsm_state <= `FSM_WAIT_DONE;
                                        end
                                else if (cordic_func == `FUNC_VECROT)
                                        begin
                                        x_c1      <= a;                // Vector to be
rotated = (a, b)
                                        y_c1      <= b;                //
                                        z_c1      <= c;                // Angle by which
rotation has to be done
                                        mode_c1   <= 1'b0;             // Rotation Mode
                                        start_c1  <= 1'b1;
                                        fsm_state <= `FSM_WAIT_DONE;
                                        end
                                else if  (cordic_func == `FUNC_GEN_VEC)      // Generic
Vectoring mode
                                        begin
                                          x_c1      <= a;                  // Vector to be
rotated = (a, b)
                                          y_c1      <= b;                  //
                                          z_c1      <= c;                  // Phase
                                          mode_c1   <= 1'b1;               // Vectoring Mode
```

```
                                        start_c1  <= 1'b1;
                                        fsm_state <= `FSM_WAIT_DONE;
                                    end
                            else if (cordic_func == `FUNC_SVD)
                                    begin
                                    // Cordic Processor 1
                                    x_c1      <= d - a;           // Stage1 :
Calculation of Theta_{sum}
                                    y_c1      <= c + b;           // inv_tan (c+b / d-a)
= theta_{sum}
                                    z_c1      <= 32'h0;           // z0 = 0 for inv_tan
calculation
                                    mode_c1   <= 1'b1;            // Vectoring mode
                                    start_c1  <= 1'b1;
                                    // Cordic Processor 2
                                    x_c2      <= d + a;           // Stage1 :
Calculation of Theta_{diff}
                                    y_c2      <= c - b;           // inv_tan (c-b / d+a)
= theta_{diff}
                                    z_c2      <= 32'h0;           // z0 = 0 for inv_tan
calculation
                                    mode_c2   <= 1'b1;            // Vectoring mode
                                    start_c2  <= 1'b1;
                                    fsm_state <= `FSM_SVD_STAGE2;    // Go to state 2
                                    end
                            end
                    end

            `FSM_WAIT_DONE:
                    begin
                    start_c1  <= 1'b0;
                    if (done_c1 == 1'b1)
                            begin
                            write_op <= 1'b1;
                            out1     <= x_n_c1;
                            out2    <= y_n_c1;
                            out3     <= z_n_c1;
                            out4     <= 32'h0;
                            out5     <= 32'h0;
                            out6    <= 32'h0;
                            fsm_state <= `FSM_Idle;
                            end
                    end

            `FSM_SVD_STAGE2:
                    begin
                    start_c1 <= 1'b0;
                    start_c2 <= 1'b0;
                    if (done_c1 == 1'b1 && done_c2 == 1'b1)
                            begin
                            // Register theta_r, theta_l
                            theta_r  <= sig_theta_r;
                            theta_l  <= sig_theta_l;
                            // Rotate vector (a,b) by theta_r
                            x_c1     <= a;
                            y_c1     <= b;
                            z_c1     <= sig_theta_r;
                            mode_c1  <= 1'b0;
                            start_c1 <= 1'b1;
                            // Rotate vector (c,d) by theta_r
                            x_c2     <= c;
                            y_c2     <= d;
                            z_c2     <= sig_theta_r;
                            mode_c2  <= 1'b0;
                            start_c2 <= 1'b1;
                            //
                            fsm_state <= `FSM_SVD_STAGE3;
                            end
                    end

            `FSM_SVD_STAGE3:
```

```
                        begin
                        start_c1 <= 1'b0;
                        start_c2 <= 1'b0;
                        if (done_c1 == 1'b1 && done_c2 == 1'b1)
                                begin
                                // Rotate vector (x1,x2) by theta_l
                                x_c1     <= x_n_c1;
                                y_c1     <= x_n_c2;
                                z_c1     <= theta_l;
                                mode_c1  <= 1'b0;
                                start_c1 <= 1'b1;
                                // Rotate vector (y1,y2) by theta_l
                                x_c2     <= y_n_c1;
                                y_c2     <= y_n_c2;
                                z_c2     <= theta_l;
                                mode_c2  <= 1'b0;
                                start_c2 <= 1'b1;
                                //
                                fsm_state <= `FSM_SVD_STAGE4;
                                end
                        end

                `FSM_SVD_STAGE4:
                        begin
                        start_c1 <= 1'b0;
                        start_c2 <= 1'b0;
                        if (done_c1 == 1'b1 && done_c2 == 1'b1)
                                begin
                                // Register the eignevalue op's
                                eig1     <= x_n_c1;
                                eig2     <= y_n_c2;
                                // Calculate sin/cos of theta_r
                                x_c1     <= XYBASEONE;
                                y_c1     <= 32'h0;
                                z_c1     <= theta_r;
                                mode_c1  <= 1'b0;
                                start_c1 <= 1'b1;
                                // Calculate sin/cos of theta_l
                                x_c2     <= XYBASEONE;
                                y_c2     <= 32'h0;
                                z_c2     <= theta_l;
                                mode_c2  <= 1'b0;
                                start_c2 <= 1'b1;
                                //
                                fsm_state <= `FSM_SVD_WAIT_DONE;
                                end
                        end

                `FSM_SVD_WAIT_DONE:
                        begin
                        start_c1  <= 1'b0;
                        start_c2  <= 1'b0;
                        if (done_c1 == 1'b1 && done_c2 == 1'b1)
                                begin
                                write_op <= 1'b1;
                                out1     <= eig1;
                                out2     <= eig2;
                                out3     <= x_n_c1;
                                out4     <= y_n_c1;
                                out5     <= x_n_c2;
                                out6     <= y_n_c2;
                                fsm_state <= `FSM_Idle;
                                end
                        end

                endcase
                end
end


cordic cordic1 (.rst_n(rst_n), .clk(clk), .x(x_c1), .y(y_c1), .z(z_c1), .mode(mode_c1),
```

```
.x_n(x_n_c1), .y_n(y_n_c1), .z_n(z_n_c1), .start(start_c1), .done(done_c1),
.XYFRACBASE(XYFRACBASE), .SCALE_FACTOR(SCALE_FACTOR), .XYBASEONE(XYBASEONE),
.PHASEFRACBASE(PHASEFRACBASE), .INVTAN0(INVTAN0), .INVTAN1(INVTAN1), .INVTAN2(INVTAN2),
.INVTAN3(INVTAN3), .INVTAN4(INVTAN4), .INVTAN5(INVTAN5), .INVTAN6(INVTAN6),
.INVTAN7(INVTAN7), .INVTAN8(INVTAN8), .INVTAN9(INVTAN9), .INVTAN10(INVTAN10),
.INVTAN11(INVTAN11), .INVTAN12(INVTAN12), .INVTAN13(INVTAN13), .INVTAN14(INVTAN14),
.INVTAN15(INVTAN15), .INVTAN16(INVTAN16), .INVTAN17(INVTAN17), .INVTAN18(INVTAN18),
.INVTAN19(INVTAN19), .INVTAN20(INVTAN20), .INVTAN21(INVTAN21), .INVTAN22(INVTAN22),
.INVTAN23(INVTAN23), .INVTAN24(INVTAN24), .INVTAN25(INVTAN25), .INVTAN26(INVTAN26),
.INVTAN27(INVTAN27), .INVTAN28(INVTAN28), .INVTAN29(INVTAN29), .INVTAN30(INVTAN30),
.INVTAN31(INVTAN31));

cordic cordic2 (.rst_n(rst_n), .clk(clk), .x(x_c2), .y(y_c2), .z(z_c2), .mode(mode_c2),
.x_n(x_n_c2), .y_n(y_n_c2), .z_n(z_n_c2), .start(start_c2), .done(done_c2),
.XYFRACBASE(XYFRACBASE), .SCALE_FACTOR(SCALE_FACTOR), .XYBASEONE(XYBASEONE),
.PHASEFRACBASE(PHASEFRACBASE), .INVTAN0(INVTAN0), .INVTAN1(INVTAN1), .INVTAN2(INVTAN2),
.INVTAN3(INVTAN3), .INVTAN4(INVTAN4), .INVTAN5(INVTAN5), .INVTAN6(INVTAN6),
.INVTAN7(INVTAN7), .INVTAN8(INVTAN8), .INVTAN9(INVTAN9), .INVTAN10(INVTAN10),
.INVTAN11(INVTAN11), .INVTAN12(INVTAN12), .INVTAN13(INVTAN13), .INVTAN14(INVTAN14),
.INVTAN15(INVTAN15), .INVTAN16(INVTAN16), .INVTAN17(INVTAN17), .INVTAN18(INVTAN18),
.INVTAN19(INVTAN19), .INVTAN20(INVTAN20), .INVTAN21(INVTAN21), .INVTAN22(INVTAN22),
.INVTAN23(INVTAN23), .INVTAN24(INVTAN24), .INVTAN25(INVTAN25), .INVTAN26(INVTAN26),
.INVTAN27(INVTAN27), .INVTAN28(INVTAN28), .INVTAN29(INVTAN29), .INVTAN30(INVTAN30),
.INVTAN31(INVTAN31));

assign temp_theta_r = z_n_c1 + z_n_c2;
assign temp_theta_l = z_n_c1 - z_n_c2;

assign sig_theta_r = (temp_theta_r[31]) ? (temp_theta_r >> 1'b1) | (32'h80000000):
(temp_theta_r >> 1'b1);
assign sig_theta_l = (temp_theta_l[31]) ? (temp_theta_l >> 1'b1) | (32'h80000000):
(temp_theta_l >> 1'b1);


endmodule
```

Cordic.v: CORDIC processor

```
module cordic (rst_n, clk, x, y, z, mode, x_n, y_n, z_n, start, done,
               XYFRACBASE, PHASEFRACBASE,
                SCALE_FACTOR, XYBASEONE,
                INVTAN0, INVTAN1, INVTAN2, INVTAN3, INVTAN4, INVTAN5, INVTAN6, INVTAN7,
INVTAN8, INVTAN9, INVTAN10,
                INVTAN11, INVTAN12, INVTAN13, INVTAN14, INVTAN15, INVTAN16, INVTAN17,
INVTAN18, INVTAN19, INVTAN20,
                INVTAN21, INVTAN22, INVTAN23, INVTAN24, INVTAN25, INVTAN26, INVTAN27,
INVTAN28, INVTAN29, INVTAN30, INVTAN31);


parameter N = 32;

input rst_n;                  // Reset
input clk;                    // Clock

input [N-1:0] x;              //
input [N-1:0] y;              //
input [N-1:0] z;              //
input mode;                   // Mode = '1' Vectoring, Mode = 0 Rotation
input start;                  // Indicate start of the codic iteration cycle

output [N-1:0] x_n;
output [N-1:0] y_n;
output [N-1:0] z_n;
output done;                  // Indicates end of cordic iteration cycles

input [4:0] XYFRACBASE, PHASEFRACBASE;
input [31:0] SCALE_FACTOR, XYBASEONE;
input [31:0] INVTAN0, INVTAN1, INVTAN2, INVTAN3, INVTAN4, INVTAN5, INVTAN6, INVTAN7,
INVTAN8, INVTAN9, INVTAN10;
input [31:0] INVTAN11, INVTAN12, INVTAN13, INVTAN14, INVTAN15, INVTAN16, INVTAN17,
```

```
INVTAN18, INVTAN19, INVTAN20;
input [31:0] INVTAN21, INVTAN22, INVTAN23, INVTAN24, INVTAN25, INVTAN26, INVTAN27,
INVTAN28, INVTAN29, INVTAN30, INVTAN31;

wire d;
reg [4:0] counter;
reg [31:0] x_prev, y_prev, z_prev;
reg start_iter;

wire [31:0] sig_x_prev_d0, sig_y_prev_d0, sig_z_prev_d0;
wire [31:0] sig_x_prev_d1, sig_y_prev_d1, sig_z_prev_d1;
wire [31:0] x_mask, y_mask, x_shift, y_shift ;
wire [31:0] inv_tan;
wire [63:0] x_mult,  y_mult;
wire [63:0] x_mult1, y_mult1;
wire [31:0] neg_z;

always @(posedge clk)
begin
       if (~rst_n)
              begin
              counter <= 5'h0;
              start_iter <= 1'b0;
              end
       else
              begin
              start_iter <= start;
              if ((start_iter == 1'b1) || (counter != 5'h0))
                     counter <= counter + 1;
              end

end

assign done = (counter == 5'h1F);

assign d = (mode)? (~(y_prev[31])) : z_prev[31]; // d = 0 add, d=1 subtract

assign neg_z = 0'h0 - z;

always @(posedge clk)
begin
       if (~rst_n)
              begin
              x_prev <= 32'h0;
              y_prev <= 32'h0;
              z_prev <= 32'h0;
              end
       else
              if (start == 1'b1 && counter == 5'h0)
                     begin
                     // Check if coarse rotation need to be performed. If yes, do it !
                     if (mode == 1'b0) // Rotation
                            begin
                            if (z[31] == 1'b0 && z > (32'h6487ED51 >>  (5'h1E -
PHASEFRACBASE)))
                                   begin
                                     x_prev <= 0'h0 - y;
                                     y_prev <= x;
                                     z_prev <= z - (32'h6487ED51 >>  (5'h1E -
PHASEFRACBASE));
                                   end
                            else if ((z[31] == 1'b1) && neg_z > (32'h6487ED51 >>
(5'h1E - PHASEFRACBASE)))
                                   begin
                                     x_prev <= y;
                                     y_prev <= 0'h0 - x;
                                     z_prev <= z + (32'h6487ED51 >>  (5'h1E -
PHASEFRACBASE));
                                   end
                            else
                                     begin
```

```
                                        x_prev <= x;
                                        y_prev <= y;
                                        z_prev <= z;
                                        end
                                end
                        else            // Vectoring
                                begin
                                if (x[31] == 1'b1 && y[31] == 1'b0) // x < 0, y > 0
                                        begin
                                        x_prev <= y;
                                        y_prev <= 0'h0 - x;
                                        z_prev <= z + (32'h6487ED51 >>  (5'h1E -
PHASEFRACBASE));
                                        end
                                else if (x[31] == 1'b1 && y[31] == 1'b1) // x < 0, y > 0
                                        begin
                                        x_prev <= 0'h0 - y;
                                        y_prev <= x;
                                        z_prev <= z - (32'h6487ED51 >>  (5'h1E -
PHASEFRACBASE));
                                        end
                                else
                                        begin
                                        x_prev <= x;
                                        y_prev <= y;
                                        z_prev <= z;
                                        end
                                end

                        end
                else if (counter != 5'h0 || start_iter == 1'b1)
                        begin
                        if (d == 0)
                                begin
                                // x_prev <= x_prev - y_shift;
                                // y_prev <= y_prev + x_shift;
                                // z_prev <= z_prev - inv_tan;
                                x_prev  <= sig_x_prev_d0;
                                y_prev  <= sig_y_prev_d0;
                                z_prev  <= sig_z_prev_d0;
                                end
                        else
                                begin
                                // x_prev <= x_prev + y_shift;
                                // y_prev <= y_prev - x_shift;
                                // z_prev <= z_prev + inv_tan;
                                x_prev  <= sig_x_prev_d1;
                                y_prev  <= sig_y_prev_d1;
                                z_prev  <= sig_z_prev_d1;
                                end
                        end
end

// d = 0 adders
adder adder_inst1 (.a_in(x_prev), .b_in(y_shift), .add_nsub(1'b0),
.s_out(sig_x_prev_d0));
adder adder_inst2 (.a_in(y_prev), .b_in(x_shift), .add_nsub(1'b1),
.s_out(sig_y_prev_d0));
adder adder_inst3 (.a_in(z_prev), .b_in(inv_tan), .add_nsub(1'b0),
.s_out(sig_z_prev_d0));

// d = 1 adders
adder adder_inst4 (.a_in(x_prev), .b_in(y_shift), .add_nsub(1'b1),
.s_out(sig_x_prev_d1));
adder adder_inst5 (.a_in(y_prev), .b_in(x_shift), .add_nsub(1'b0),
.s_out(sig_y_prev_d1));
adder adder_inst6 (.a_in(z_prev), .b_in(inv_tan), .add_nsub(1'b1),
.s_out(sig_z_prev_d1));

assign y_shift = (y_prev >> counter) | y_mask ;
assign y_mask  = ~(y_prev[31]) ? 32'h0 : ~(32'hFFFFFFFF >> counter);
```

```
assign x_shift = (x_prev >> counter) | x_mask ;
assign x_mask  = ~(x_prev[31]) ? 32'h0 : ~(32'hFFFFFFFF >> counter);

assign x_mult = (x_prev[31])? ({32'hFFFFFFFF, x_prev} * SCALE_FACTOR) : (x_prev *
SCALE_FACTOR );
assign y_mult = (y_prev[31])? ({32'hFFFFFFFF, y_prev} * SCALE_FACTOR) : (y_prev *
SCALE_FACTOR );

assign x_mult1 = x_mult >> XYFRACBASE;
assign y_mult1 = y_mult >> XYFRACBASE;
assign x_n = x_mult1[31:0];
assign y_n = y_mult1[31:0];
assign z_n = z_prev;


assign inv_tan = (counter == 5'd0) ? INVTAN0 :
                 (counter == 5'd1) ? INVTAN1 :
                 (counter == 5'd2) ? INVTAN2 :
                 (counter == 5'd3) ? INVTAN3 :
                 (counter == 5'd4) ? INVTAN4 :
                 (counter == 5'd5) ? INVTAN5 :
                 (counter == 5'd6) ? INVTAN6 :
                 (counter == 5'd7) ? INVTAN7 :
                 (counter == 5'd8) ? INVTAN8 :
                 (counter == 5'd9) ? INVTAN9 :
                 (counter == 5'd10) ? INVTAN10 :
                 (counter == 5'd11) ? INVTAN11 :
                 (counter == 5'd12) ? INVTAN12 :
                 (counter == 5'd13) ? INVTAN13 :
                 (counter == 5'd14) ? INVTAN14 :
                 (counter == 5'd15) ? INVTAN15 :
                 (counter == 5'd16) ? INVTAN16 :
                 (counter == 5'd17) ? INVTAN17 :
                 (counter == 5'd18) ? INVTAN18 :
                 (counter == 5'd19) ? INVTAN19 :
                 (counter == 5'd20) ? INVTAN20 :
                 (counter == 5'd21) ? INVTAN21 :
                 (counter == 5'd22) ? INVTAN22 :
                 (counter == 5'd23) ? INVTAN23 :
                 (counter == 5'd24) ? INVTAN24 :
                 (counter == 5'd25) ? INVTAN25 :
                 (counter == 5'd26) ? INVTAN26 :
                 (counter == 5'd27) ? INVTAN27 :
                 (counter == 5'd28) ? INVTAN28 :
                 (counter == 5'd29) ? INVTAN29 :
                 (counter == 5'd30) ? INVTAN30 :
                 (counter == 5'd31) ? INVTAN31 : 32'h00000000;

Endmodule
```

Testbench.v: Testbench used for simulations

```
`timescale 1ns/10ps

module test;

reg PCLK;
reg PRESETn;
reg PSEL;
reg PENABLE;
reg [5:0] PADDR;
reg PWRITE;
reg [31:0] PWDATA;
wire [31:0] PRDATA;
wire INT;


// cordic address map
```

```
`define CONTROL        6'h0
`define PROG_A         6'h4
`define PROG_B         6'h8
`define PROG_C         6'hc
`define PROG_D         6'h10
`define OUT1           6'h14
`define OUT2           6'h18
`define OUT3           6'h1c
`define OUT4           6'h20
`define OUT5           6'h24
`define OUT6           6'h28
`define XYFRACBASE      6'h2C
`define PHASEFRACBASE   6'h30
`define CMD_SVD               6'h07
`define CMD_CLEAR       6'h00
`define CMD_SINCOS      6'h01
`define CMD_INVTAN      6'h03
`define RAD_60         32'h10c15238
`define RAD_120               32'h2182a470
`define RAD_N_60       32'hefbeadc8
`define RAD_N_120      32'hde7d5b90
`define RAD_0          32'h00000000
`define RAD_90         32'h1921FB54
`define RAD_45_BASE_20  32'h000C90FE


initial begin
    PCLK=0;
    PRESETn=0;
    APB_write(`CONTROL,`CMD_CLEAR);
    @(posedge PCLK);
    @(posedge PCLK);
    PRESETn=1;

/*
//Test 1 SVD Mode format XYBASE 1.15.16 - a=1 b=2 c=5 d=7 PHASE 3.28
    APB_write(`PROG_A,32'h00010000);
    APB_write(`PROG_B,32'h00020000);
    APB_write(`PROG_C,32'h00050000);
    APB_write(`PROG_D,32'h00070000);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);


//Test 2 SVD Mode format XYBASE 1.11.20 - a=1 b=2 c=5 d=7 PHASE 3.28
    APB_write(`CONTROL,`CMD_CLEAR);
    APB_write(`XYFRACBASE,5'h14);
    APB_write(`PROG_A,32'h00100000);
    APB_write(`PROG_B,32'h00200000);
    APB_write(`PROG_C,32'h00500000);
    APB_write(`PROG_D,32'h00700000);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);

//Test 3 SVD Mode format XYBASE 1.7.24 - a=1 b=2 c=5 d=7 PHASE 3.28
    APB_write(`CONTROL,`CMD_CLEAR);
    APB_write(`XYFRACBASE,5'h18);
    APB_write(`PROG_A,32'h01000000);
    APB_write(`PROG_B,32'h02000000);
```

```
    APB_write(`PROG_C,32'h05000000);
    APB_write(`PROG_D,32'h07000000);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);

//Test 4 SVD Mode format XYBASE 1.19.12 - a=1 b=2 c=5 d=7 PHASE 3.28
    APB_write(`CONTROL,`CMD_CLEAR);
    APB_write(`XYFRACBASE,5'h0c);
    APB_write(`PROG_A,32'h00001000);
    APB_write(`PROG_B,32'h00002000);
    APB_write(`PROG_C,32'h00005000);
    APB_write(`PROG_D,32'h00007000);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);

//Test 5 SVD Mode format XYBASE 1.23.8 - a=1 b=2 c=5 d=7 PHASE 3.28
    APB_write(`CONTROL,`CMD_CLEAR);
    APB_write(`XYFRACBASE,5'h08);
    APB_write(`PROG_A,32'h00000100);
    APB_write(`PROG_B,32'h00000200);
    APB_write(`PROG_C,32'h00000500);
    APB_write(`PROG_D,32'h00000700);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);


//Test 6 SVD Mode format XYBASE 1.27.4 - a=1 b=5 c=2 d=7 (T of earlier matrix) PHASE 3.28
    APB_write(`XYFRACBASE,5'h04);
    APB_write(`PROG_A,32'h00000010);
    APB_write(`PROG_B,32'h00000020);
    APB_write(`PROG_C,32'h00000050);
    APB_write(`PROG_D,32'h00000070);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);

//Test 7 SVD Mode format XYBASE 1.30.0 - a=1 b=2 c=5 d=7
    APB_write(`XYFRACBASE,5'h00);
    APB_write(`PROG_A,32'h00000001);
    APB_write(`PROG_B,32'h00000002);
    APB_write(`PROG_C,32'h00000005);
    APB_write(`PROG_D,32'h00000007);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
```

```
    APB_read(`OUT5);
    APB_read(`OUT6);

//Test 8 SVD Mode format XYBASE 1.15.16 - a=1 b=2 c=5 d=7 (T of earlier matrix) PHASE
2.29
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PROG_A,32'h00010000);
    APB_write(`PROG_B,32'h00050000);
    APB_write(`PROG_C,32'h00020000);
    APB_write(`PROG_D,32'h00070000);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);

//Test 9 SVD Mode format XYBASE 1.15.16 - a=-1 b=-2 c=-5 d=-7 PHASE 2.29
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PROG_A,32'hffff0000);
    APB_write(`PROG_B,32'hfffe0000);
    APB_write(`PROG_C,32'hfffb0000);
    APB_write(`PROG_D,32'hfff90000);
    APB_write(`CONTROL,`CMD_SVD);
    repeat(128) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);
    APB_read(`OUT3);
    APB_read(`OUT4);
    APB_read(`OUT5);
    APB_read(`OUT6);



// Test 10 SINCOS Mode format 1.15.16 - cos/sin(60) PHASE Format 3.28
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,`RAD_60);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_SINCOS);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);


//Test 11 SINCOS  Mode format 1.15.16 - cos/sin(120)  PHASE Format 3.28
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,`RAD_120);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_SINCOS);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);

//Test 12 SINCOS  Mode format 1.15.16 - cos/sin(-60)   PHASE Format 3.28
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,`RAD_N_60);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_SINCOS);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);
```

```
    APB_read(`OUT2);

//Test 13 SINCOS  Mode format 1.15.16 - cos/sin(-120)   PHASE Format 3.28
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,`RAD_N_120);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_SINCOS);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);

//Test 14 SINCOS  Mode format 1.15.16 - cos/sin(0)   PHASE Format 3.28
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,`RAD_0);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_SINCOS);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);


//Test 15 SINCOS  Mode format 1.15.16 - cos/sin(0)   PHASE Format 3.28
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,`RAD_90);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_SINCOS);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);


//Test 16 SINCOS  Mode format 1.15.16 - cos/sin(45)   PHASE Format 11.20
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h14);
    APB_write(`PROG_A,`RAD_45_BASE_20);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_SINCOS);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);
    APB_read(`OUT2);

*/
//Test 17 INVTAN  Mode format 1.15.16 -  atan(1)
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,32'h00010000);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_INVTAN);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);


//Test 18 INVTAN  Mode format 1.15.16 -  atan(32767)
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,32'h7fff0000);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
```

```
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_INVTAN);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);


//Test 19 INVTAN  Mode format 1.15.16 -  atan(-6550)
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,32'he66a0000);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_INVTAN);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);

//Test 20 INVTAN  Mode format 1.15.16 -  atan(0)
    APB_write(`XYFRACBASE,5'h10);
    APB_write(`PHASEFRACBASE,5'h1c);
    APB_write(`PROG_A,32'h00000000);
    APB_write(`PROG_B,32'h00000000);
    APB_write(`PROG_C,32'h00000000);
    APB_write(`PROG_D,32'h00000000);
    APB_write(`CONTROL,`CMD_INVTAN);
    repeat(32) @(posedge PCLK);
    APB_read(`OUT1);

end
always
       #10 PCLK = ~PCLK;
// main inputs
main Imain(     .PCLK(PCLK),
               .PRESETn(PRESETn),
               .PSEL(PSEL),
               .PADDR(PADDR),
               .PENABLE(PENABLE),
               .PWRITE(PWRITE),
               .PWDATA(PWDATA),
               .PRDATA(PRDATA),
               .INT(INT));
//=================================================
// test tasks
//=================================================
task APB_write;
input [5:0] paddr;
input [31:0] pdata;
begin
    PSEL <= 1;
    PADDR <= paddr;
    PWDATA <= pdata;
    PENABLE <= 0;
    PWRITE <= 1;

    @(posedge PCLK);
    PENABLE <= 1;
    @(posedge PCLK);
    PENABLE <= 0;
    PSEL <= 0;
end
endtask

task APB_read;
input [5:0] paddr;
begin
    PSEL <=1;
    PADDR <= paddr;
    PENABLE <= 0;
    PWRITE <= 0;

    @(posedge PCLK);
```

```
    PENABLE <= 1;

    @(posedge PCLK);
    PENABLE <= 0;
    PSEL <= 0;
end
endtask

endmodule
```

9 Appendix B – Matlab Codes

Cordic_svd.m: To compute the SVD by CORDIC arithmetic

```
a = 1;
b = 2;
c = 5;
d = -7;

M = [a b; c d];
disp('SVD of M : ');
[U S V] = svd(M)
disp('Start of CORDIC SVD...');

[j1 j2 theta_sum] = cordic(d-a, c+b, 0, 'vec', 32);
[j1 j2 theta_diff] = cordic(d+a, c-b, 0, 'vec', 32);
theta_r = (theta_sum + theta_diff )/2;
theta_l = (theta_sum - theta_diff )/2;

[x1 y1 j1] = cordic(a,b,theta_r,'rot',32);
[x2 y2 j1] = cordic(c,d,theta_r,'rot',32);

[R11 R21 j1] = cordic(x1, x2, theta_l, 'rot',32);
[R12 R22 j1] = cordic(y1, y2, theta_l, 'rot',32);

cordic_S = [R11 R12; R21 R22]

[c_r s_r j1] = cordic(1, 0, theta_r, 'rot', 32);
[c_l s_l j2] = cordic(1, 0, theta_l, 'rot', 32);
% S = U.' * M * V
cordic_V = [c_r s_r; -1*s_r c_r]
cordic_U = [c_l s_l ;-1*s_l c_l]
recons_cordic_M = cordic_U*cordic_S*cordic_V.'
```

Cordic.m: core cordic engine

```
function [x_n y_n z_n] = cordic(x,y,z,mode,N)
x1 = x; y1 = y;
% Initial Coarse Rotation
if (mode == 'rot')
    if (z >= pi/2)
        z = z - pi/2;
        x1 = -1*y;
        y1 = x;
    else if (z <= -pi/2)
            z = z + pi/2;
            x1 = y;
            y1 = -1* x;
        end
    end
end

if (mode == 'vec')
    if (x <0 && y >0)
        z = z + pi/2;
        x1 = y;
        y1 = -1* x;
    else if (x<0 && y<0)
            z = z - pi/2;
            x1 = -1* y;
            y1 = x;
        end
    end
end


x = x1; y = y1;
```

```
x_n= x;
y_n= y;
z_n= z;

for i=0:N
    x = x_n; y = y_n; z = z_n;
    if (mode == 'vec')
        if (y >= 0) d = -1;
            else d=+1;
        end
    else
        if (z >= 0) d = 1;
            else d= -1;
        end
    end

    x_n = x - y * 2^(-1*i) * d;
    y_n = y + x * 2^(-1*i) * d;
    z_n = z - d* atan(2^(-1*i));

end
x_n = x_n*.6073;
y_n = y_n*.6073;
```

Cordic_error_analysis.m: Error analysis for CORDIC arithmetic

```
clear; close all;
% Error Characterization for CORDIC Arithmetic
c_cos = []; c_sin = [];
for i=-pi:.01:pi
    [x,y,z] = cordic(1,0,i,'rot',32);
    c_cos = [c_cos x];
    c_sin = [c_sin y];
end
% Sine
plot([-pi:0.01:pi], c_sin);
xlabel('Angle (\theta) in radians'); ylabel ('Sine of angle Sin(\theta)');
title('Sin(\theta) computed by CORDIC arithmetic, where \theta varies from [-\pi,
+\pi]');
axis tight;
figure;
err = sin([-pi:0.01:pi]) - c_sin;
plot([-pi:0.01:pi], err.*err);
xlabel('Angle (\theta) in radians'); ylabel ('Square error'); title('Error in calculation
of Sin(\theta) computed by CORDIC arithmetic, \newline               where \theta varies
from [-\pi, +\pi]');
axis tight;
% Cosine
figure;
plot([-pi:0.01:pi], c_cos);
xlabel('Angle (\theta) in radians'); ylabel ('Cosine of angle Cos(\theta)');
title('Cos(\theta) computed by CORDIC arithmetic, where \theta varies from [-\pi,
+\pi]');
axis tight;
figure;
err = cos([-pi:0.01:pi]) - c_cos;
plot([-pi:0.01:pi], err.*err);
xlabel('Angle (\theta) in radians'); ylabel ('Square error'); title('Error in calculation
of Cos(\theta) computed by CORDIC arithmetic, \newline              where \theta varies
from [-\pi, +\pi]');
axis tight;

% Inverse Tangent
c_invtan = [];
for i = [-10:.01:10]
    [x y z] = cordic(1,i,0,'vec',32);
    c_invtan = [c_invtan z];
```

```
end
figure;
plot([-10:.01:10], c_invtan);
xlabel('x'); ylabel ('Tan^{-1}(x)'); title('Tan^{-1}(x)computed by CORDIC arithmetic,
where x varies from [-10, 10]');
err = atan([-10:.01:10]) - c_invtan;
figure;
plot([-10:.01:10], err.*err);
xlabel('x'); ylabel ('Square Error'); title('Error in computation of Tan^{-1}(x)computed
by CORDIC arithmetic, \newline                    where x varies from [-10, 10]');
```

10 Appendix C.1 – Timing report

Worst Case delay = 9940.88 ps
Clock Period = 10000 ps, +slack of 0.02 ps

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
        -sort_by group
Design : main
Version: V-2004.06-SP2
Date   : Mon Dec 11 06:20:16 2006
****************************************
 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: WCCOM   Library: HT018
Wire Load Model Mode: top

  Startpoint: cordic_sysfsm/SCALE_FACTOR_reg[27]
              (rising edge-triggered flip-flop clocked by vclk)
  Endpoint: cordic_sysfsm/cordic_control/x_c2_reg[19]
            (rising edge-triggered flip-flop clocked by vclk)
  Path Group: vclk
  Path Type: max

  Point                                                    Incr       Path
  -----------------------------------------------------------------------
  clock vclk (rise edge)                                   0.00       0.00
  clock network delay (ideal)                              0.00       0.00
  cordic_sysfsm/SCALE_FACTOR_reg[27]/CLK (dff)             0.00 #     0.00 r
  cordic_sysfsm/SCALE_FACTOR_reg[27]/Q (dff)              84.15      84.15 f
  cordic_sysfsm/U2978/Z (inv1)                            23.85     107.99 r
  cordic_sysfsm/U2924/Z (inv4)                            69.56     177.56 f
  cordic_sysfsm/cordic_control/SCALE_FACTOR[27] (control)
                                                           0.00     177.56 f
  cordic_sysfsm/cordic_control/cordic1/SCALE_FACTOR[27] (cordic_1)
                                                           0.00     177.56 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/B[27] (cordic_1_DW02_mult_32_32_1)
                                                           0.00     177.56 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U915/Z (inv1)
                                                          67.65     245.20 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U908/Z (inv1)
                                                         168.80     414.01 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U344/Z (nand2x1)
                                                          60.43     474.43 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7107/Z (xor2x1)
                                                         108.30     582.73 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7106/Z (xor2x1)
                                                         119.30     702.03 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7105/Z (inv4)
                                                          49.56     751.59 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7103/Z (xor2x1)
                                                         107.65     859.23 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7102/Z (xor2x1)
                                                         112.51     971.75 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7101/Z (inv4)
                                                          56.12    1027.86 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7100/Z (xor2x1)
                                                         108.44    1136.30 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7099/Z (xor2x1)
                                                         119.70    1256.00 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7098/Z (inv4)
                                                          53.99    1309.98 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7096/Z (xor2x1)
                                                         108.25    1418.23 r
```

```
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7095/Z (xor2x1)
                                                  112.42    1530.65 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7094/Z (inv4)
                                                   56.38    1587.03 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7092/Z (xor2x1)
                                                  108.48    1695.50 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7091/Z (xor2x1)
                                                  119.69    1815.19 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7090/Z (inv4)
                                                   53.99    1869.17 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7088/Z (xor2x1)
                                                  108.25    1977.42 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U208/Z (xor2x1)
                                                  118.96    2096.38 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7087/Z (xor2x1)
                                                  117.15    2213.53 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7086/Z (xor2x1)
                                                  118.05    2331.58 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7085/Z (inv4)
                                                   53.36    2384.94 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7083/Z (xor2x1)
                                                  108.16    2493.10 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U112/Z (xor2x1)
                                                  118.98    2612.07 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7081/Z (xor2x1)
                                                  116.37    2728.44 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7080/Z (xor2x1)
                                                  119.90    2848.34 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7079/Z (inv4)
                                                   53.36    2901.70 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7077/Z (xor2x1)
                                                  108.16    3009.87 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U218/Z (xor2x1)
                                                  118.98    3128.84 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7075/Z (xor2x1)
                                                  118.49    3247.33 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7074/Z (xor2x1)
                                                  119.90    3367.23 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7073/Z (inv4)
                                                   53.36    3420.59 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7071/Z (xor2x1)
                                                  108.16    3528.75 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U226/Z (xor2x1)
                                                  118.98    3647.73 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7069/Z (xor2x1)
                                                  118.49    3766.22 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7068/Z (xor2x1)
                                                  119.90    3886.12 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7067/Z (inv4)
                                                   53.36    3939.48 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7065/Z (xor2x1)
                                                  108.16    4047.64 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7064/Z (xor2x1)
                                                  133.85    4181.49 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7063/Z (inv4)
                                                   50.82    4232.32 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7061/Z (xor2x1)
                                                  107.95    4340.27 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7060/Z (xor2x1)
                                                  120.73    4461.00 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7059/Z (inv4)
                                                   53.36    4514.36 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7057/Z (xor2x1)
                                                  108.16    4622.52 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7056/Z (xor2x1)
                                                  133.85    4756.37 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7055/Z (inv4)
                                                   50.82    4807.19 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7053/Z (xor2x1)
                                                  107.95    4915.14 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7052/Z (xor2x1)
```

```
                                                          120.73    5035.87 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7051/Z (inv4)
                                                           53.36    5089.23 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7049/Z (xor2x1)
                                                          108.16    5197.39 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U61/Z (xor2x1)
                                                          120.24    5317.63 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7047/Z (xor2x1)
                                                          116.60    5434.23 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7046/Z (xor2x1)
                                                          121.98    5556.22 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7045/Z (inv4)
                                                           53.36    5609.58 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7043/Z (xor2x1)
                                                          107.10    5716.68 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U171/Z (xor2x1)
                                                          121.20    5837.87 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7041/Z (xor2x1)
                                                          116.37    5954.24 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7040/Z (xor2x1)
                                                          122.00    6076.24 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7039/Z (inv4)
                                                           53.36    6129.60 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7037/Z (xor2x1)
                                                          107.10    6236.70 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U199/Z (xor2x1)
                                                          121.19    6357.89 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7035/Z (xor2x1)
                                                          117.08    6474.97 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7034/Z (xor2x1)
                                                          122.04    6597.01 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7033/Z (inv4)
                                                           53.36    6650.38 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7031/Z (xor2x1)
                                                          107.10    6757.48 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U7030/Z (xor2x1)
                                                          135.73    6893.21 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U6801/Z (inv4)
                                                           50.82    6944.03 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U6799/Z (xor2x1)
                                                          106.82    7050.85 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U6798/Z (xor2x1)
                                                          122.04    7172.90 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U6797/Z (inv4)
                                                           53.36    7226.26 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U6265/Z (xor2x1)
                                                          108.16    7334.42 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U6264/Z (xor2x1)
                                                          112.43    7446.85 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5981/Z (inv4)
                                                           56.38    7503.23 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5710/Z (xor2x1)
                                                          107.52    7610.75 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5709/Z (xor2x1)
                                                          122.04    7732.79 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5428/Z (nand2x4)
                                                           80.10    7812.89 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5427/Z (nand2x4)
                                                           37.65    7850.54 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5426/Z (nand2x4)
                                                           57.15    7907.69 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5423/Z (inv4)
                                                           17.29    7924.98 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5421/Z (and2x2)
                                                           44.58    7969.56 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5420/Z (inv4)
                                                           30.86    8000.43 f
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5419/Z (nand2x4)
                                                           33.06    8033.48 r
cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5418/Z (nand2x4)
                                                           54.03    8087.51 f
```

```
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5415/Z (inv4)
                                                    17.91    8105.43 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5414/Z (nand2x4)
                                                    35.91    8141.33 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5413/Z (nand2x4)
                                                    37.40    8178.73 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U5412/Z (nand2x4)
                                                    66.61    8245.35 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/U77/Z (and2x4)
                                                    58.36    8303.70 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/B[31] (cordic_1_DW01_add_62_1)
                                                     0.00    8303.70 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U16/Z (inv1)
                                                    43.04    8346.74 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U300/Z (nor2x4)
                                                    41.45    8388.19 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U298/Z (nand2x4)
                                                    29.50    8417.69 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U297/Z (nand2x4)
                                                    46.38    8464.07 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U296/Z (nand2x4)
                                                    25.76    8489.83 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U295/Z (nand2x4)
                                                    49.47    8539.30 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U287/Z (nand2x4)
                                                    26.54    8565.84 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U286/Z (nand2x4)
                                                    54.08    8619.92 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U279/Z (nand2x4)
                                                    24.71    8644.64 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U278/Z (nand2x4)
                                                    32.80    8677.44 f
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/U277/Z (xor2x1)
                                                   100.31    8777.75 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/FS_1/SUM[43] (cordic_1_DW01_add_62_1)
                                                     0.00    8777.75 r
  cordic_sysfsm/cordic_control/cordic1/mult_338_2/PRODUCT[45]
(cordic_1_DW02_mult_32_32_1)
                                                     0.00    8777.75 r
  cordic_sysfsm/cordic_control/cordic1/U1297/Z (and2x2)
                                                    71.38    8849.13 r
  cordic_sysfsm/cordic_control/cordic1/U1295/Z (or4x2)
                                                   214.75    9063.88 r
  cordic_sysfsm/cordic_control/cordic1/U1233/Z (and2x2)
                                                    65.53    9129.41 r
  cordic_sysfsm/cordic_control/cordic1/U1232/Z (or4x2)
                                                   248.29    9377.70 r
  cordic_sysfsm/cordic_control/cordic1/U1215/Z (and2x2)
                                                    65.63    9443.34 r
  cordic_sysfsm/cordic_control/cordic1/U1214/Z (or4x2)
                                                   241.42    9684.76 r
  cordic_sysfsm/cordic_control/cordic1/y_n[19] (cordic_1)
                                                     0.00    9684.76 r
  cordic_sysfsm/cordic_control/U885/Z (and2x2)       75.14    9759.90 r
  cordic_sysfsm/cordic_control/U881/Z (or5x2)       180.98    9940.88 r
  cordic_sysfsm/cordic_control/x_c2_reg[19]/D (dff)   0.00    9940.88 r
  data arrival time                                           9940.88

  clock vclk (rise edge)                          10000.00   10000.00
  clock network delay (ideal)                         0.00   10000.00
  cordic_sysfsm/cordic_control/x_c2_reg[19]/CLK (dff)     0.00   10000.00 r
  library setup time                                -59.10    9940.90
  data required time                                          9940.90
  ------------------------------------------------------------------------
  data required time                                          9940.90
  data arrival time                                          -9940.88
  ------------------------------------------------------------------------
  slack (MET)                                                    0.02
```

11 Appendix C.2 – Area Report

Total Cell area : 3296853.25 square microns

```
****************************************
Report : area
Design : main
Version: V-2004.06-SP2
Date   : Mon Dec 11 06:24:18 2006
****************************************

Library(s) Used:

    HT018 (File: /home/ecelrc/students/turo/public_html/vlsi1/lab3/synopsys/HT018.db)

Number of ports:               76
Number of nets:               412
Number of cells:                2
Number of references:           2

Combinational area:       3118804.000000
Noncombinational area:    176783.421875
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          3296853.250000
Total area:                 undefined
```

12 Appendix C.3 – Power Report

Dynamic Power = 1.2227 mW (as reported by DC without activity knowledge)

```
design_vision> report_power
Warning: In design 'control', there are 6 submodules connected to power or ground. (LINT-
30)
Warning: In design 'cordic_1', there are 32 ports not connected to any nets. (LINT-30)
Warning: In design 'cordic_1', there are 19 submodules connected to power or ground.
(LINT-30)
Warning: In design 'cordic_1', there are 11 submodules with pins connected to the same
net. (LINT-30)
Warning: In design 'adder_11', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_10', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_9', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_8', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_7', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_6', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_1', there is 1 unconnected input pin.  Logic
0 assumed. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_1', there are 2 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_1', there is 1 submodule connected to power
or ground. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_1', there is 1 submodule with pins connected
to the same net. (LINT-30)
Warning: In design 'cordic_1_DW01_add_62_1', there are 36 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW01_add_62_1', there are 31 feedthroughs. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_63_32_1', there are 64 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_0', there is 1 unconnected input pin.  Logic
0 assumed. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_0', there are 2 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_0', there is 1 submodule connected to power
or ground. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_32_32_0', there is 1 submodule with pins connected
to the same net. (LINT-30)
Warning: In design 'cordic_1_DW01_add_62_0', there are 36 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW01_add_62_0', there are 31 feedthroughs. (LINT-30)
Warning: In design 'cordic_1_DW02_mult_63_32_0', there are 64 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_4', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_1_DW01_add_32_1', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_1_DW01_add_32_0', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_3', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_2', there are 34 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_2', there is 1 feedthrough. (LINT-30)
Warning: In design 'cordic_1_DW01_cmp2_32_1', there are 4 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_1', there are 34 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_1', there is 1 feedthrough. (LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_0', there are 34 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_1_DW01_sub_32_0', there is 1 feedthrough. (LINT-30)
Warning: In design 'cordic_1_DW01_cmp2_32_0', there are 4 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0', there are 32 ports not connected to any nets. (LINT-30)
```

```
Warning: In design 'cordic_0', there are 19 submodules connected to power or ground.
(LINT-30)
Warning: In design 'cordic_0', there are 11 submodules with pins connected to the same
net. (LINT-30)
Warning: In design 'adder_5', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_4', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_3', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_2', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_1', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'adder_0', there is 1 cell that doesn't drive any nets. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_1', there is 1 unconnected input pin.  Logic
0 assumed. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_1', there are 2 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_1', there is 1 submodule connected to power
or ground. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_1', there is 1 submodule with pins connected
to the same net. (LINT-30)
Warning: In design 'cordic_0_DW01_add_62_1', there are 36 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW01_add_62_1', there are 31 feedthroughs. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_63_32_1', there are 64 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_0', there is 1 unconnected input pin.  Logic
0 assumed. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_0', there are 2 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_0', there is 1 submodule connected to power
or ground. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_32_32_0', there is 1 submodule with pins connected
to the same net. (LINT-30)
Warning: In design 'cordic_0_DW01_add_62_0', there are 36 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW01_add_62_0', there are 31 feedthroughs. (LINT-30)
Warning: In design 'cordic_0_DW02_mult_63_32_0', there are 64 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_4', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_0_DW01_add_32_1', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_0_DW01_add_32_0', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_3', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_2', there are 34 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_2', there is 1 feedthrough. (LINT-30)
Warning: In design 'cordic_0_DW01_cmp2_32_1', there are 4 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_1', there are 34 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_1', there is 1 feedthrough. (LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_0', there are 34 ports not connected to any
nets. (LINT-30)
Warning: In design 'cordic_0_DW01_sub_32_0', there is 1 feedthrough. (LINT-30)
Warning: In design 'cordic_0_DW01_cmp2_32_0', there are 4 ports not connected to any
nets. (LINT-30)
Warning: In design 'control_DW01_sub_32_2', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'control_DW01_add_32_2', there are 3 ports not connected to any nets.
(LINT-30)
Warning: In design 'control_DW01_sub_32_1', there are 2 ports not connected to any nets.
(LINT-30)
Warning: In design 'control_DW01_add_32_1', there are 2 ports not connected to any nets.
(LINT-30)
Warning: In design 'control_DW01_add_32_0', there are 2 ports not connected to any nets.
(LINT-30)
Warning: In design 'control_DW01_sub_32_0', there are 2 ports not connected to any nets.
(LINT-30)

Information: Use the 'check_design' command for
```

```
          more information about warnings. (LINT-99)

Information: Updating design information... (UID-85)
Warning: Main library 'HT018' does not specify the following unit required for power:
'Leakage Power'. (PWR-424)
Warning: Design 'main' contains 2 high-fanout nets. A fanout number of 1000 will be used
for delay calculations involving these nets. (TIM-134)
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)

****************************************
Report : power
         -analysis_effort low
Design : main
Version: V-2004.06-SP2
Date   : Mon Dec 11 06:30:42 2006
****************************************


Library(s) Used:

    HT018 (File: /home/ecelrc/students/turo/public_html/vlsi1/lab3/synopsys/HT018.db)

Warning: The library cells used by your design are not characterized for internal power.
(PWR-26)

Operating Conditions: WCCOM   Library: HT018
Wire Load Model Mode: top


Global Operating Voltage = 2.35
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ps
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = Unitless

  Cell Internal Power  =   0.0000 mW    (0%)
  Net Switching Power  =   1.2227 mW  (100%)
                         ---------
Total Dynamic Power    =   1.2227 mW  (100%)

Cell Leakage Power     =   0.0000

1
Information: Defining new variable 'gui_hide_timing_workbench'. (CMD-041)
1
```

13 Appendix D – Layout Result

```
Following are the layout results obtained from CADENCE Silicon
Ensemble:


Aspect Ratio: 1.00  Width: 2800.00 microns,  Height: 2800.00 microns.
Core row utilization = 83.19%.
Chip Area = 7840000.00 sq. microns.
IO to Core Distance (microns):  X: 38.00  Y: 38.00
Number of Standard Cell Rows = 368.
Design is core-limited.
```

Note: We were unable to add a snapshot of the layout since the tool crashed due to “out of memory” issue. The layout was successful and “Verify Connectivity” step passed without any errors.