FPGA BASED DIGITAL INTERFACE PLATFORM FOR CUSTOMIZED ANTENNA ARRAY

A project report submitted by

LT CDR ASHUTOSH KAUSHAL

EE18M010

in partial fulfilment for the award of the degree of

MASTER OF TECHNOLOGY

in

COMMUNICATION & SIGNAL PROCESSING

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

JUN 2020

CERTIFICATE

This is to certify that the thesis entitled **"FPGA Based Digital Interface Platform for Customized Antenna Array"** submitted by **"Lt Cdr Ashutosh Kaushal"** to the Indian Institute of Technology, Madras for the award of **Master of Technology in Communication & Signal Processing** is a **bonafide** record of the project work done by him under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Jun 2020

Dr. S. Christopher
Professor,
Department of Electrical Engineering,
IIT Madras

ABSTRACT

The present project is a critical and important part of a big venture that is to make Modern Day Phased Array Antennas simple and at the same time, self-sufficient. The idea being to co-locate the brain and the body of the antennas to enhance portability, commonality and reliability. In this project, a basic skeleton for development of intelligent antennas was developed through a beam steering algorithm that was coded on an ARTIX-7 FPGA based controller (Digital Controller - DC).

There was a requirement to be able to control each element of Customized Antenna of 16 elements (4 x 4 array) individually for electronic beam steering. Although direct functions for phased array simulation are available off the shelf, none provides a suitable methodology for hardware implementation. Therefore, it became prudent to go back to basics and develop from scratch.

To validate the algorithm, a Graphical user interface platform was developed and transformed to a standalone application using MATLAB to enable user interaction with the system. The DC had the task of suitably mapping one set of input to 16 array elements. The phase and attenuation values calculated by the algorithm were validated by reverse mapping and the RMSE between the Array factors of ideal and practical result was observed to be less than 1dB.

FPGA was chosen over microprocessor as it offered flexibility of designing configurable SoC (system-on-chip) as well as spatial parallelism during execution of commands whereas, on the other hand, microprocessor executes the set of instructions sequentially. VHDL has been used to implement the design on FPGA. The antenna elements had specific least count and data requirement in a typical format. Therefore, the peripheral interface protocol was suitably modified to meet the requirements. Although, the clock frequency on FPGA board was kept to be 25 MHz, the data processing block and interface module were found to be suitable to operate at maximum clock frequency of 682 MHz and 259 MHz respectively allowing a scope of use of clock frequency at least 10 times the present clock. This project is an important step towards the bigger objective discussed earlier. Future work is discussed towards the end of the report.

ACKNOWLEDGEMENTS

I extend my sincere gratitude to Dr. S. Christopher, Professor, Department of EE, IIT Madras and Dr. V. Natarajan, Scientist 'G', Director, Research and Innovation Centre (RIC), DRDO for their kind supervision. I would also like to thank Dr. P. Balasubramanian, Scientist 'F' of RIC, DRDO for guiding me through the difficult territory of VHDL coding and System on Chip design. I am grateful to the institute for giving me all the facilities required for completing the project.

Most importantly, none of this could have happened without the support of my family. I thank my father Late Shri Pradeep Kaushal, my Mother and Sister who constantly supported me through ups and downs.

At last I would like to thank all my friends for their personal and professional support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1.

INTRODUCTION

1.1 **Motivation**. The Indian Navy Platforms comprise of variety of different kinds of equipment and are designed such that every inch on the platform is utilized completely. The expectation to operate in the environment of asymmetric threat has given birth to multirole platforms. There have been instances where the equipment fitted onboard have struggled for adequate spatial clearances from the nearby equipment resulting in mutual or one-sided degradation in performance. Additionally, the increase in the area of operation and changing circumstances requires continuous upgradation. The modern Defence platforms are installed with Phased Array Radars working on the principal of Electronic Beam Steering. The systems available to the Indian Defence forces are state-of-art, however, sometimes they face degradation in performance due to other equipment. One such problem was faced on a serving Indian Navy Destroyer which had a S-band Multifunction Phased Array Radar with Antenna array modules installed on all 04 sides of the main mast to facilitate 360^0 surveillance operation. The Aft (looking towards rear side of the ship) section of the antenna happened to be in the close vicinity of one of the Gas Turbine Exhaust which would emit gases in the temperature range of 400 – 500^0 C. Whenever the GT's feeding that exhaust were run, the tracking performance of the radar in the Aft section was observed to be degraded. Upon contacting the OEM (original equipment manufacturer), it was found that the exposure of the antenna elements to high temperatures result in the degradation of performance, moreover, there was no

compensation algorithm in the system that would facilitate the system to adapt to the ambience and maintain the desired performance. Additionally, it was observed that there was no mechanism to check the health of individual array element. The system would only get indication if at least one-quarter of the array elements of one of the 04 sides will be rendered defective.

This problem has opened up new avenues for discussion and research and thereby is the motivation of our present project. There is a need for development of compensation algorithms which would adapt the system to the temperature gradient experienced by the antenna elements. Also there is a requirement to graduate from the centralized monitoring and control system to a distributed monitoring system that is intelligent enough to locally monitor and adapt to the ambient parameters. The venture is thus divided into two parts viz. development of the basic skeleton and incorporation of intelligence. The extensive work carried out towards our goal is discussed in this report.

1.2 **Background**. Electronic beam steering is the need of the hour. The available antenna simulation platforms from various literatures and surveys generate direct results without displaying the computational flow and do not furnish the back end data for hardware implementation. The approach briefed in this report provides the user with analyzable details that facilitate building of an antenna array beam steering system from scratch.

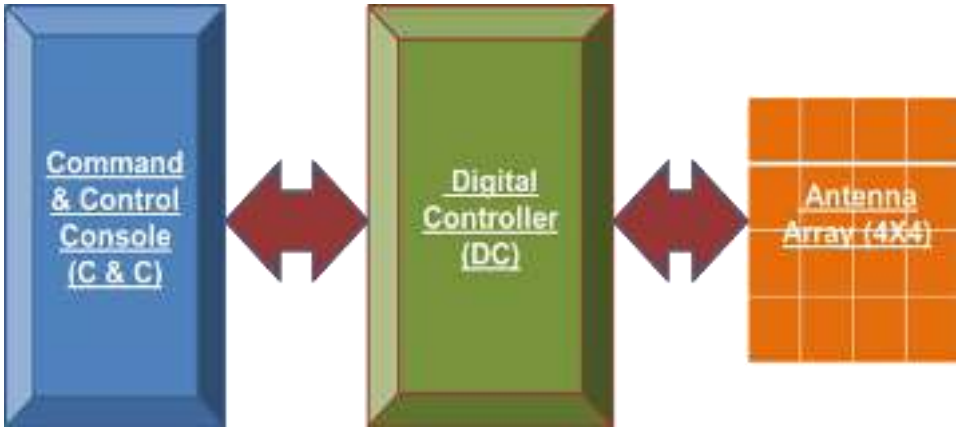


Fig 1.1. System Block Diagram

The active phased array antenna system under consideration is a customized array with specific data requirements for beam steering. The system consists of majorly three modules viz. Command and Control console (C&C), a Digital Controller (FPGA based), and the Phased Array antenna as shown in figure 1.1

The DC is the brain of the antenna and houses all processing/ compensation algorithms. It also controls all signal/data lines. All desired functions of the antenna are digitally controlled via customized serial peripheral interface between DC and Antenna. All computations are performed by the DC and only relevant data/control signals are sent to the antenna elements. This increases standardization and paves way for minimizing complexity of the command and control console and antenna array.

1.3 **The available FPGA platform**. The ARTIX-7 (XC7A200T-1FBG676IFPGA) FPGA based platform is being used as the Digital Controller (DC).

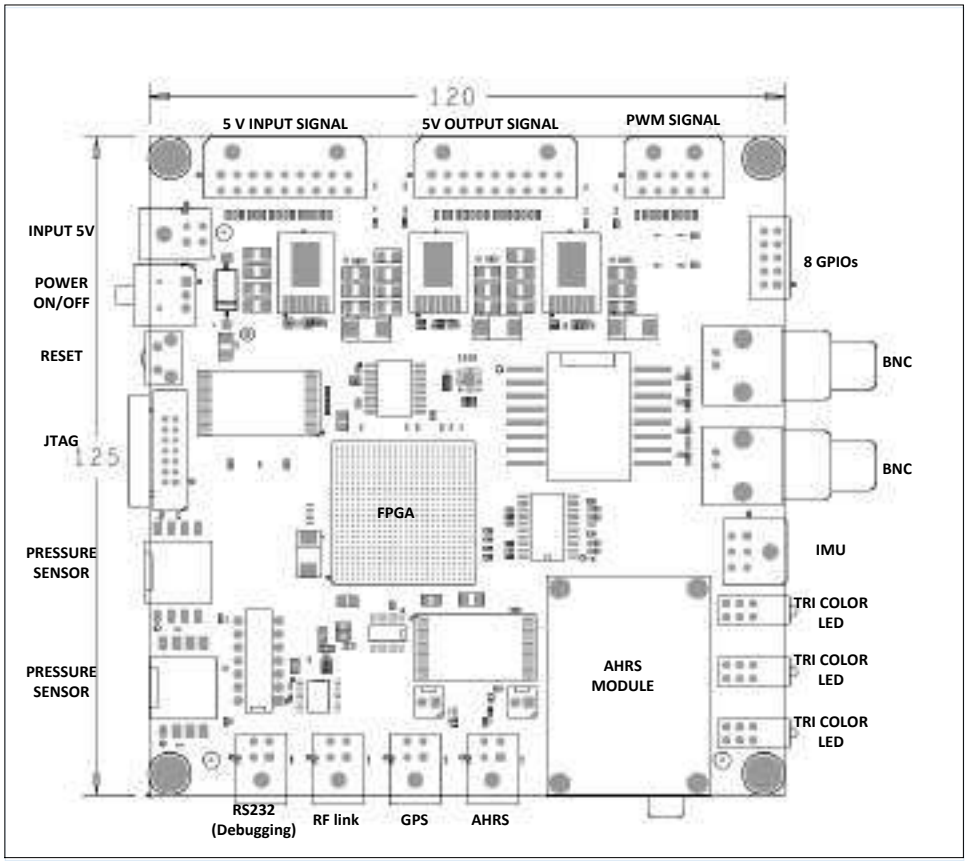


Fig 1.2. Layout of ARTIX-7 FPGA board

This contains the beam steering algorithms and acts as an interface between the C&C (command and control console) and antenna panel and furnishes requisite data to the ICs (chips) of antenna panel based on the user requirements as well as the feedback from the antenna array. The salient features of the board that are relevant for present design purpose are as follows: -

(a) 15 output signals, 15 Input signals & 8 GPIO's.

(b) Inbuilt clock of 50 MHz

(c) 64 Mb parallel flash memory.

(d) 256 Mb of SPI flash memory.

(e) Interfaces – RS 232, JTAG

1.4 **The Customized Antenna Array Data Requirement**. Customized antenna Array consists of 'N^2' array elements (N rows * N columns square array). Each array element of the antenna array comprises of 05 SIPOs (Serial in parallel out) of 08 Bits each and therefore receives the 40-bit data from DC through 05 parallel SPI interfaces. All signal lines are depicted in figure 1.3.

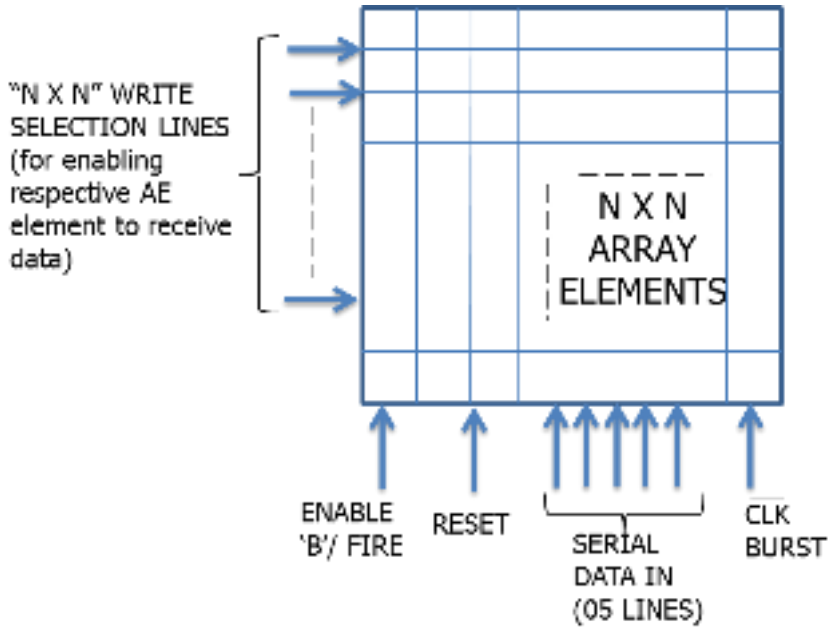


Fig 1.3 Antenna Array – All Signal lines

The data distribution requirement for each element of the antenna array is shown in Table 1.1 below.

Table – 1.1 Data distribution for each antenna array element

SIPO5_Data (39-32)	SIPO4_Data (31-24)	SIPO3_Data (23-16)	SIPO2_Data (15-8)	SIPO1_Data (07-00)
Phase	Attenuation Level (TX)	Attenuation Level (RX)	Attenuation Level (BITE)	TX/RX/BITE Switch select

1.5 **Digital Controller** The figure 1.4 explains the digital controller (DC) hardware implementation. The DC comprises of FPGA for computing the algorithms for processing and formatting the data received via RS-232 interface for further transmission to antenna array. Although, the implementation of algorithm on FPGA platform via Hardware Descriptive Language was pretty challenging, but it gave us the flexibility of designing task specific combinational structure that facilitated parallel placement and operation of various modules.

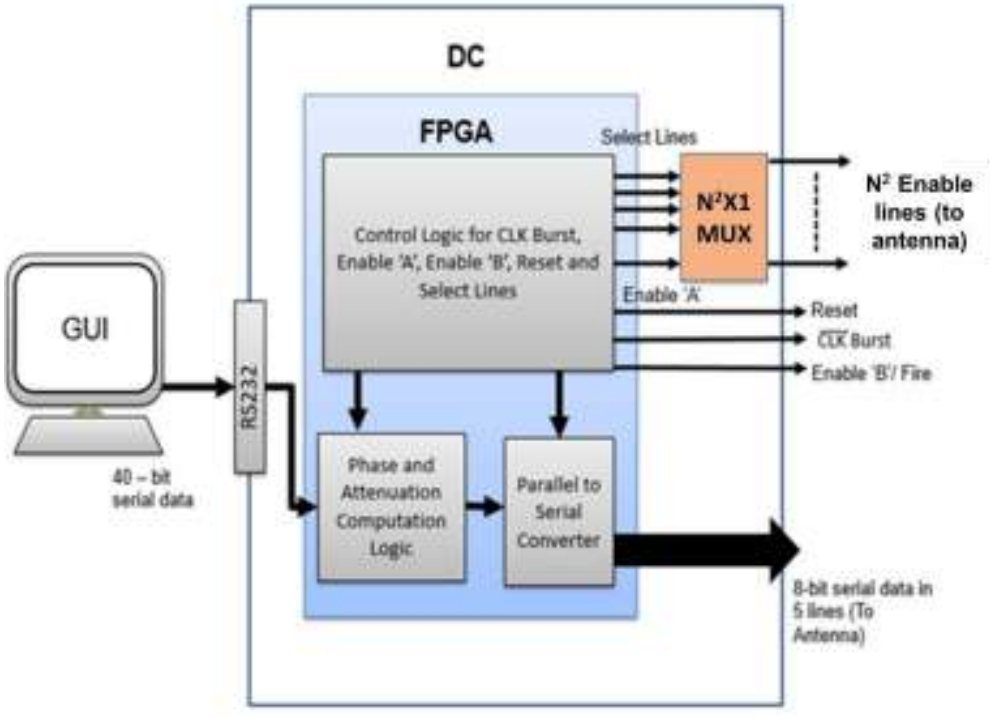


Fig 1.4 Digital Controller Hardware Implementation

The data received from GUI through RS232 is formatted and fed as input to the algorithm. The algorithm implemented in the FPGA computes the phase angle, and attenuation level for all N^2 elements of the array and formats the data for onward transmission to antenna module. The Enable 'B' and Reset are hardwired signal interrupts generated by the FPGA as per the user requirements.

1.6 **Problem Statement**.With the requirement of the antenna array clear to us, we are required to formulate the problem statement. The problem can be broken down into 07 parts as below: -

- Development of a Graphical User Interface at the Command and Control PC to enable the user enter desired beam look angles and various other parameters.
- Enabling the GUI to work on all platforms with serial port by converting the code to a standalone application.
- Validation of the algorithm at C&C.
- Establishing RS-232 link between FPGA and C&C.
- The C&C only transmits fixed point representations of the desired beam angle and Attenuation level. The remaining computation has to take place on the Digital controller.
- Processing the raw data received from C&C on the FPGA and calculate relevant data set for each and every antenna element so that the antenna beam looks in the desired direction. The phase state of the antenna element has a resolution of 1.40625 degrees (256 states for range 0 - 360 degrees) and the attenuation state has a resolution of 0.25dB (256 states for range 0 – 64dB).
- Establishing a robust communication link between DC and Antenna Array.

1.7 **Organization of thesis**

The next chapter consist of the background study about planar array antennas and fixed point mathematical operations. Chapter 3 discusses the development of a Graphical User Interface, conversion of code to exe, development and validation of the algorithm using MATLAB, the block level architecture of FPGA and the Data Processing Block. Chapter 4 discusses the implementation of the interface algorithm on the FPGA board. Chapter 5 consist of results and discussions. Chapter 6 concludes the work and discusses the future scope.

CHAPTER 2.

LITERATURE SURVEY

2.1 **Planar Antenna Array**. If the individual radiating antenna elements are positioned along a rectangular grid, the arrangement is known as a planar antenna array. The planar arrays are versatile and can provide more symmetrical patterns with lower side lobes. The array under consideration is a uniformly spaced customized planar array whose beam is desired to be steered according to the input values entered by the user. An example of a 2D planar array is shown in Figure 2.1. The elements are arranged uniformly along a rectangular grid in the *XY*-plane, with an element spacing *dx* in the x-direction and an element spacing *dy* in the y-direction. The elements depicted in red are the individual radiating elements.

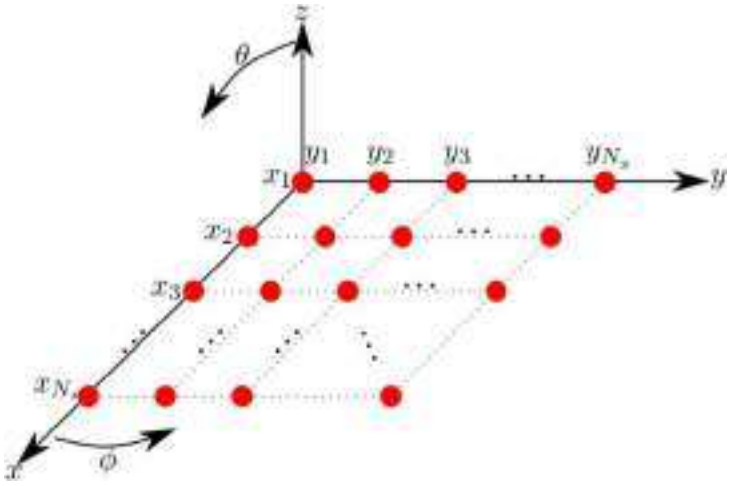


Figure 2.1 Example of Planar Antenna Array

2.2 **Array Factor**. The Array factor for the entire planar array with M x N elements may be expressed as below

$$\text{AF} = \sum_{n=1}^{N} In * \left[\sum_{m=1}^{M} Im * e^{j((m-1)(kd\sin\theta\cos\varphi + \beta x)}\right] * e^{j((n-1)(kd\sin\theta\sin\varphi + \beta y)} \ldots (1)$$

Where $\boldsymbol{I}$ m and $\boldsymbol{I}$ n are the excitation coefficient of each element along x and y axis respectively. If we assume *M* elements placed along the x-axis, the spacing and the progressive phase shifts between the elements along the x-axis are represented by d_x and β_x respectively. If *N* such arrays are placed next to each other in the y-direction at a distance d_y apart and with a progressive phase shift β_y, a rectangular array will be formed as shown in figure 2.1.

When the spacing between the elements is equal or greater than λ, multiple maxima of equal magnitude are formed.to avoid grating lobes in the x-z and y-z plane s, the spacing between the elements in the x and y directions respectively must be less than λ. For a rectangular array, the major lobe and grating lobes are located at

$$kdx\sin\theta\cos\emptyset + \beta x = \pm 2m\pi \quad ; m = 0,1,2\ldots.$$

$$kdy\sin\theta\sin\emptyset + \beta y = \pm 2n\pi \quad ; n = 0,1,2\ldots.$$

The phases β_x and β_y are independent of each other. In most practical applications it is required that the conical main beams of linear arrays along x and y directions intersect at their maxima and be directed along the same direction. If it is desired to have only one main beam that is directed along $\theta = \theta_0$ and $\varphi = \varphi_0$, the progressive phase shift between the elements in the x and y directions is given as under

$$\boldsymbol{\beta x = -kdx\ sin\ \theta 0\ cos\ \varphi 0} \ldots\ldots\ldots\ldots (\mathbf{2})$$

$$\boldsymbol{\beta y = -kdy\ sin\ \theta 0\ sin\ \varphi 0} \ldots\ldots\ldots\ldots (\mathbf{3})$$

using equations (4) and (5) below we get 'N X M' (here N X M = 16) values of phase for given value of Θ_0 and Φ_0. The application of the progressive phase calculation formula is shown in table below.

$$\boldsymbol{\beta x = -kdx \sin \theta 0 \cos \emptyset 0 = X\ (say)} \ldots\ldots (4)$$

$$\boldsymbol{\beta y = -kdx \sin \theta 0 \sin \emptyset 0 = Y\ (say)} \ldots\ldots\ldots (5)$$

Ser	***Element Address***	***Phase Calculation***
1	(0, 0)	Phase = 0 (assume)
2	(0, -1)	Phase (0, -1) = -X
3	(0, 2)	Phase (0, 2) = 2X
4	(1, 2)	Phase (1, 2) = Y + 2X
Similarly, for any array element		
$\boldsymbol{Phase\ (i,j) = i * Y + j * X}$……(6)		

Table 2.1: - Phase Calculation for each Array Element

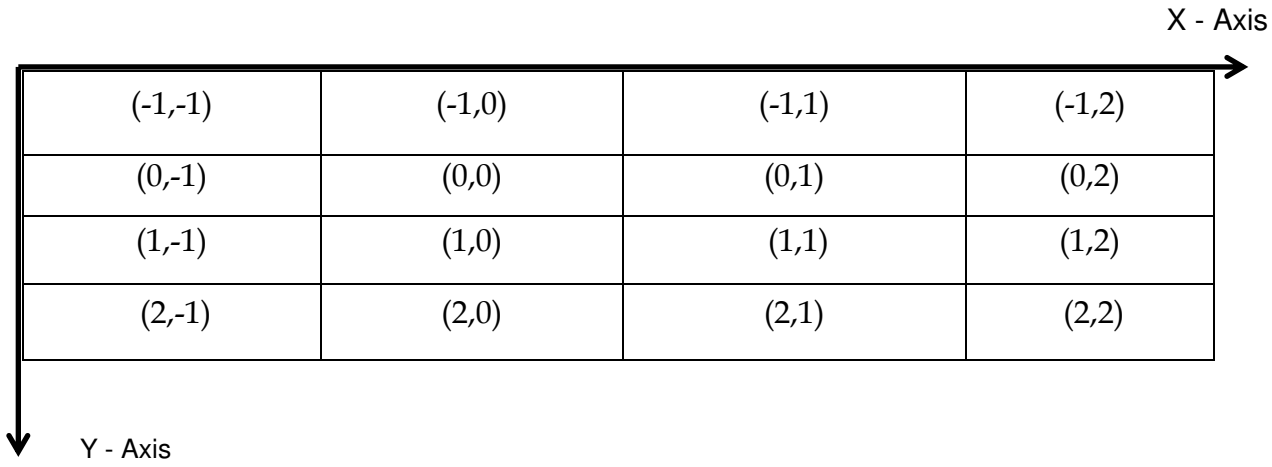


Fig 2.2: Antenna Array (Top view)

2.3 **Fixed Point Number Representation and Arithmetic**. Fixed point is a simple yet very powerful way to represent fractional numbers. By reusing all integer arithmetic circuits of a computer, fixed point arithmetic is faster than floating point arithmetic. This is the reason why it is being used in many game and DSP applications. On the other hand, it lacks the range and precision that floating point number representation offers. A circuit designer must do the tradeoff. In real life, we deal with

real numbers with fractional part. Most modern computers possess native (hardware) support for *floating point* numbers (IEEE-754 standard). However, the use of floating point is not necessarily the only way to represent fractional numbers. This article describes the fixed point representation of real numbers. The fixed point data type is used where performance is sometimes more important than precision. As we will observe, fixed point arithmetic is much faster than floating point arithmetic.

A binary number 110101_2 represents the value:

$1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0* 2^1 + 1 * 2^0$
$= 32 + 16 + 4 + 1$
$= 53_{10}$

Now, if we divide the number 53 by 2, we know the result should be 26.5. However, how do we represent it if we only had integer representations?

The key to represent fractional numbers, like 26.5 above, is the concept of *binary point*. A binary point is like the decimal point in a decimal system. It acts as a divider between the integer and the fractional part of a number. In a decimal system, a decimal point denotes the position in a numeral that the coefficient should multiply by $10^0 = 1$. For example, in the numeral 26.5, the coefficient 6 has a weight of $10^0 = 1$. But what happen to the 5 to the right of decimal point? We know from our experience, that it carries a weight of 10^{-1}. We know the numeral "26.5" represents the value "twenty-six and a half" because

$2 * 10^1 + 6 * 10^0 + 5 * 10^{-1} = 26.5$

The very same concept of decimal point can be applied to our binary representation, making a "binary point". As in the decimal system, a binary point represents the coefficient of the term $2^0 = 1$. All digits (or bits) to the left of the binary point carries a weight of 2^0, 2^1, 2^2, and so on. Digits (or bits) on the right of binary point carries a weight of 2^{-1}, 2^{-2}, 2^{-3}, and so on. For example, the number:

11010.1_2

represents the value = $1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0* 2^0 + 1 * 2^{-1}$

$= 16 + 8 + 2 + 0.5$

$= 26.5$

(a) **The Shifting Property**. If we observe carefully, we realize the bit pattern of 53 and 26.5 is exactly **the same**. The *only* difference, is the position of binary point. In the case of 53_{10}, there is "no" binary point. Alternatively, we can say the binary point is located at the far right, at position 0. (Think in decimal, 53 and 53.0 represents the same number). In the case of 26.5_{10}, binary point is located one position to the *left* of 53_{10}. In general, **given a fixed binary point position, shifting the bit pattern of a number to the right by 1 bit *always* divide the number by 2. Similarly, shifting a number to the left by 1 bit multiplies the number by 2.**

(b) **The fixed point representation**. The shifting process above is the key to understand fixed point number representation. To represent a real number, we can define a fixed point number type simply by implicitly *fixing* the binary point to be at some position of a numeral. We will then simply adhere to this implicit convention when we represent numbers.

To define a fixed point type conceptually, all we need are two parameters:

- width of the number representation, and
- binary point position within the number

We will use the notation "***QM.N***" for the rest of the discussion, where Q denotes the sign bit, *M* denotes the number of bits used before the decimal (integer word length) and *N* denotes the fractional word length of the number. For example, Q4.3

represents an 8-bit fixed point number, of which 3 right most bits are fractional. Therefore, the bit pattern represents a real number:

00100.110_2

$= 1 * 2^2 + 1 * 2^{-1} + 1 * 2^{-1}$

$= 4 + 0.5 + 0.25 = 4.75$

(c) **Advantages and Disadvantages of fixed-point representation**. The benefit of fixed point arithmetic is that they are as straight-forward and efficient as integers arithmetic in computers. We can **reuse all the hardware built to for integer arithmetic** to perform real numbers arithmetic using fixed point representation. The disadvantage of fixed point number is the loss of range and precision when compare with floating point number representations.

(d) **Unsigned by Signed Multiplication**

Assume that a=01.001 and b=10.010 are two numbers in Q2.3 format. Assume that a is an unsigned number but b is signed. Find the product of a × b

1						0	1	0	0	1		9
2		×				1	0	0	1	0		-14
3						0	0	0	0	0		
4					0	1	0	0	1			
5				0	0	0	0	0				
6			0	0	0	0	0					
7		+	1	1	0	1	1	1				
8			1	1	1	0	0	0	0	1	0	-126

The multiplication is done by considering the multiplier as an unsigned number except we must interpret the sign bit with a negative weight. Considering the position of the binary point, we obtain a × b = 1110.0000102. The result is obviously a signed number because our last partial product was signed. To find the equivalent decimal value of this negative number, we can write a × b = −(0001.1111102) =−1.9687510. This is consistent with the decimal calculations, i.e. −1.96875=−126/64.

(e) **Signed by Signed Multiplication**

Assume that a=11.0012 and b=10.0102 are two signed numbers in Q2.3 format. Find the product of a × b.

1							1	1	0	0	1	-7
2		×					1	0	0	1	0	-14
3			0	0	0	0	0	0	0	0	0	0	
4			1	1	1	1	1	1	0	0	1		
5			0	0	0	0	0	0	0	0			
6			0	0	0	0	0	0	0				
7		+	0	0	0	1	1	1					
8			0	0	0	1	1	0	0	0	1	0	98

Similar to the unsigned-by-signed multiplication, we have to consider a negative weight for the MSB of b. Hence, as shown in the above calculations, the last partial product is obtained by calculating the two's complement of a. When calculating this two's complement, we must represent the multiplicand, a, with one bit longer and, then, find its two's complement. Since here a is a signed number, representing it with one bit longer is equal to sign-extending it by one bit.

(f) **Fixed point division**. The division should be avoided in hardware if not strictly necessary. There is a simple trick that can be used if you need to divide a number by a constant. The trick is to use a multiplication instead of a division. if you need to perform

A/B you have to simply implement A * (1/B).

Example: -

435/17 = 25

we get the same result multiplying and dividing by, for example, 32 (2^5)

435 x (32/17) / 32 =

435 x (1.882) / 32 =

818.823 / 32 = 25.588

The division by 32 is simply implemented by right shift of 5 positions. In this case there is no need to perform division, we need to perform only a multiplication and right shift by a constant number of bits.

2.4 **Booth's Algorithm for Signed bit multiplication**. Let 'M' be the multiplicand and 'Q' be the multiplier, A be the accumulator of 'n' bits (higher size among M and Q) and q0 is an extra initialization bit and q1 is the LSB of Q. The booth multiplication flowchart is depicted in figure 2.4. Booth algorithm gives a procedure for **multiplying binary integers** in signed 2's complement representation **in efficient way**, i.e., less number of additions/subtractions required. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.

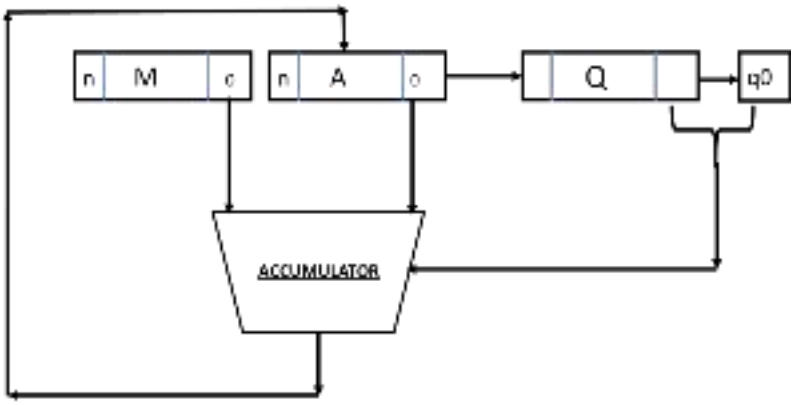


Fig 2.3 Hardware Structure Implementing Booth's Algorithm

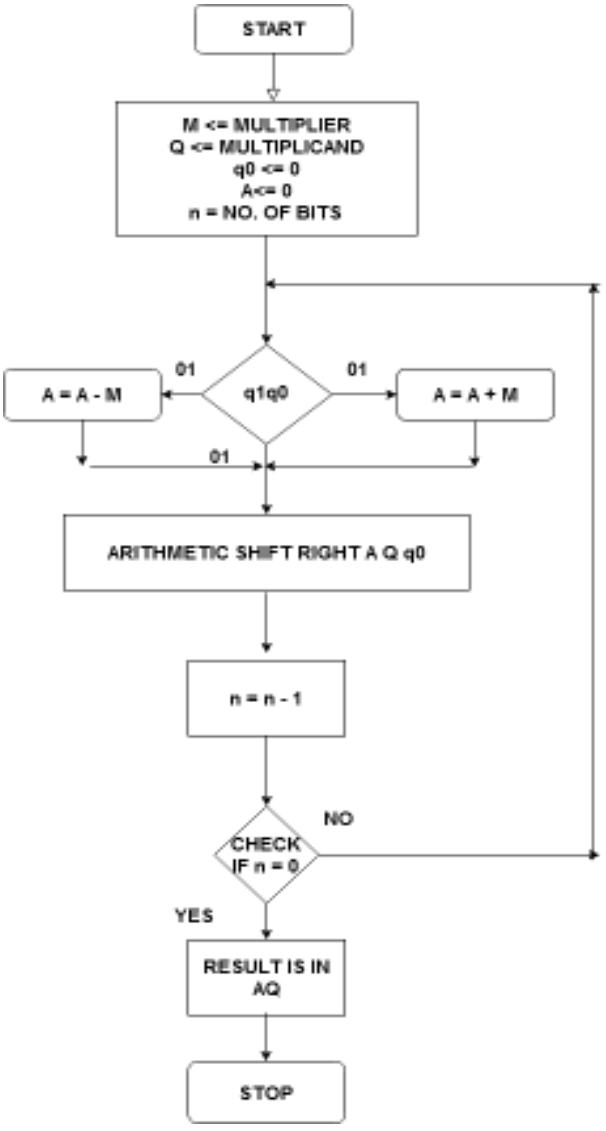


Fig 2.4. Flowchart of Booth's Algorithm

2.5 **Look- Up Table Based Sine-Cosine calculations**. The suggested implementation method uses the inbuilt Random Access Memory (RAM) within the FPGA board to store 256 values of the fixed point representation of quarter period of the sine/cosine signal. For a particular value of angle entered by the user, the search operation similar to binary search is required which starts comparing the bits from the MSB and therefore eliminates search area by half in every clock. For example, if the MSB of the angle is 0 then it lies in the memory segment that contains the positive half of the sine wave i.e. [0, π]. Similarly, by comparing the next highest bit, the quadrant is ascertained and so on till value of the nearest address is mapped to the output.

Samples range	Quarter location
1-64	first quarter
65-128	second quarter
129-192	third quarter
193-256	fourth quarter

2.6 **Serial Peripheral Interface**. SPI is a synchronous, full duplex master-slave-based interface. The data from the master or the slave is synchronized on the rising or falling clock edge. Both master and slave can transmit data at the same time. A 3 - wire SPI device has three signals:

- Clock (SPI CLK, SCLK)
- Chip select (CS)
- Master out, slave in

The device that generates the clock signal is called the master. Data transmitted between the master and the slave is synchronized to the clock generated by the master. SPI devices support much higher clock frequencies compared to I2C interfaces. Users should consult the product data sheet for the clock frequency specification of the SPI interface. The chip select signal from the master is used to select the slave. This is normally an active low signal and is pulled high to disconnect the slave from the SPI bus. When multiple slaves

are used, an individual chip select signal for each slave is required from the master. In chapter 4, we have used a customized version of SPI to communicate between FPGA and Antenna Array. The interface diagram of FPGA and Antenna array is shown in fig 2.5.

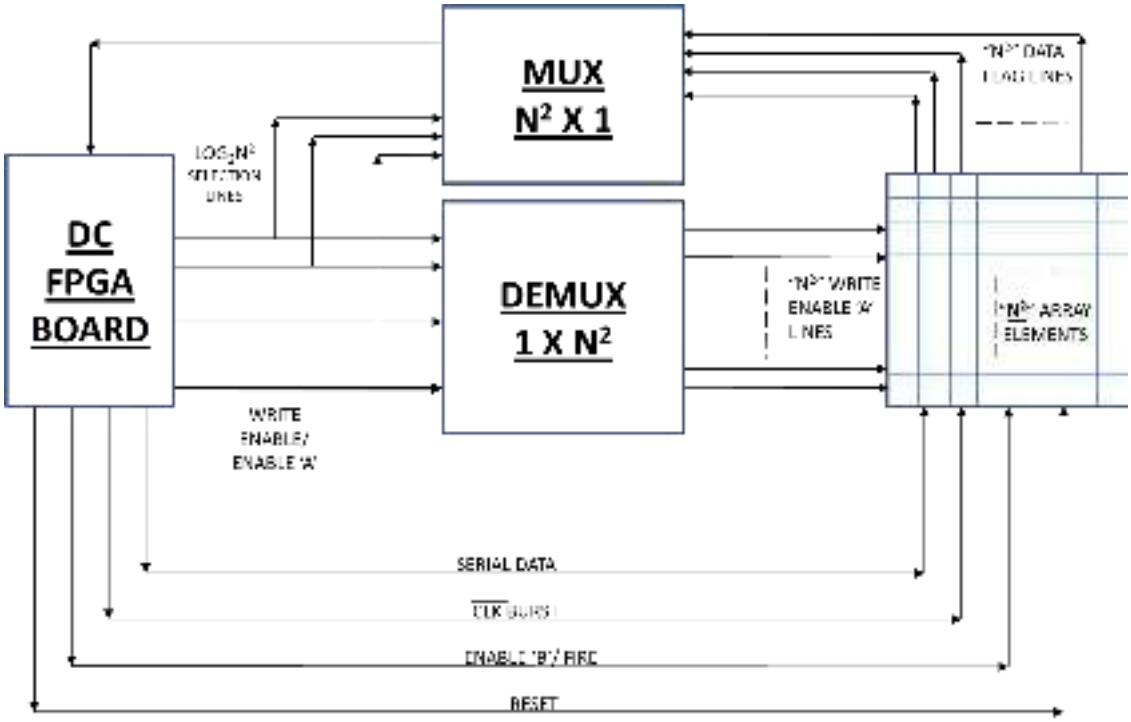


Fig 2.5. DC – Antenna array interface diagram

CHAPTER 3.

WORK DONE – I

3.1 **Graphical User Interface**. An application based Graphical User Interface is developed using MATLAB. This interface facilitates user to enter the desired beam direction (in terms of azimuth & elevation angle) and attenuation level. It also provides a self-test facility which ensures code validation on application boot up. The GUI receives the desired parameters from the user and sends them to the serial port for further communication with the FPGA board. The algorithm (later implemented on the FPGA) calculates and furnishes the progressive phase values corresponding to all antenna elements of the array for the beam to look in a particular direction. The values obtained, after suitable conversion and mapping can now be directly transmitted to antenna array thereby minimizing the hardware and processing requirement at the antenna end. The Command and control console comprises of a graphical user interface (figure 3.1) that enables user to enter the size of array antenna, desired azimuth, elevation, attenuation level and TX/RX/BITE mode. The data is entered in unsigned integer format and is converted to the binary equivalent as per table-3.1 and is transmitted over the RS232 channel to DC for further processing. Co-incidentally the data packet from C&C to Digital controller has been also selected to be of 40 bits – same as the antenna array data requirement, however, the difference is that for one set of input data entered by the user, C&C generates only one 40-bit packet while on the other hand, the DC undertakes further processing and maps the input data to generate N^2 sets (equal to the number of antenna array elements) of 40-bit data. For speedy communication and to reduce time lag, we

have chosen 05 parallel data communication channels between DC and Antenna array. The commands Reset, Fire and Feed user data are signals used for similar functions respectively. The user fed data is converted to fixed point binary equivalents and further communicated to DC through a RS232 interface. The GUI is further converted to a .exe file using the inbuilt MATLAB application compiler. This enables us to use the interface as a standalone application. The angle processing algorithm discussed in chapter 2 maps the combined details of azimuth and elevation to one of the 256 available state values for each antenna element.

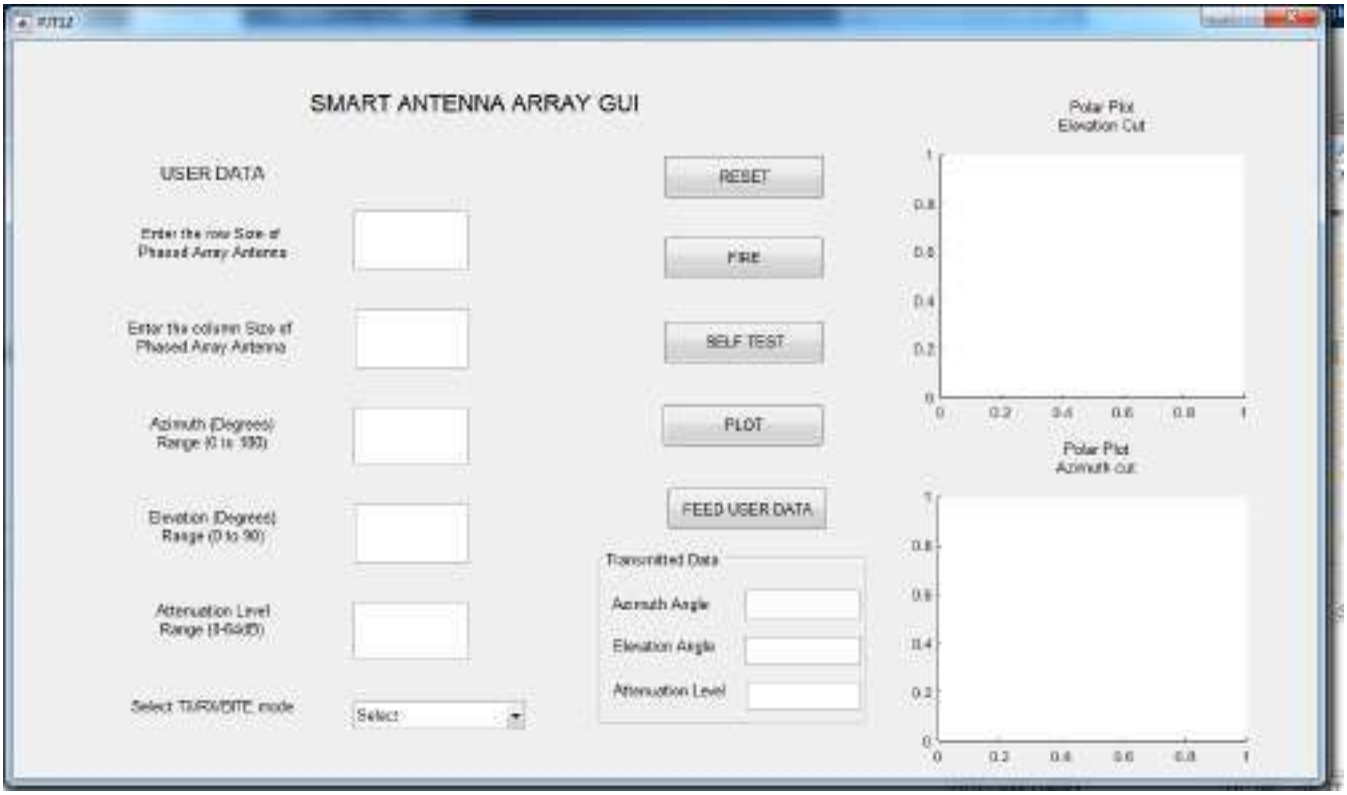


Fig 3.1 The Graphical User Interface

Table – 3.1 Bit Assignment for Input Parameters

Parameter	*No. of Bits*
Azimuth Angle	13
Elevation Angle	13
Attenuation	10
Mode Selection	02
Reset	01
Fire	01

3.2 **Algorithm Development**. The step by step explanation of the adopted methodology is mentioned below: -

(a) **Command & Control Console**

- Enter the value of looktheta, lookphi, attenuation, row, column.
- Calculate βx, βy, taper values.
- Plot AF_Simulated.
- Calculate relative phase values for each array element (radians)
- Convert to degrees by multiplying with 180/π.
- Divide each value obtained in degrees by 1.40625 to get the nearest phase state.
- Convert the result obtained to binary. This gives the 08-bit state value for each array element.
- For validation of the algorithm follow the reverse chronology.
- From the phase state values calculate βx and βy (by mutual subtraction of any two values on the same axis.
- Apply the obtained βx, βy to the AF formula in chapter 2.
- Both the AF_Verified.
- There will be some error in the previous and latest AF plot due to multiple mapping to nearest states.
- This error should be minimum

(b) **Digital controller**

- Receive the 40-bit data corresponding to the desired look angle, attenuation and mode of operation from C&C via RS-232 interface.
- Calculate sine θ, cos φ and sine φ via method of look-up- tables.
- Calculate the value of βx & βy using booth's multiplier and other fixed point mathematical operations.
- Calculate Phase states and Amplitude excitation states.

- Transmit the data to antenna array using customized peripheral interface

3.3 **FPGA Block Level Architecture**. The FPGA chain algorithm touched upon in chapter 1 is expanded in this chapter for a customized 4 x 4 antenna array. The 40-bit data is received via a RS-232 interface between the command and control console & FPGA board and is ready for further processing. The RS-232 interface architecture used in this project is an off-the-shelf available architecture which has been utilized for our purpose at a baud rate of 9600 bps. The architecture is broadly divided into three blocks.

- RS-232 data receiver
- Data Processing Block
- Customized Interface Block

The same is depicted in fig 3.2

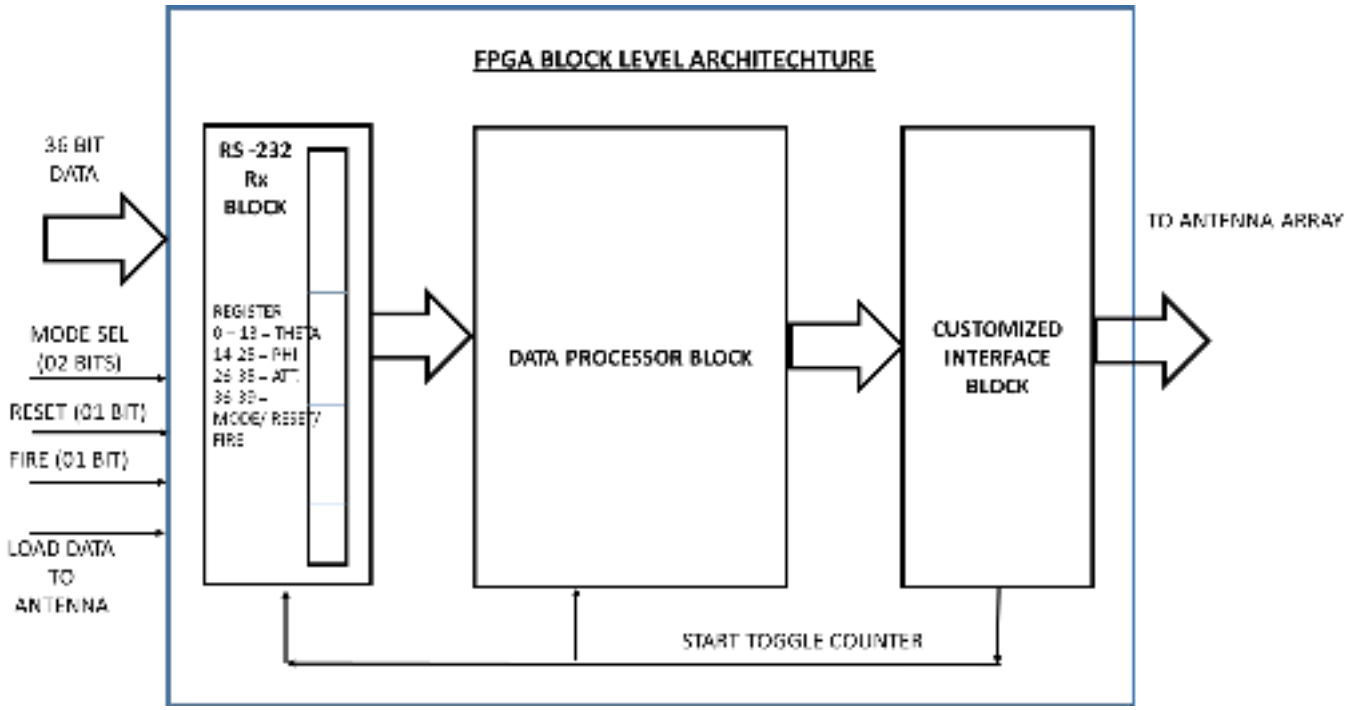


Fig 3.2 FPGA Block Level Architecture

The 40-bit data transmitted from C&C is stored in the register and is transmitted for further processing once the previous data has been fed into the registers of antenna array. The data processor block calculates the respective attenuation and phase states which are multiplexed into the temporary register of customized interface depending upon the state of the start toggle counter. The same is further transmitted to the antenna array in the required format.

3.4 **Data Processing Block**. This block is the heart of the FPGA architecture. This processes the input values and generates a set of 16 Taper states as well as 16 Phase states for the 16 elements of the antenna array. The following steps are followed

- Calculation of Sine (θ), Sine (φ) and Cos (φ) in radians by using look-up-table method
- Calculation of βx and βy

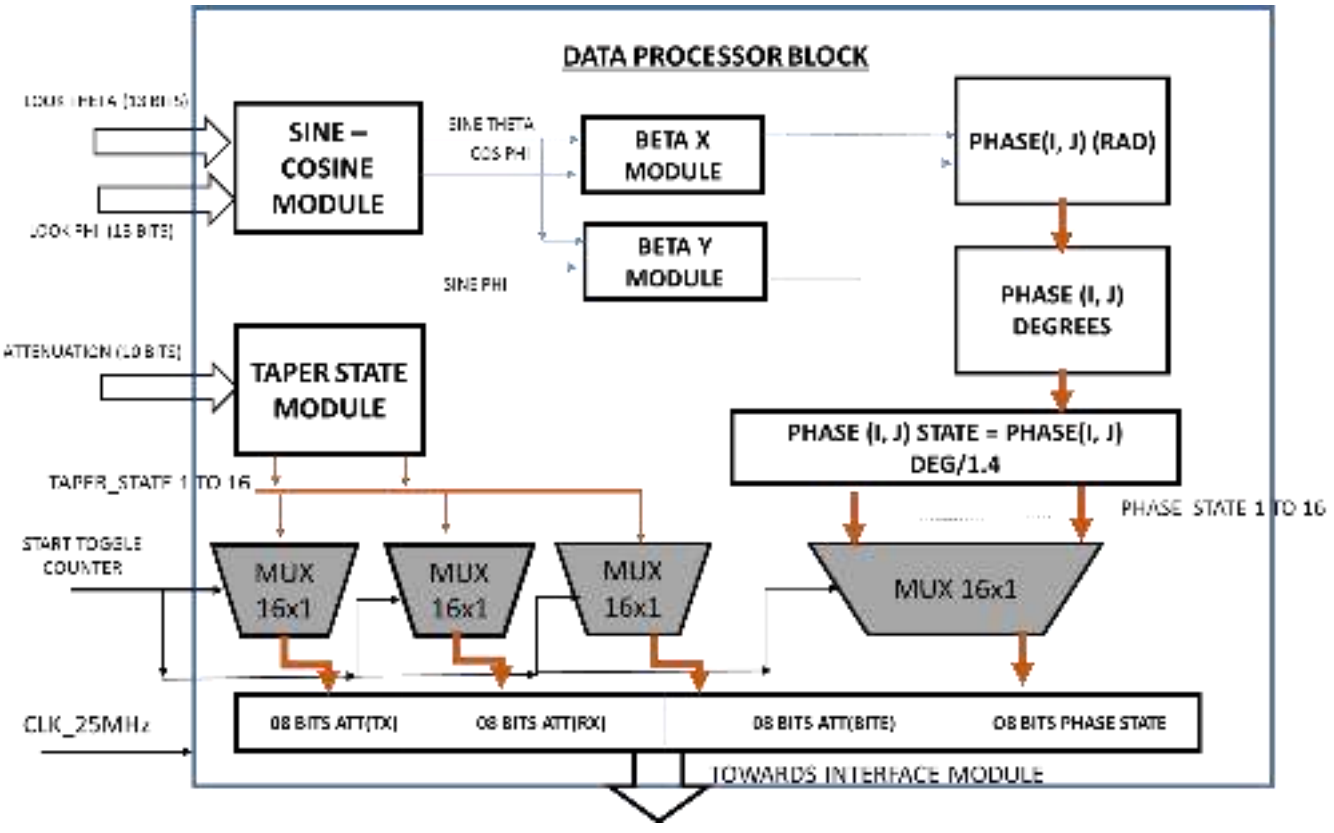


Fig 3.3 Data Processing Block – Top Level Architecture

- Calculation of Taper state for each element (using Taylor distribution)
- Calculation of Phase (I, J) values in radians as per equation (6) of chapter 2.
- Conversion of Phase (I, J) values to degrees by multiplication by 180/π.

Division of Phase value in degrees by 1.4 (angle resolution of the antenna array element) to obtain the phase stateThe top level architecture of Data processing module is shown in fig 3.3. It is to be noted that the data processor block gives a 32-bit output for interface module. Along with this 32-bit data, the 04 bits of mode selection, reset and fire are attached for further transmission to antenna array. The start-toggle-counter is the counter that changes state when the interface module completes data transmission to one array element.

(a) **Sine – Cosine Module**. The looktheta and lookphi data is received in the fixed point form (13 bits) from the C&C PC out of which 8 bits are assigned to the integer part and the remaining 05 bits are assigned to the fractional part. The sine-cosine module comprises of sine and cosine look up tables. The search and map algorithm makes use of the modified binary search algorithm and starts comparison from the MSB of the temp_reg. The algorithm thus reaches the nearest value by eliminating half of the set under consideration.

Example, if the looktheta MSB is 1, then the algorithm eliminates the first 128 of the 256 states and starts searching in the states between 128 and 256. With next bit also being 1, the next 64 states are eliminated and the search segment reduces to 191- 256 and so on.

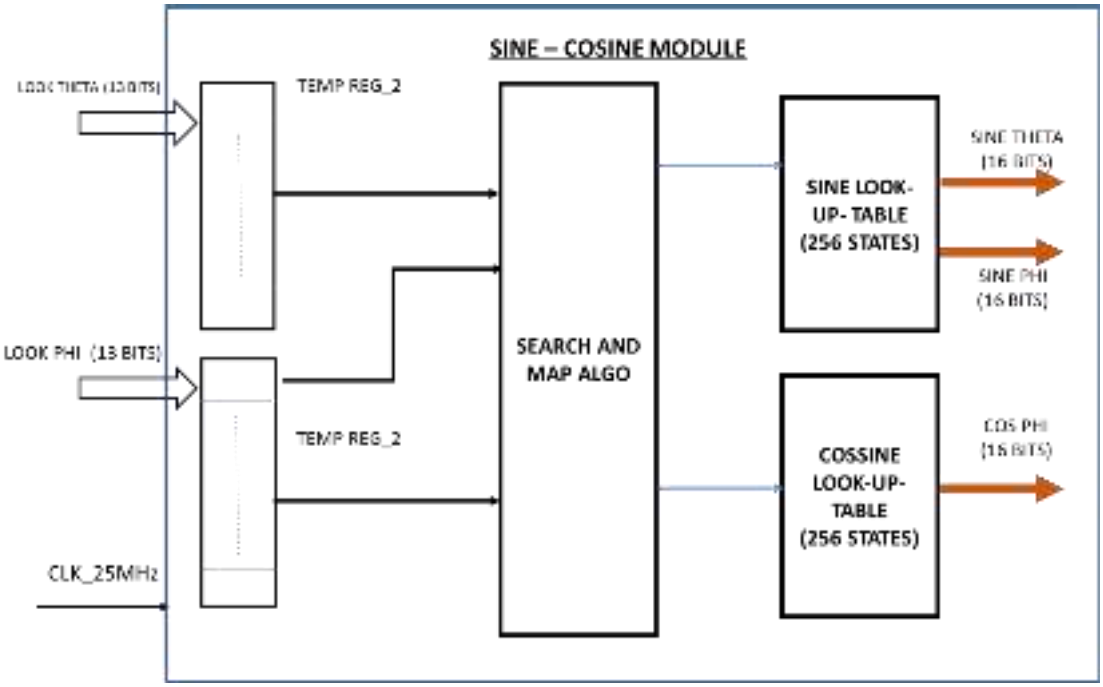


Fig 3.4 Sine – Cosine Module

(b) **Beta x – Beta y Module**. This module is the most complex module of the architecture. This module comprises of the following: -

- Memory registers (03 in numbers) for storing the value of frequency of operation, wavelength, the wave number ($k = 2\pi/\lambda$), distance between the array elements ($\lambda/2$).
- 06 Booth Multiplier modules for signed fixed point number multiplication (as discussed in chapter 2).
- Two 2's complement modules which are used depending upon the value of the sign bit.
- Shift Right registers for dividing the multiplier output by adequate number of bits to get the correct answer.

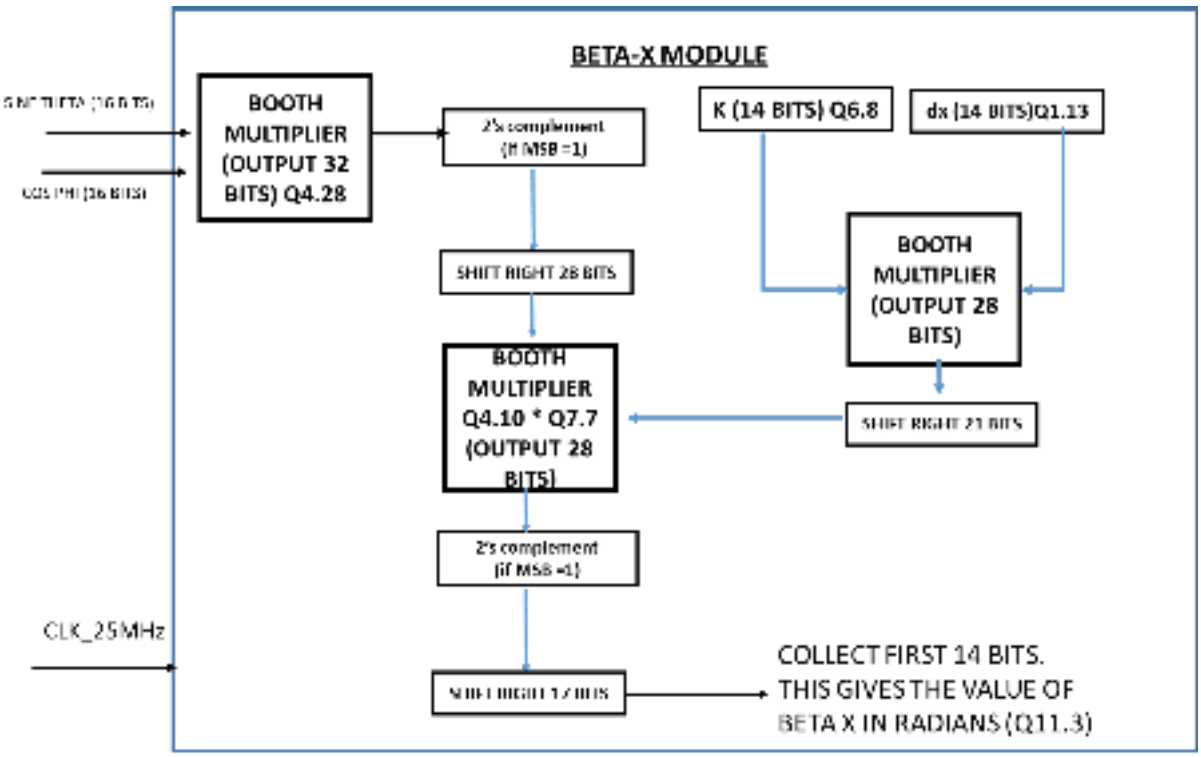


Fig 3.5 Beta X Module

(Remember: - "QM.N" is a fixed point representation where M is the integer word length and N is the fractional word length of a fixed point representation).

The Beta Y module works on similar lines as Beta X.

(c) **Phase (I, J) calculation module**. For the calculation of Phase angle for individual array elements, the value of $\boldsymbol{\beta x}$ and $\boldsymbol{\beta y}$ are used. The same was discussed in chapter 2. The mathematics used is as follows: -

For I = 1 to 4

For J = 1 to 4

Phase (I,J) = I * $\boldsymbol{\beta y}$ + J * $\boldsymbol{\beta x}$

Phase_deg (I,J) = Phase(I,J) *180/pi;

Phase_state (I,J) = Phase_deg/1.40625;

Phase_state_bin = Dec 2 Bin (Phase_state);

End

End

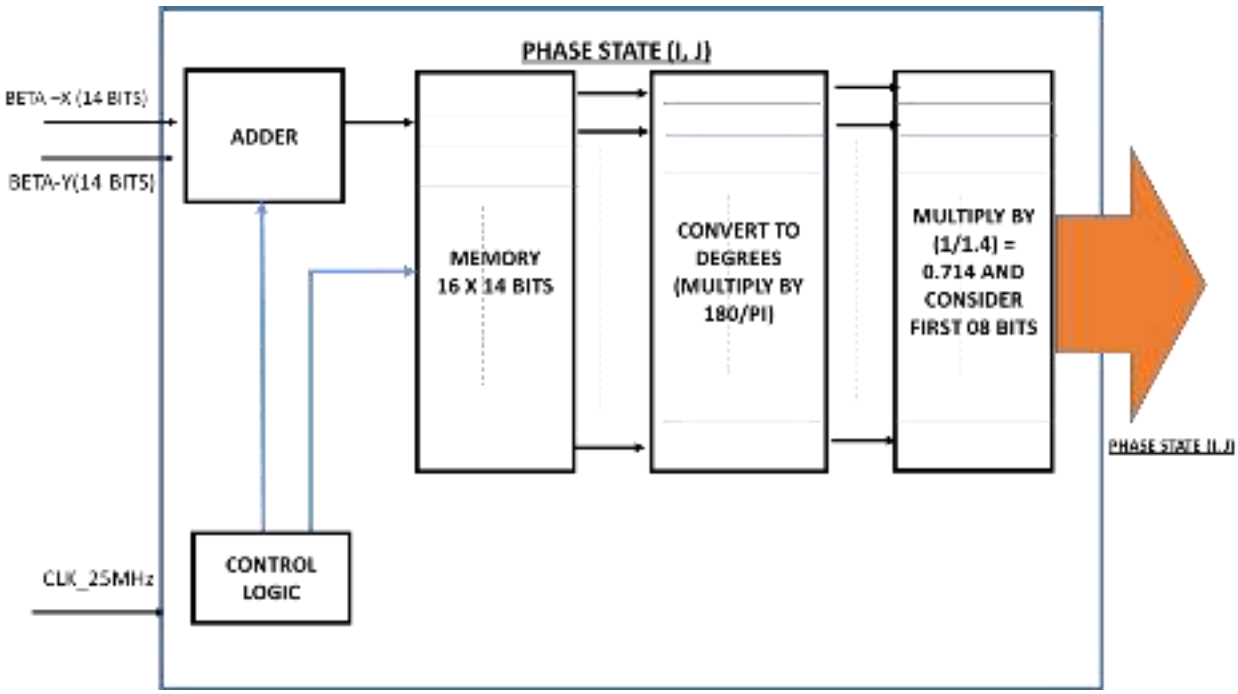


Fig 3.6 Phase State Module

The phase state output is 16 sets of 08 bits each for each array element. The output is further multiplexed with taper value, mode select and other command bits and transmitted to antenna array via interface module.

(d) **Taper State module**. This module gives the 08-bit state value for the amplitude excitation of each element. The distribution in our case has been assumed to be Taylor Distribution. The algorithm is as follows: -

- Taper1 = taylorwin(4);
- Taper2 = taylorwin(4);
- for a = 1 to 4
    - for b = 1 to 4
    - taper (a,b) = taper1(a) x taper2(b);
    - end

- end

taper =

0.3472	0.8312	0.8312	0.3472
0.8312	1.9903	1.9903	0.8312
0.8312	1.9903	1.9903	0.8312
0.3472	0.8312	0.8312	0.3472

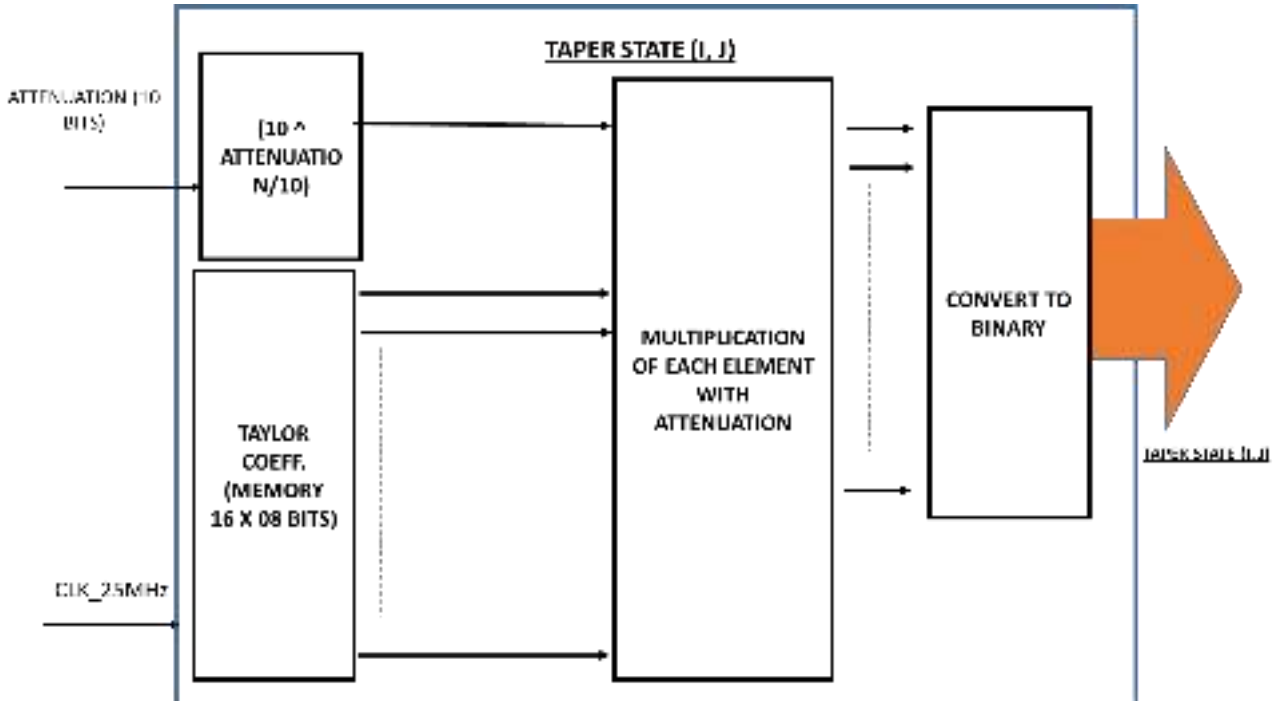


Fig 3.7 Taper State Module

The in-built exponent operator in VHDL has been used to calculate the decimal value of attenuation. The input value of attenuation is multiplied with each element taper 10^ attenuation(dB)/10 * taper) and the first 08 bits are considered as a sensible approximation for taper state.

CHAPTER 4.

WORK DONE –II

CUSTOMIZED PERIPHERAL INTERFACE

4.1 **Understanding Interface signals and their functions**. The signals enable 'a', clk' burst and 05 links of 08-bit serial data together form the peripheral interface between the FPGA board and the antenna array. The clock speed of the system is intended to be 25MHz.

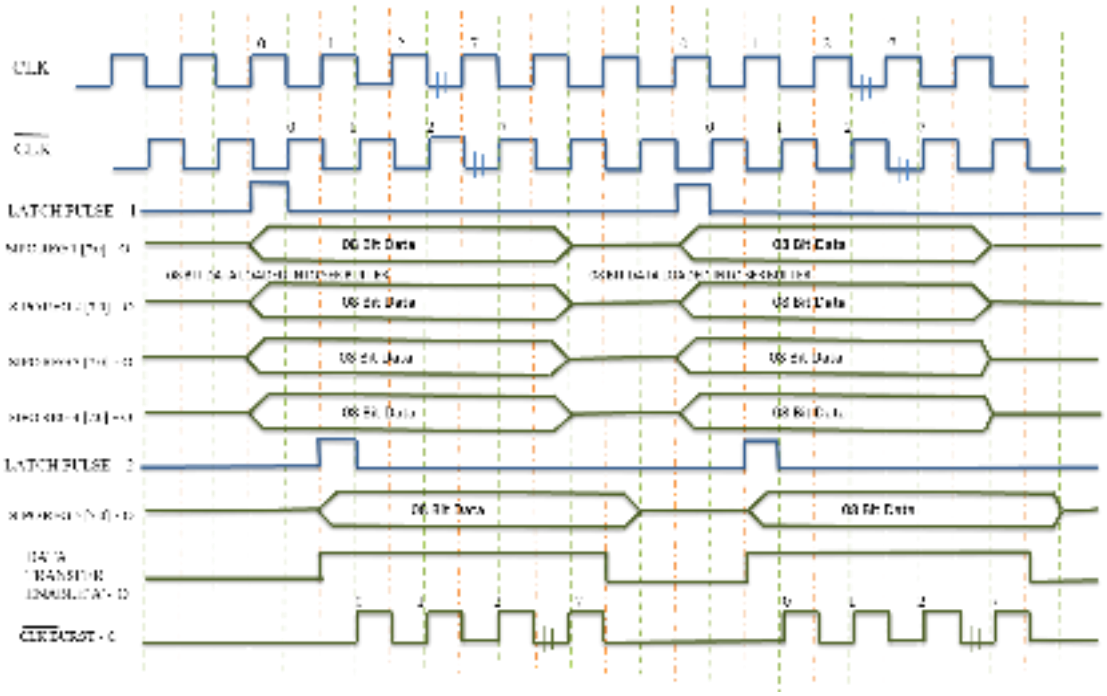


Fig 4.1 SPI Interface – Waveforms

The timing diagram and synchronization of the signals is explained in the document. As long as enable 'a' is active high, the data and the burst clock are made available to the chip of the array element which is selected by the selection lines. The flag data bit is read back in an additional clock cycle. If the data flag is '0', then the five sets of 08-bit data are retransmitted for the same array element. The number of such iterations can be modified at a later stage to suit our latency requirement. We have made the interface module such that it is useful as a standalone module when we intend using the inbuilt 32 bit RISC-V microprocessor. Therefore, we require an additional clock cycle to load/latch the 40-bit data to the 08-bit shift registers of the parallel to serial conversion block.

(a) **Selection lines**. The selection lines are the output of the FPGA board which are used to select the particular array element with the help of a 1 X N^2 De-multiplexer that routes the enable 'a' signal to the relevant chip to initiate 32-bit data communication between FPGA board and antenna array element. This arrangement allows use of less output lines from the FPGA board to select a large number of array elements. Similarly, the flag data bit is read from all the N array elements through an N X 1 multiplexer.

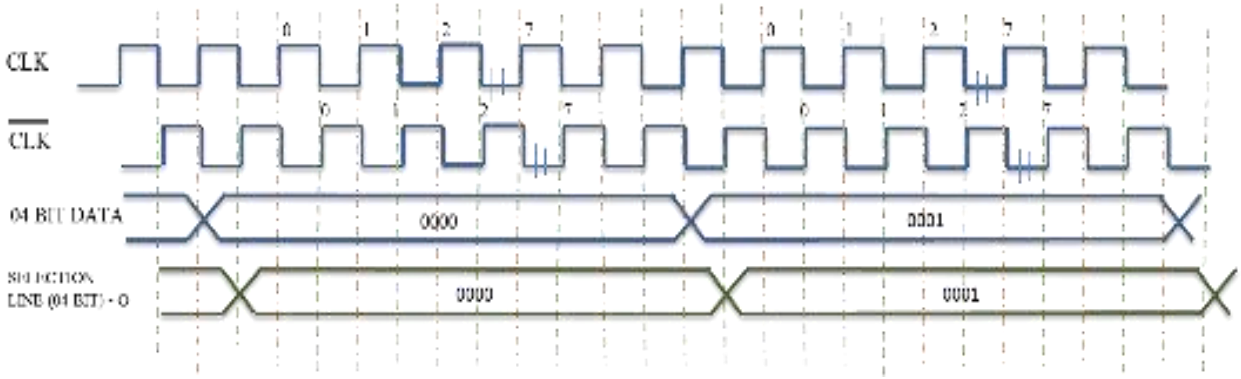


Fig 4.2. Selection lines

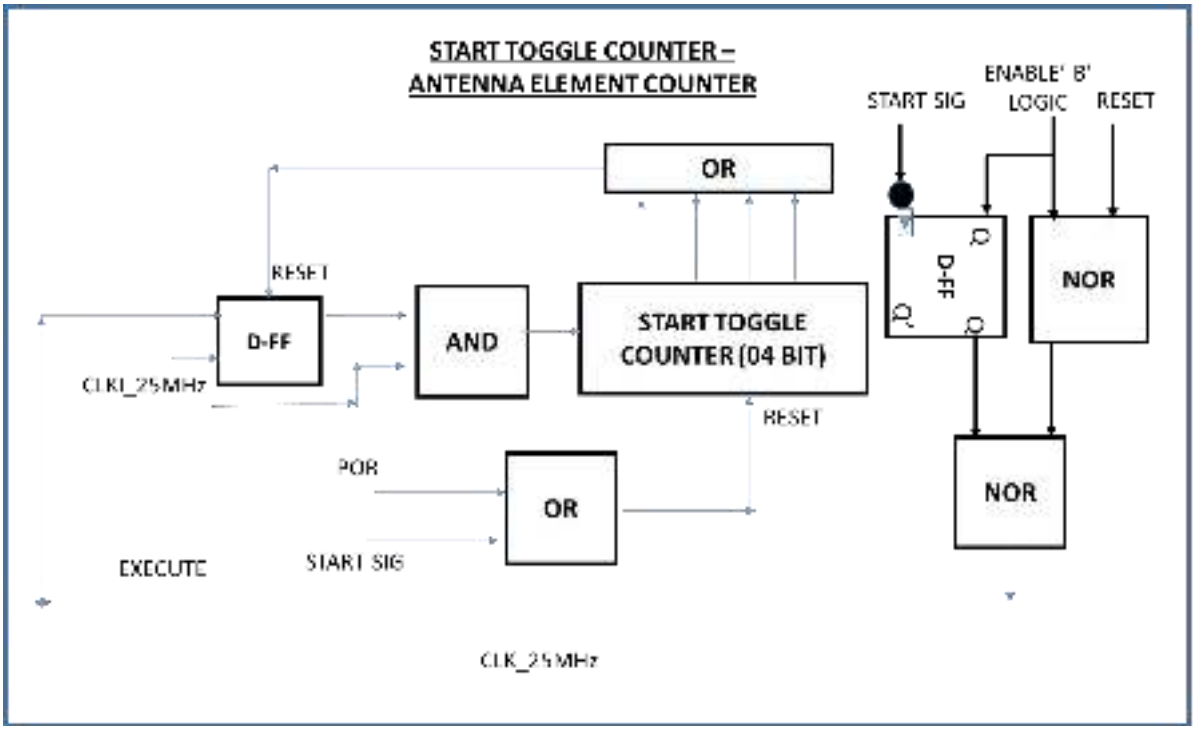


Fig 4.3 Start Toggle Counter – The change of state of this counter shifts data transmission from one Array Element to Another

(b) **Enable 'b'/Fire**. This is a user driven interface signal. The 05 sets of 08-bit phase/attenuation level /operation mode data gets loaded into the registers of all the array element chips, however, the transmission only starts when the enable 'b'/fire signal is given by the user. If the fire signal is initiated in between the load cycle, the DC holds the signal for the duration in which all array elements get loaded with the requisite data for the beam to look in a particular direction. Once all 'N' array elements are loaded, the fire command is initiated by DC. All data communication between DC and Antenna panel is paused during the duration of transmission. The loading cycle starts once again after completion of firing/transmission operation.

(c) **Reset**. This is also a user driven signal to hard reset the registers of all the array element chips to "0" and the data loading cycle continues again.

4.2 **Implementation Design**. The present design caters for a 4 X 4 array antenna (the design can be extended to larger array wherein the selection line bits would be $\log_2 N$). The computational algorithm resides in the processor of the FPGA board. The 40-bit data (05 sets of 08 bit each) required for each array element along with its address (0000 for 1st array element, 0001 for 2nd and so on) is generated and synched. The control logic synchronizes the enable 'a' signal and latch pulse. The enable 'a' signal activates all 05 SIPOs of the relevant antenna array element through the Demux to receive the data and the latch pulse loads the data from the processor to five 08-bit shift registers for parallel to serial flag conversion simultaneously. The data communication takes place through 05 serial links (SD-1 to SD-5). The data flag is read just after loading of the 08-bit data thereby indicating correctness of the transmission channel. The enable 'b' and reset are hardwired signal interrupts generated as per user requirement.

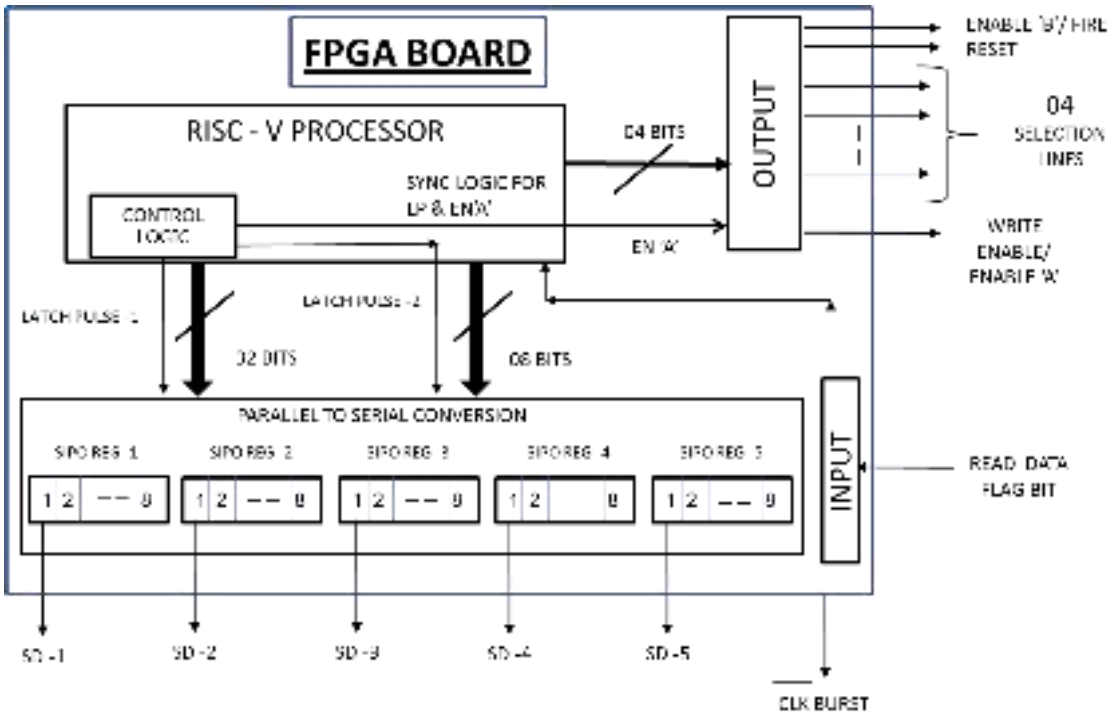


Fig 4.4 FPGA Signal Diagram – Implementation Design

4.3 **Module wise Architecture and expected Timing diagrams**. The module wise architectures, expected timing and the signal waveforms are shown in figures. The

signals indicated in blue are internal to the FPGA board. The signals in green are the SPI signals from the master (FPGA board) to the slave (antenna array). The one in yellow is being read back from the antenna array and the red signals are the user driven interrupts. We assume the data to be sampled and loaded at the rising edge of the clock for all operations.

(a) **CLK**. The clk signal is the internal clock running at 25 MHz (generated form a 50 MHz oscillator by a divide-by-2 circuit). This governs the timing of all computation and communication processes.

(b) **CLK'**. The clk' or $\overline{clk}$ (inverted clock) is generated internally to cater for the timing lag generated due to transmission channel and therefore, this becomes reference for the interface and data communication.

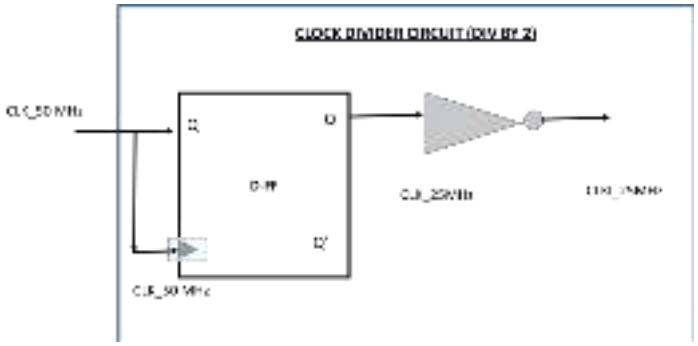


Fig 4.5 Clock Divider Circuit and Inverted Clock Generation (50MHz to 25 MHz)

(c) **Selection line**. The selection line is the 04-bit data which acts as an address to the array element in which 05 sets of the phase/attenuation 08-bit data are is to be loaded. This being an outgoing signal, the activation takes place at the rising edge of clk. The selection lines are also the input to the Multiplexer in the read chain and therefore remain active for an extra clock cycle to enable data flag bit to be read back by DC.

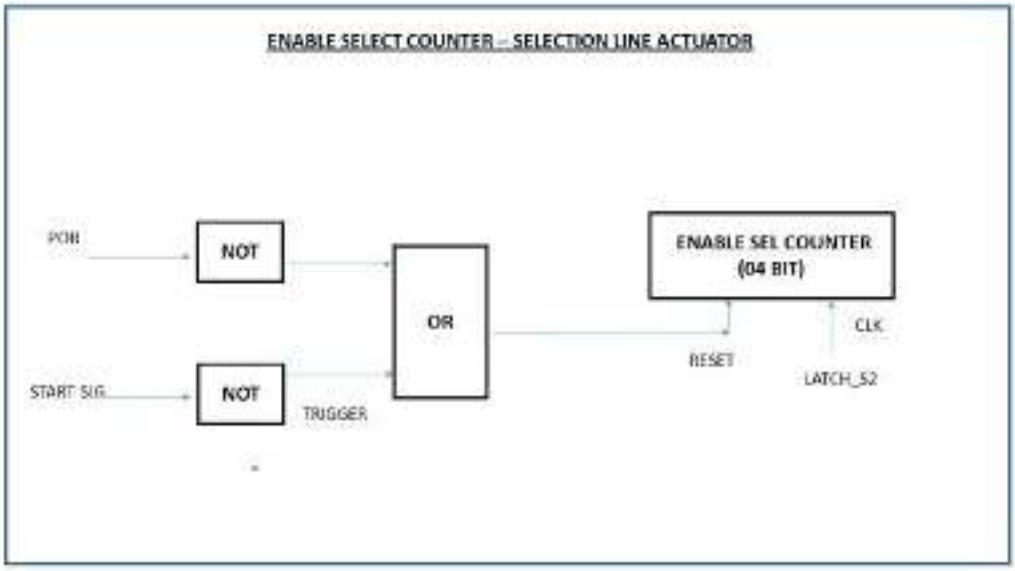


Fig 4.6 Selection Line Actuator

(d) **Latch Pulse 1 & 2**. As soon as the selection lines are activated, the control logic generates the latch pulse-1. This happens at the next rising edge of the clk signal. The first 32-bit data gets loaded in the first 04 dedicated shift registers of parallel to serial converter module with the rising edge of the latch pulse. The last 08 bits of the computational data require another latch pulse to load into the fifth shift register of the P-to-S conversion block. With the advent of the second latch pulse, the entire data is ready for transmission.

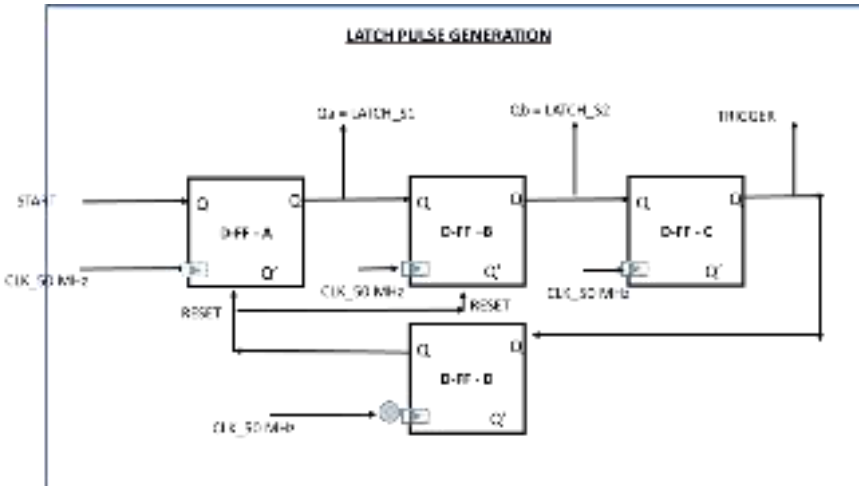


Fig 4.7 Latch Pulse Generator

(e) **32 bit/ 08bit Data**. The total data of 40 bits is required to be sent to the antenna element through 05 serial links (each link carrying 08 bit of data dedicated for a particular SIPO of the antenna element chip). The RISC-V processor of the FPGA board is a 32-bit processor and can process/store only 32-bit data in one clock cycle. The remaining 08 bits of data require another clock to process. The data once ready, is required to be transmitted from FPGA board to antenna chip. Post processing of the computation algorithm, the selection lines and five serial links of 08-bit data are the customized outputs for each array element in which the data changes with every load cycle.

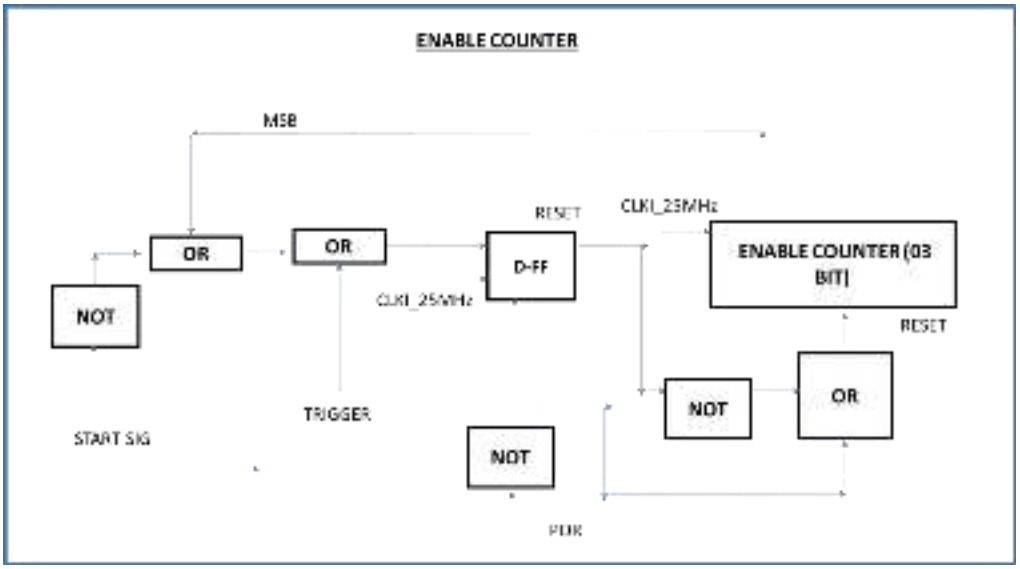


Fig 4.8 Enable Counter – Ensures timing for 08-bit data transfer to SIPO of Antenna Array

(f) **Enable 'a'**. This signal is the write enable which activates the array element chip for receiving data. This gets active high with the rising edge of the latch pulse-2 and remains active for exactly 08 clock cycles till the end of data transmission.

(g) **CLK Burst**. This signal is the actual clock that flows between DC and antenna array and governs SPI interface. This remains active only for the time enable 'a' is high. The sampling and loading of data takes place at the rising edge of this clock.

(h) **Data ack flag**. This is the last bit of one of the 08 bit registers which is set to be '1' before transmission. This flag bit is '0' by default in the array ICs. As soon as the data gets loaded the flag bit '1' is being read back in the next clock cycle and the combinatorial logic in the DC then changes the selection line to another state. If the flag bit is read as '0' then the selection lines remain in the earlier state and the previous 40-bit data (all five sets of 08 bits) is retransmitted to the array element.

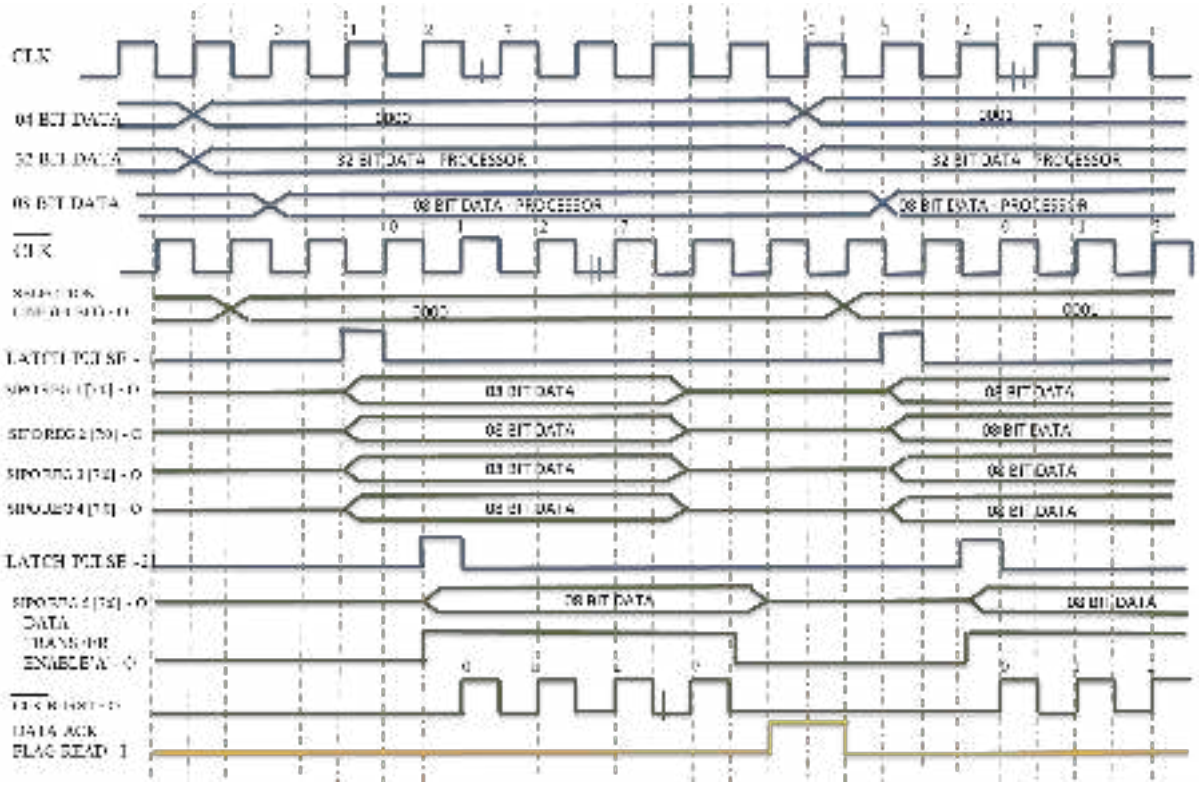


Fig 4.9. FPGA Board - Smart Antenna Communication Timing Diagram

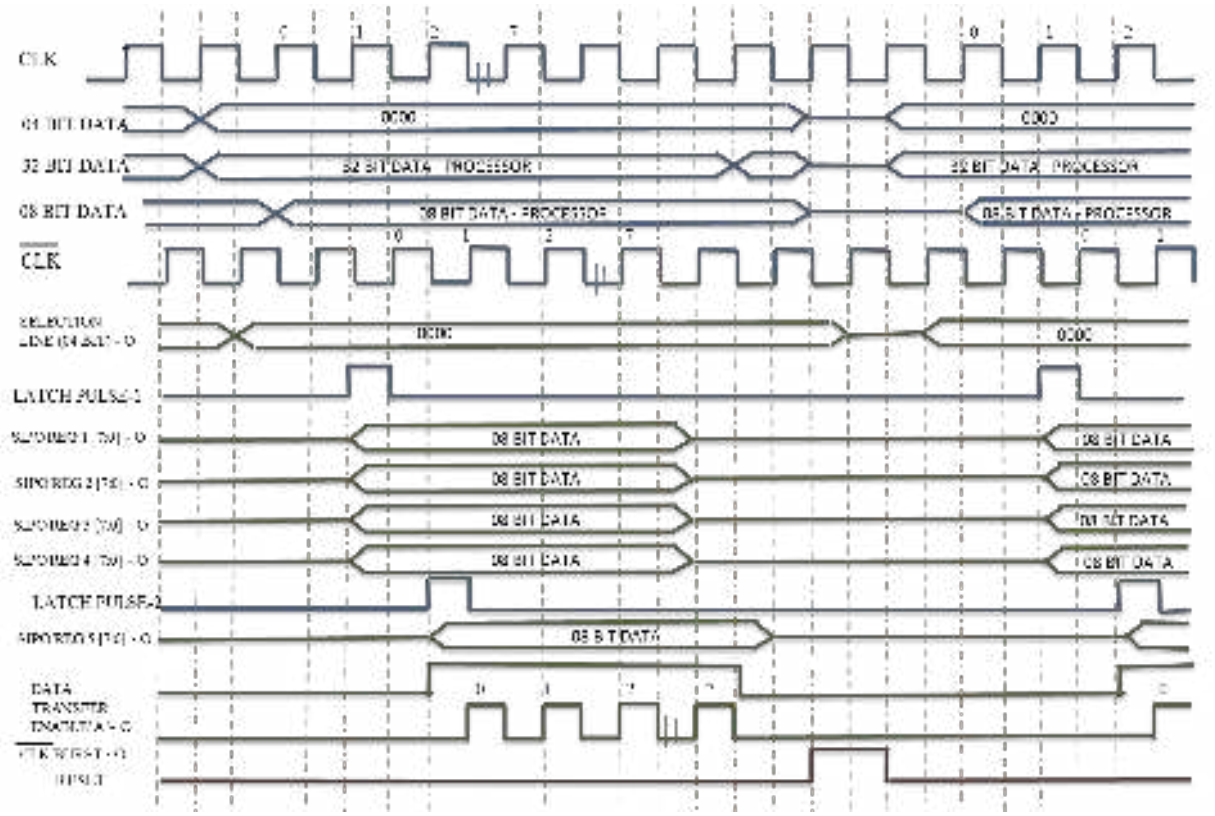


Fig 4.10. Timing Diagram with Reset Interrupt Signal

CHAPTER 5.

RESULTS AND DISCUSSIONS

5.1 **Introduction**. In this chapter we observe results of variety of inputs to the models developed in MATLAB and VHDL. The Validation of the algorithm will also be undertaken and the RMSE error between the AF_simulated and the AF_verified will be calculated for a few cases. The performance of the VHDL modules will also be analyzed and the simulated timing diagrams will be displayed.

5.2 **MATLAB Results**.

(a) **Case 1**: (Array = 4x4, looktheta = 45, lookphi = 120, attenuation = 10dB)

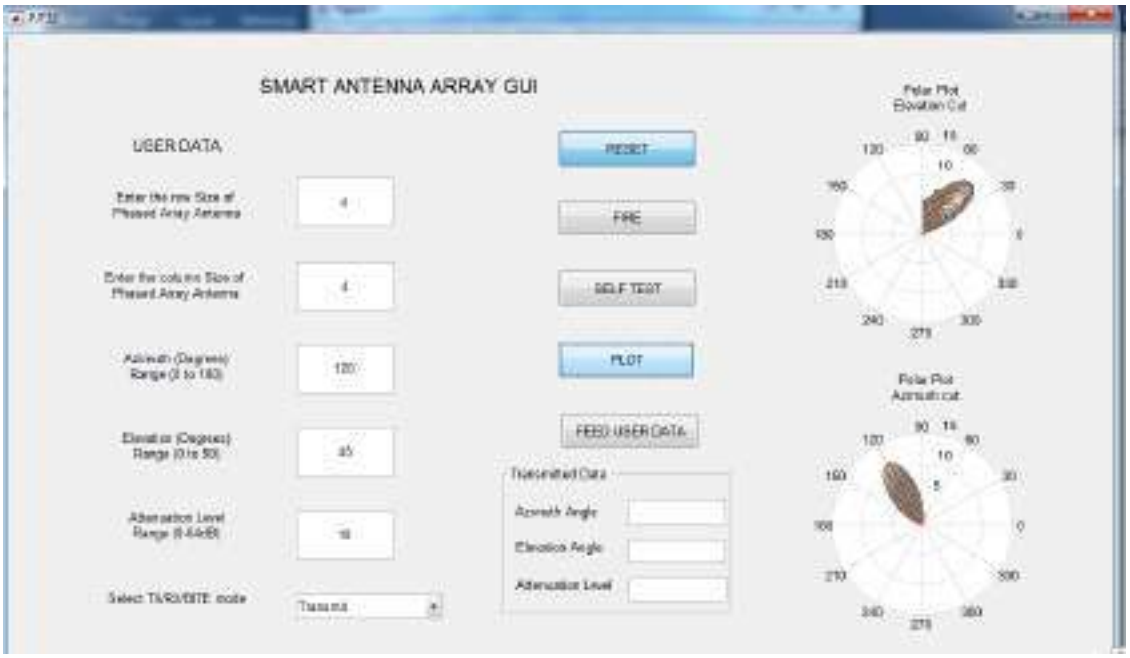


Fig 5.1 GUI Interface for case 1

taper =

0.3472	0.8312	0.8312	0.3472
0.8312	1.9903	1.9903	0.8312
0.8312	1.9903	1.9903	0.8312
0.3472	0.8312	0.8312	0.3472

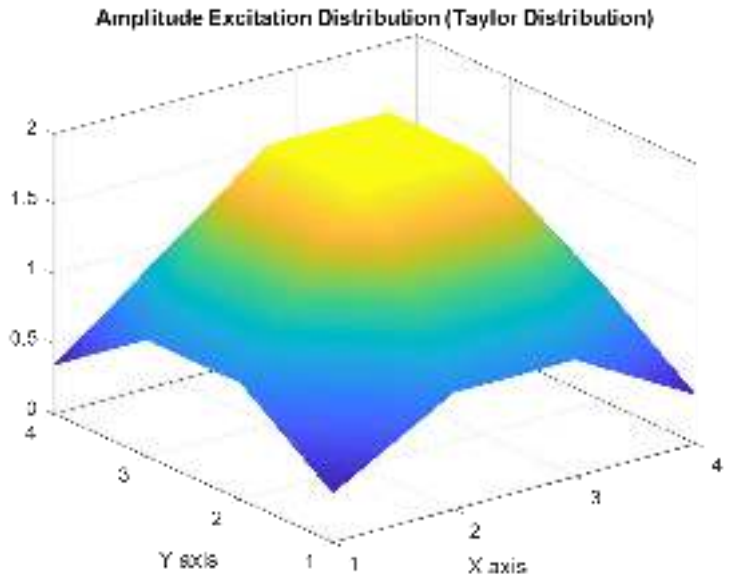


Fig 5.2 Taper Distribution for case1

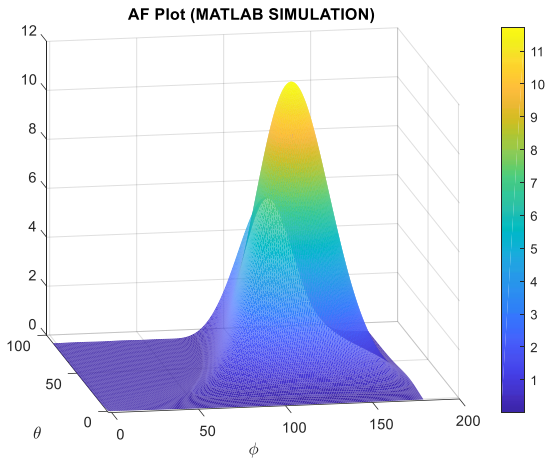


Fig 5.3. AF_Simulated

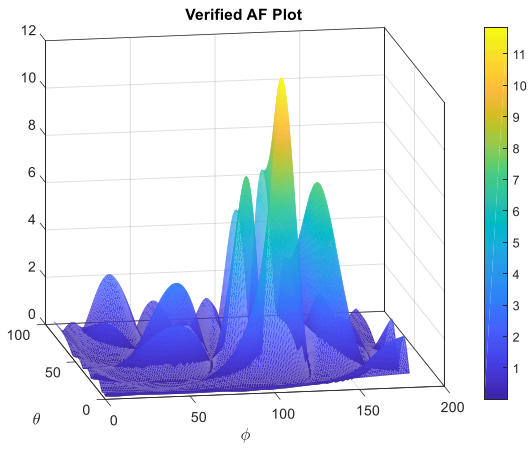


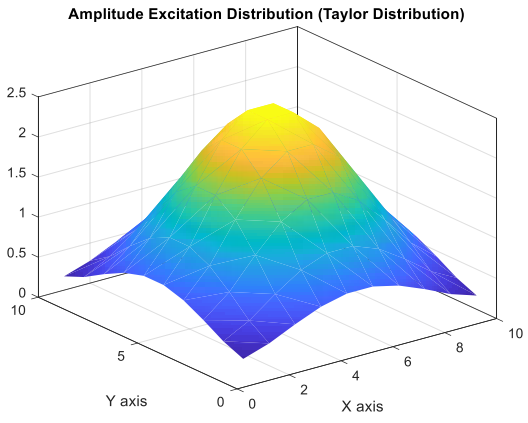
Fig 5.4. AF_Verified

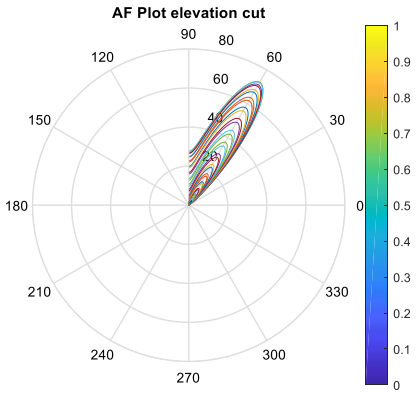
Phase State for all 16 elements =

'00100001',
'01001110'
'01111100'
'10101001'
'11010011'
'00000000'
'00101101'
'01011010'
'10000100'
'10110010'
'11011111'
'00001100'
'00110110'
'01100011'
'10010001'
'10111110'

The RMSE between the Simulated and Verified AF plot is = **0.507148 dB**

(b) **Case 2** : (Array = 10x10, looktheta = 60, lookphi = 130, attenuation = 20dB)





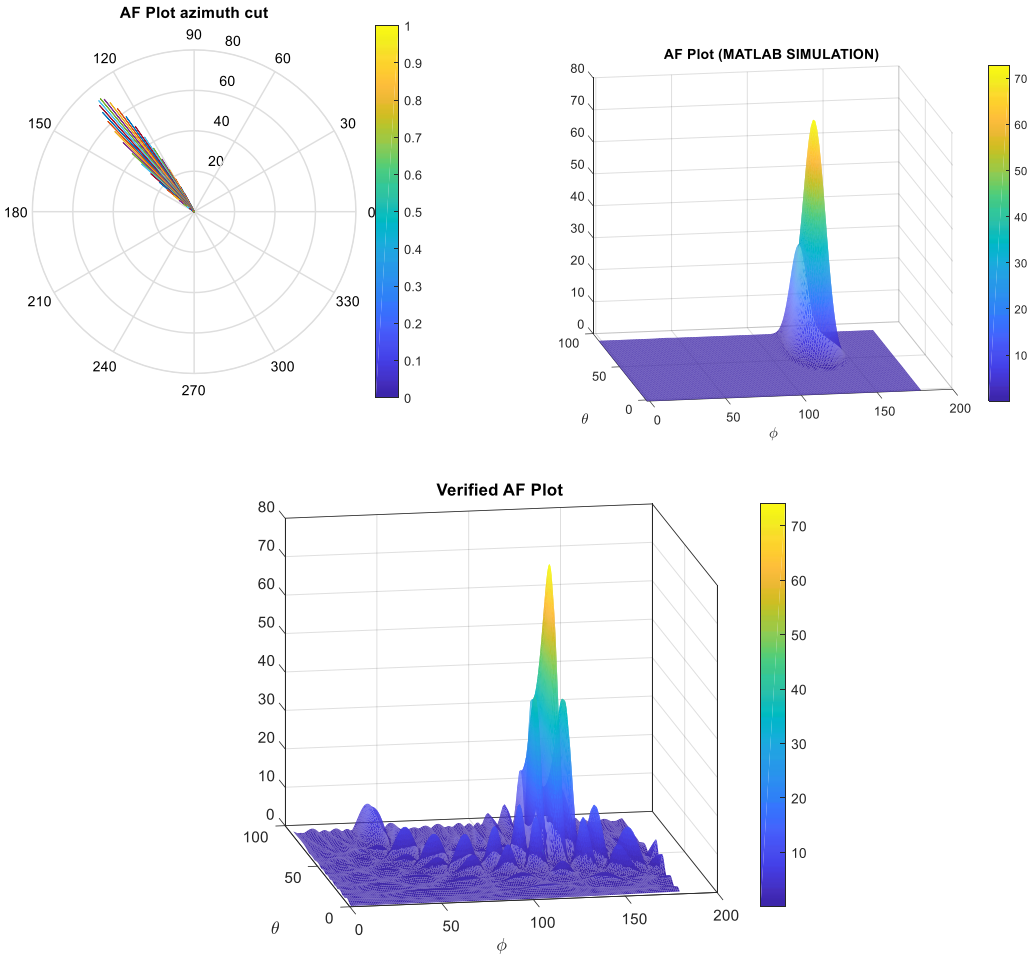


Fig 5.5. Results of MATLAB Simulation of Case 2

The RMSE between the Simulated and Verified AF plot is = **0.321592 dB**

Kindly note that the AF plots are in decimal number scale and the RMSE has been calculated by MATLAB in dB scale. Many cases were tried and the RMSE between AF_simulated and AF_verified was found to be below 1dB. Also, as the number of array elements increases, the RMSE was observed to decrease.

5.3 **FPGA Simulation Results**. The FPGA results assume the array size to be fixed to 4 x 4 and frequency of operation 3.25Ghz. We will analyze two designed modules viz. the Data Processing Block and the customized interfacing module.

(a) **Data Processing Block**

A few details of the synthesis report of the Data Processing block are mentioned as under

```
=========================================================
==============
*                      Design Summary                   *
=========================================================
==============

Top Level Output File Name         : amp_phase_state_fixpt.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                      : 2615
#      GND                  : 1
#      INV                  : 228
#      LUT1                 : 2
#      LUT2                 : 389
#      LUT3                 : 199
#      LUT4                 : 48
#      LUT5                 : 39
#      LUT6                 : 396
#      MUXCY                : 554
#      MUXF7                : 174
#      MUXF8                : 80
#      VCC                  : 1
#      XORCY                : 504
# FlipFlops/Latches         : 13
#      FDC                  : 1
#      FDCE                 : 4
#      LDC                  : 8
# Clock Buffers             : 1
#      BUFGP                : 1
# IO Buffers                : 78
#      IBUF                 : 38
#      OBUF                 : 40
# DSPs                      : 74
#      DSP48E1              : 74
```

Device utilization summary:

Selected Device : 7a100tcsg324-3

Slice Logic Utilization:
Number of Slice Registers: 5 out of 126800 0%
Number of Slice LUTs: 1301 out of 63400 2%
Number used as Logic: 1301 out of 63400 2%

Slice Logic Distribution:
Number of LUT Flip Flop pairs used: 1301
Number with an unused Flip Flop: 1296 out of 1301 99%
Number with an unused LUT: 0 out of 1301 0%
Number of fully used LUT-FF pairs: 5 out of 1301 0%
Number of unique control sets: 3

IO Utilization:
Number of IOs: 79
Number of bonded IOBs: 79 out of 210 37%
IOB Flip Flops/Latches: 8

Specific Feature Utilization:
Number of BUFG/BUFGCTRLs: 1 out of 32 3%
Number of DSP48E1s: 74 out of 240 30%

Timing Summary:

Speed Grade: -3

Minimum period: 1.464ns (Maximum Frequency: 682.967MHz)
Minimum input arrival time before clock: 51.515ns
Maximum output required time after clock: 1.425ns
Maximum combinational path delay: 1.112ns

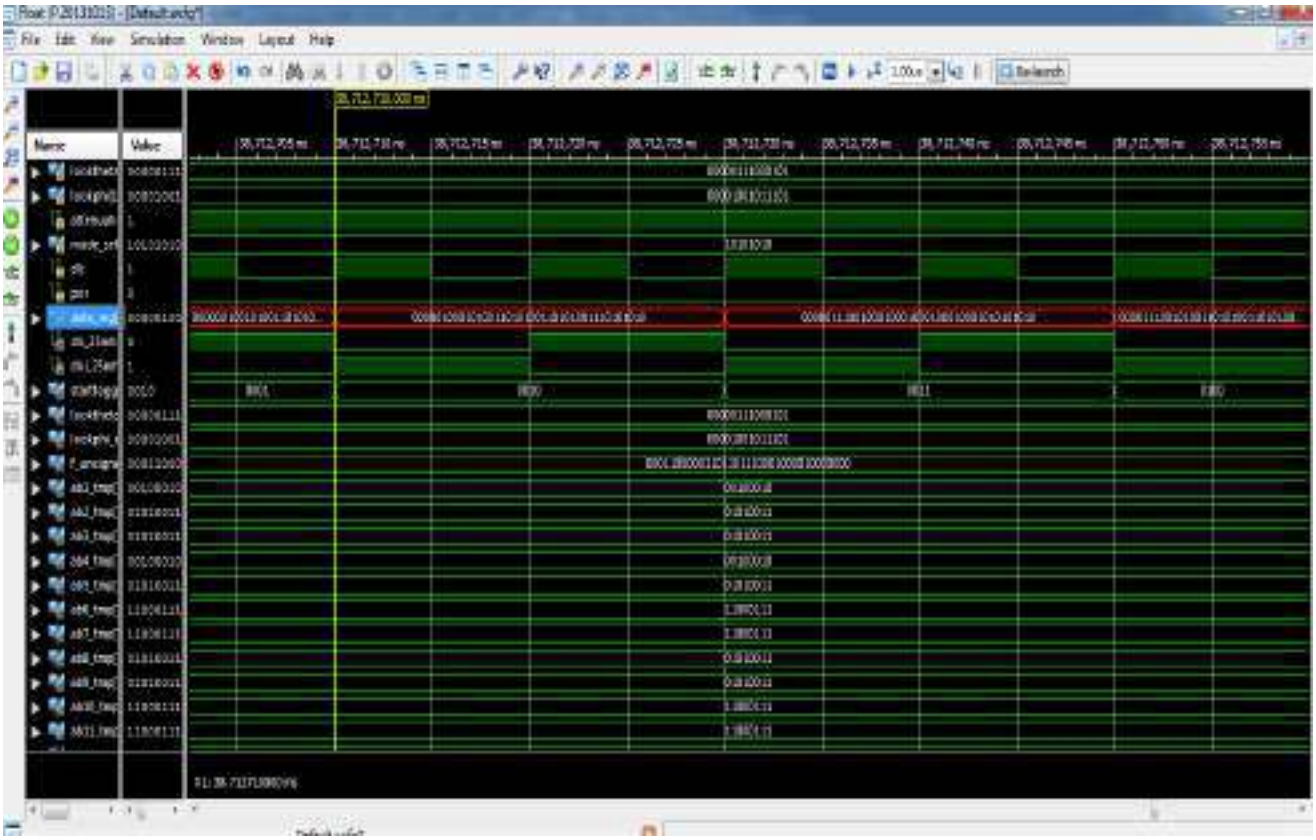


Fig 5.6. Simulation Result of Data Processing Module

The 40 Bit data register has been highlighted in red. The First 08 bits determine the phase state, the next 24 bits represent 03 copies of the attenuation state. The next 04 bits represent mode of operation (Tx/Rx/Bite) and Reset/Fire commands. It can be observed that the 40-bit data register changes value for every state change in the start toggle register. This implies that the previous data has been transferred to the designated Antenna Array Element and the next data can be loaded into the temporary data register of interface module.

(b) **Customized Interface Module**.

```
=========================================================
==============
*                  Design Summary                  *
=========================================================
==============
```

Top Level Output File Name : spicode.ngc

Primitive and Black Box Usage:

BELS : 145
GND : 1
INV : 5
LUT2 : 55
LUT3 : 76
LUT4 : 4
LUT5 : 2
LUT6 : 1
VCC : 1
FlipFlops/Latches : 148
FD : 40
FDC : 10
FDC_1 : 1
FDCE : 5
FDE : 5
FDP : 44
LDC : 40
LDC_1 : 1
LDCE : 2
Clock Buffers : 4
BUFG : 3
BUFGP : 1
IO Buffers : 17
IBUF : 4
OBUF : 13

Device utilization summary:

Selected Device : 7a200tfbg676-1

Slice Logic Utilization:
Number of Slice Registers: 141 out of 269200 0%
Number of Slice LUTs: 143 out of 134600 0%
Number used as Logic: 143 out of 134600 0%

Slice Logic Distribution:
Number of LUT Flip Flop pairs used: 227

Number with an unused Flip Flop: 86 out of 227 37%
Number with an unused LUT: 84 out of 227 37%
Number of fully used LUT-FF pairs: 57 out of 227 25%
Number of unique control sets: 91

IO Utilization:
Number of IOs: 18
Number of bonded IOBs: 18 out of 400 4%
IOB Flip Flops/Latches: 7

Specific Feature Utilization:
Number of BUFG/BUFGCTRLs: 4 out of 32 12%

Timing Summary:

Speed Grade: -1

Minimum period: 3.856ns (Maximum Frequency: 259.336MHz)
Minimum input arrival time before clock: 1.916ns
Maximum output required time after clock: 1.702ns
Maximum combinational path delay: 0.406ns

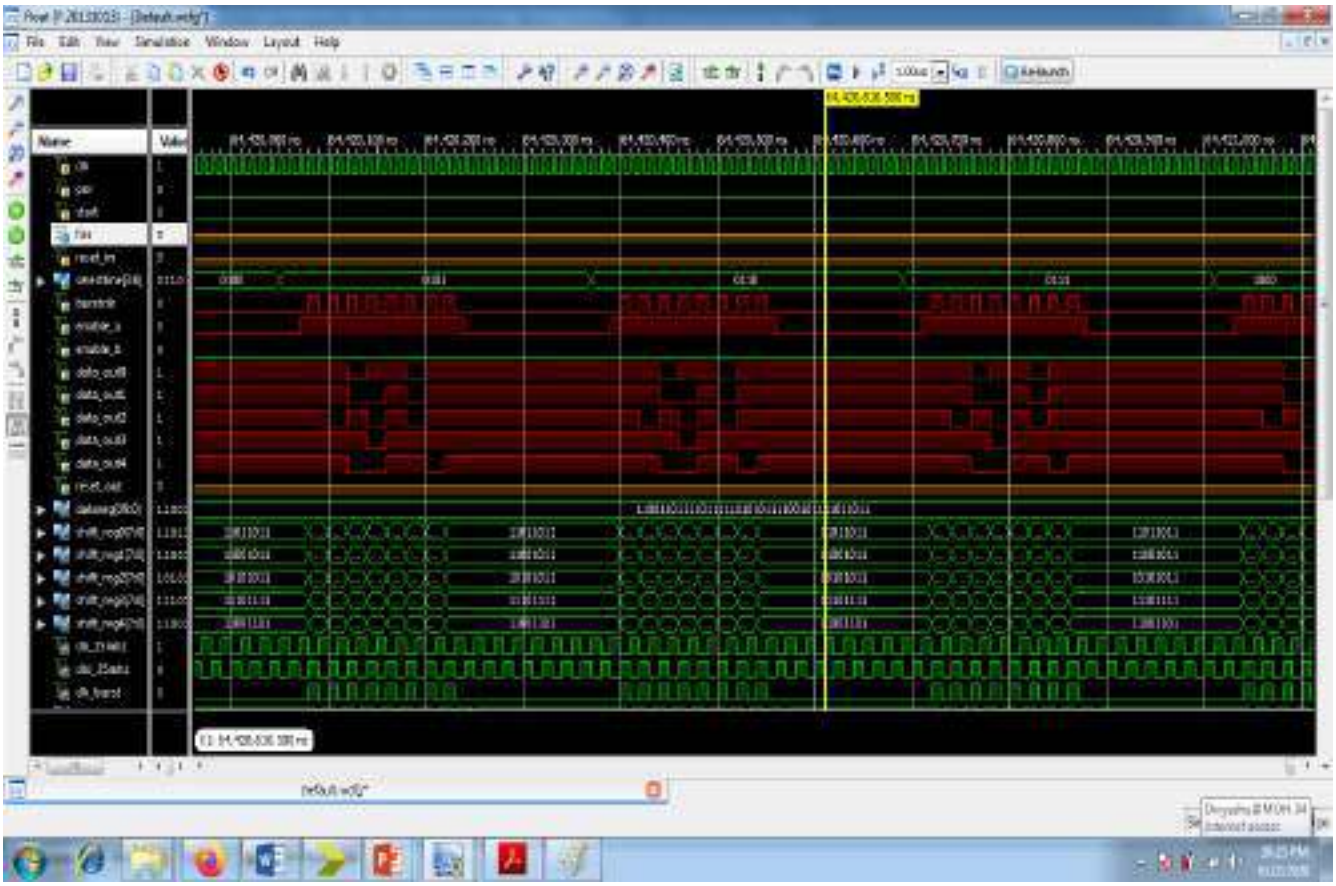


Fig 5.7 Simulation of the Customized interface module

The lines highlighted in red are the lines going out of the FPGA carrying essential data/signals. The 05 data_out registers carry the data for the 05 SIPOs of the antenna array. The response of the module to the change in status of Reset, Fire and other signals was as expected. The design included many concurrent processes that consumed relatively larger space on the FPGA but resulted in a faster design.

If the Fire signal is sent from the C&C, the FPGA waits till the data is loaded into all the 16 array elements and as soon as the data is loaded, the enable -'b' signal is activated. If the reset signal is received from C&C, the FPGA immediately sends to command to Antenna and all registers of the antenna array are reset to '0'. Both these cases are depicted in figures 5.8 & 5.9

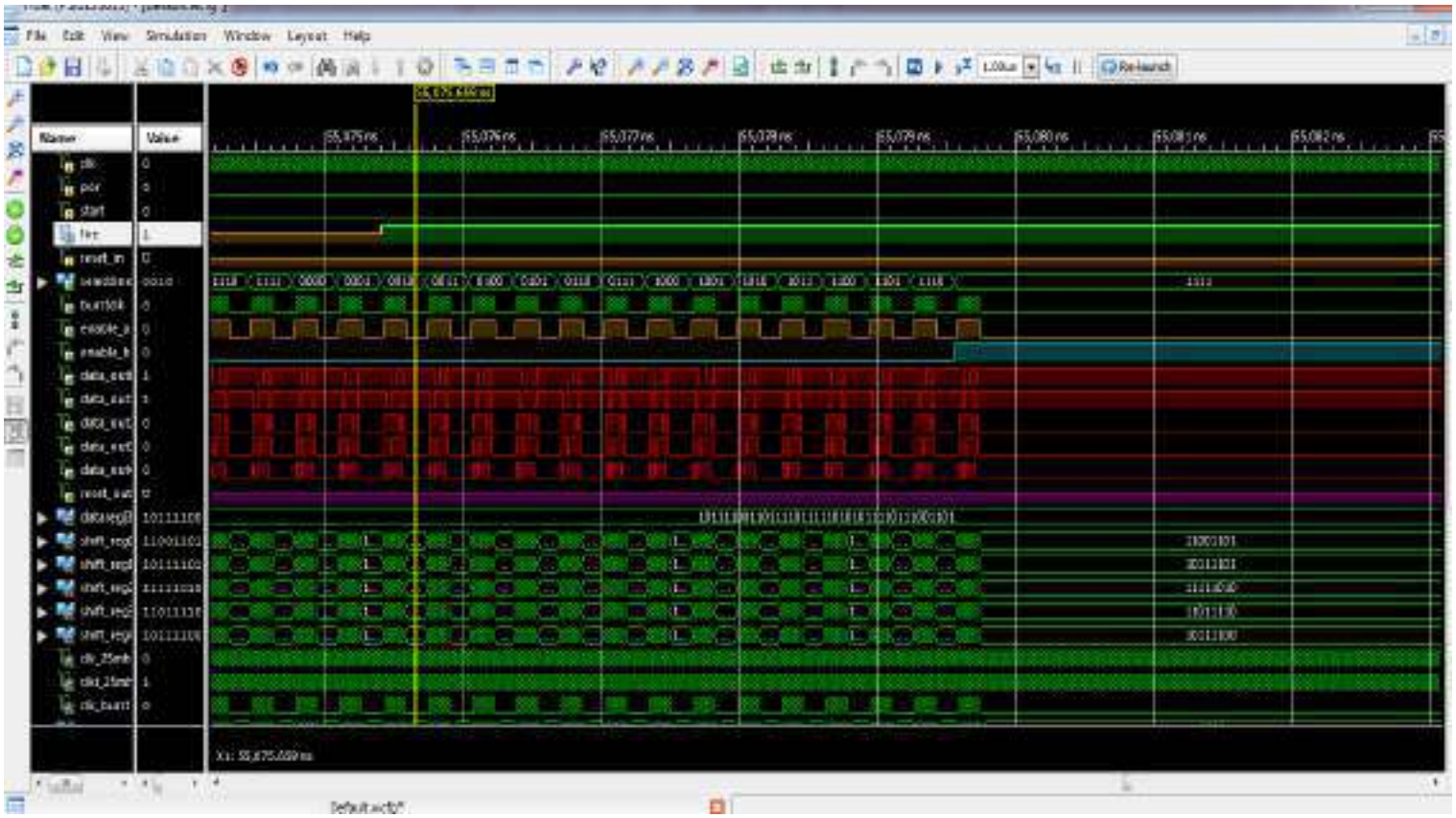


Fig 5.8 Fire Command Timing Diagram

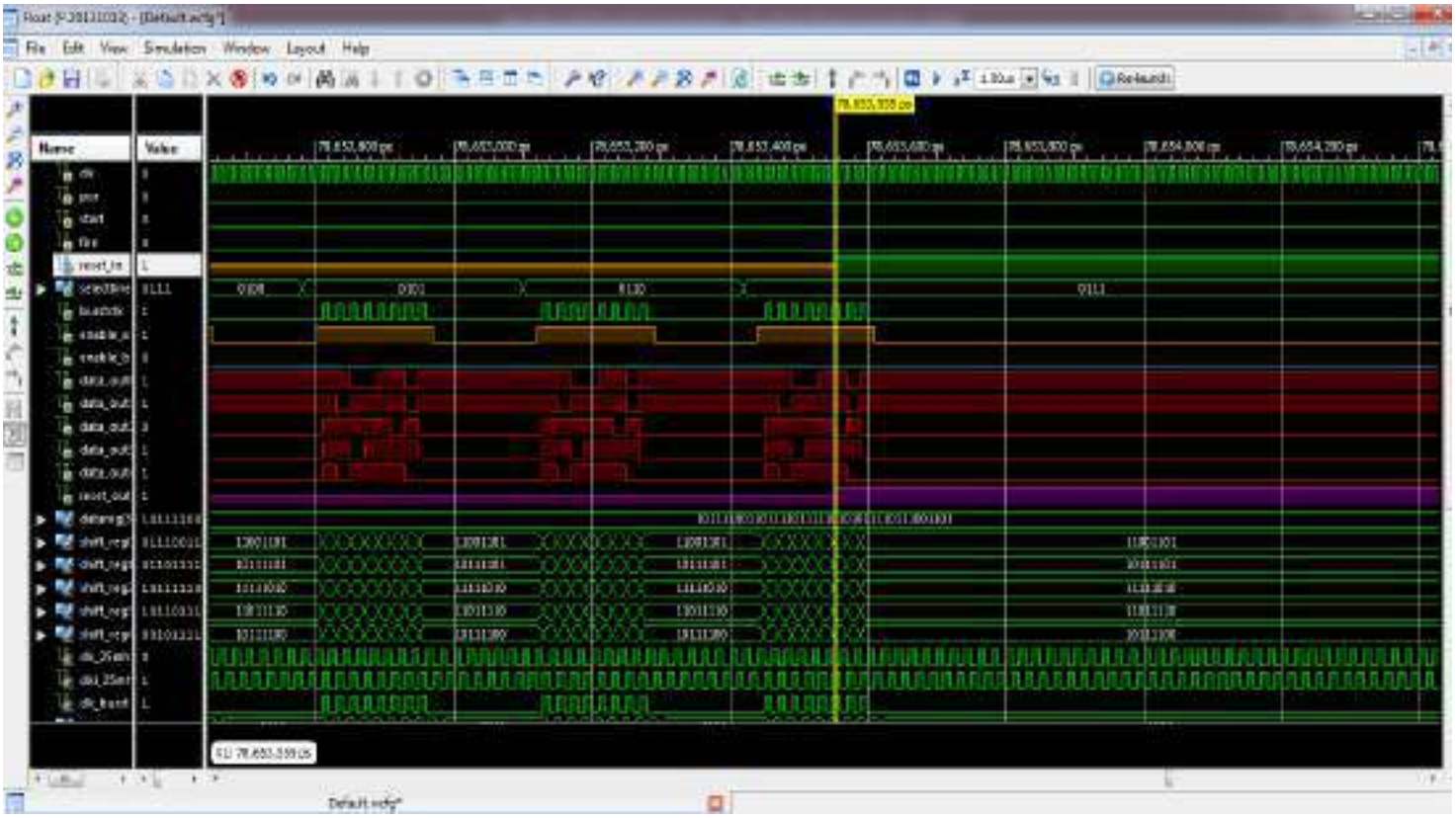


Fig 5.9 Reset Command Timing Diagram

CHAPTER 6.

CONCLUSION & FUTURE WORK

6.1 **Conclusion**. A basic customized System-on-chip was designed and developed for an intelligent FPGA interface with the C&C and Antenna Array. The addition of temperature compensation algorithm is carried forward and can be completed by either creating a parallel compensation offset module or by a mere insertion of look-up table that selects the relevant resonant frequency of the patch microstrip antenna to maintain the desired performance. A GUI and a standalone application was successfully developed for user interaction. The algorithm for calculation of phase and attenuation states was validated by comparing the theoretical and practical Array factors (less than 1dB error). The off-the shelf communication protocol RS-232 was used to communicate between C&C and DC. The data processing block and interface module were developed and were successfully simulated. This basic skeleton forms a critical part of the system and will serve as a foundation for further additions of intelligent features in the algorithm. The flexibility of FPGA in building parallel combinational logics is a handy tool to add features without drastically affecting the speed of operations. The present structure can be upgraded for various enhancements in the future. Some of them will be discussed in the next segment.

6.2 **Future Work**. The prototype developed in this project can be used in various future enhancements. Some of them are mentioned below: -

- Designing of compensation module to cater for different ambient conditions and defective antenna elements.
- Incorporation of Modes of beam steering. Rather than taking one input at a time, the program can be upgraded to store a set of pre-defined inputs that get executed when the user selects a particular mode of operation.
- Use of hardware friendly multipliers and CORDIC algorithms for sine and cosine calculations to save on space on the FPGA module (time delay to be kept under check).
- Co-location of DC and Antenna Array.

BIBLIOGRAPHY

[1] C. A. Balanis, "Antenna Theory," *3rd Edition, USA: John Wiley and Sons, 2005.*

[2] Morris Mano, "Digital Design," *3rd Edition, Prentice-Hall India, 2001.*

[3] Douglas L. Perry, "VHDL Programming by Examples," *4th Edition, McGrow_Hill, 2002*.

[4] Mayur Bansal, "Digital control board for phased array antenna beam steering in adaptive communication application," Dept. Elect. Engg., California Polytechnic State University, 2013.

[5] Norun Abdul Malek, Rob Seager, James Flint, "Beam Steering of four Dipoles Antenna Array using Genetic Algorithm," *IEEE Symposium on Wireless Technology and Applications, 2014*.

[6] R. L. Haupt, "Genetic Algorithm Applications for Phased Arrays," *ACES Journal, vol. 21, no. 3, 2006.*

[7] Michael Y. W. Chia, Teck-Hwee Lim, Jee-Khoi Yin,"Electronic Beam-Steering Design for UWB Phased Array," *IEEE Trans. On Microwave Theory and Techniques, vol. 54, no. 6, June 2006.*

[8] M. M. Abusitta, R. A. Abd-Alhameed, D. Zhou, "New Approach for Designing Beam Steering Uniform Antenna Arrays Using Genetic Algorithms," *in Loughborough Antenna & Propagation Conf., Loughborough, UK, 2009.*

[9] Chris Pell, "Phased Array Radars," *Microwave Exhibitions &Publishers Ltd., 1988.*

[10] J.C Baraza, J. Gracia, Sara Blanc, Daniel Gil, P.J. Gil, "Enhancement of Fault Injection Techniques Based on the Modification of VHDL Code" *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 16, No. 6, June 2008*

[11] Rui Yao, Jungie Du, Ping Zhu, Meiquin Wang, "Structure and Online Self-Repair Mechanisms for Digital Systems Based on System-on-Programmable-Chip", *Journal of Aerospace Information Systems Vol. 15, No. 10, October 2018.*

[12] Pooja Jawandhiya, "Hardware Design for Machine Learning", *International Journal of Artificial Intelligence and Applications (IJAIA), Vol.9, No.1, January 2018*

[13] Sarath Babu and Girish Kumar,"Parametric Study and Temperature Sensitivity of Microstrip Antennas Using an Improved Linear Transmission Line Model", *Senior Member, IEEE, IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, VOL. 47, NO. 2, FEBRUARY 1999*

[14] Zhi Guo, Walid Najjar, Frank Vahid, "A Quantitative Analysis of the Speedup Factors of FPGAs over Processors", University of Clifornia, Symposium on FPGAs, Monterry, CA, Feb 2004.

[15] Maximilian C. Scardelletti*, Senior Member, IEEE*, Jennifer L. Jordan*, Member, IEEE*, and George E. Ponchak*, Fellow, IEEE,* "Temperature Dependency (25 C–400 C) of a Planar Folded Slot Antenna on Alumina Substrate", *IEEE ANTENNAS AND WIRELESS PROPAGATION LETTERS, VOL. 7, 2008*

[16] My X. NGUYEN,Ho Chi Minh City University of Technology, Ho Chi Minh City, Viet Nam, "Hardware-Based Algorithm for Sine and Cosine Computations using Fixed Point Processor"

APPENDIX 'A'

MATLAB CODE

```
function varargout = PJT12(varargin)

% PJT12 MATLAB code for PJT12.fig
% Last Modified by GUIDE v2.5 19-May-2020 13:47:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 0;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PJT12_OpeningFcn, ...
                   'gui_OutputFcn',  @PJT12_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PJT12 is made visible.
function PJT12_OpeningFcn(hObject, eventdata, handles, varargin)

%To close the serial port before using it
    if ~isempty(instrfind)
        fclose(instrfind)
        delete(instrfind)
    end

% hObject    handle to figure
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes PJT12 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = PJT12_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
function azimuth_Callback(hObject, eventdata, handles)
% hObject    handle to azimuth (see GCBO)
data1=guidata(hObject);
data1.s1=get(handles.azimuth,'String');
guidata(hObject,data1);


% --- Executes during object creation, after setting all properties.
function azimuth_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function elevation_Callback(hObject, eventdata, handles)
data2=guidata(hObject);
data2.s2=get(handles.elevation,'String');
guidata(hObject,data2);

% hObject    handle to elevation (see GCBO)


% --- Executes during object creation, after setting all properties.
function elevation_CreateFcn(hObject, eventdata, handles)
% hObject    handle to elevation (see GCBO)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function attenuation_Callback(hObject, eventdata, handles)
% hObject    handle to attenuation (see GCBO)
data3=guidata(hObject);
data3.s3=get(handles.attenuation,'String');
guidata(hObject,data3);



% --- Executes during object creation, after setting all properties.
function attenuation_CreateFcn(hObject, eventdata, handles)
% hObject    handle to attenuation (see GCBO)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function select_Callback(hObject, eventdata, handles)
% hObject    handle to select (see GCBO)
% data6=guidata(hObject);
% data6.s6=get(handles.select,'Value');
% guidata(hObject,data6);




% --- Executes during object creation, after setting all properties.
function select_CreateFcn(hObject, eventdata, handles)
% hObject    handle to select (see GCBO)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function row_Callback(hObject, eventdata, handles)
% hObject    handle to row (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
data4=guidata(hObject);
data4.s4=get(handles.row,'String');
guidata(hObject,data4);

% Hints: get(hObject,'String') returns contents of row as text
%        str2double(get(hObject,'String')) returns contents of row as a
double


% --- Executes during object creation, after setting all properties.
function row_CreateFcn(hObject, eventdata, handles)
% hObject    handle to row (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function column_Callback(hObject, eventdata, handles)
% hObject    handle to column (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
data5=guidata(hObject);
```

```
data5.s5=get(handles.column,'String');
guidata(hObject,data5);


% Hints: get(hObject,'String') returns contents of column as text
%        str2double(get(hObject,'String')) returns contents of column as a
double


% --- Executes during object creation, after setting all properties.
function column_CreateFcn(hObject, eventdata, handles)
% hObject    handle to column (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
% hObject    handle to reset (see GCBO)



% --- Executes on button press in send.
function send_Callback(hObject, eventdata, handles)
% hObject    handle to send (see GCBO)

%Get the state of the toggle_button 'send'
state=get(handles.send,'Value');

%Get the state of the toggle_button 'reset'
state1=get(handles.reset,'Value');

if(state1==1)
        set(handles.reset,'String','RESET activated');
        st1='1';
else
        set(handles.reset,'String','RESET');
        st1='0';
end

%Get the state of the toggle_button 'fire'
state2=get(handles.fire,'Value');

if(state2==1)
        set(handles.fire,'String','FIRE activated');
        st2='1';
else
```

```
        set(handles.fire,'String','FIRE');
        st2='0';
end


%To get select option from pop-up menu
str=get(handles.select,'String');
val=get(handles.select,'Value');

switch val
    case 2
        s='00';
    case 3
        s='01';
    case 4
        s='10';
end


%Get user data from edit box 'azimuth'
data1=guidata(hObject);
n1=str2num(data1.s1);
b1=dec2bin(n1);
c1=char(data1.s1);
sign='0';
for i=1:numel(c1)
if (c1(i)=='.')
c2 = c1(i+1);
break;
else
    c2='0';
end
end
t2=str2num(c2);
b2=dec2bin(t2);
b=strcat(b1,b2);
e=numel(b1)-1;
ex=e+127;
exponent=dec2bin(ex);
m=b(2:numel(b));
a=blanks(23-numel(m));
n=(23-numel(m));
z=num2str(zeros(1,n));
for i=1:numel(z)
if(z(i)=='0')
a(1:(23-numel(m)))='0';
end
end
azi=strcat(sign,exponent,m,a)



%Get user data from edit box 'elevation'
data2=guidata(hObject);
n2=str2num(data2.s2);
```

```
b1=dec2bin(n2);
c1=char(data2.s2);
sign='0';
for i=1:numel(c1)
if c1(i)=='.'
c2=c1(i+1);
break;
else
    c2='0';
end
end
t2=str2num(c2);
b2=dec2bin(t2);
b=strcat(b1,b2);
e=numel(b1)-1;
ex=e+127;
exponent=dec2bin(ex);
m=b(2:numel(b));
a=blanks(23-numel(m));
n=(23-numel(m));
z=num2str(zeros(1,n));
for i=1:numel(z)
if(z(i)=='0')
a(1:(23-numel(m)))='0';
end
end
ele=strcat(sign,exponent,m,a)

%Get user data from edit box 'attenuation'
data3=guidata(hObject);
n3=str2num(data3.s3);
b1=dec2bin(n3);
c1=char(data3.s3);
sign='0';
for i=1:numel(c1)
if c1(i)=='.'
c2=c1(i+1);
break;
else
    c2='0';
end
end
t2=str2num(c2);
b2=dec2bin(t2);
b=strcat(b1,b2);
e=numel(b1)-1;
ex=e+127;
exponent=dec2bin(ex);
m=b(2:numel(b));
a=blanks(23-numel(m));
n=(23-numel(m));
z=num2str(zeros(1,n));
for i=1:numel(z)
if(z(i)=='0')
a(1:(23-numel(m)))='0';
end
end
```

```
att=strcat(sign,exponent,m,a)



%Establish Serial Communication when 'send' toggles
if((n1>=0 & n1<=180) & (n2>=0 & n2<=90) & (n3>=0 & n3<=64))
    if(state==1)


        %To configure serial port
        handles.Ser=serial('COM1','Baudrate',9600,'FlowControl','none');
        %To open the serial port
        fopen(handles.Ser)
        if(handles.Ser.Status=='open')
            set(hObject,'String','Connected');
        end



        %To transmit data through serial port
        fwrite(handles.Ser,azi)
        fwrite(handles.Ser,ele)
        fwrite(handles.Ser,att)
        fwrite(handles.Ser,s)
        fwrite(handles.Ser,st1)
        fwrite(handles.Ser,st2)
        pause(10)
        if(handles.Ser.ValuesSent~=0)
            set(handles.azi_disp,'string',azi);
            set(handles.ele_disp,'string',ele);
            set(handles.att_disp,'string',att);
        end
        handles.Ser.Timeout=40



        %To read the transmitted data from serial port
%         output=fread(handles.Ser,42)
%         pause(2)
%         handles.Ser.ValuesReceived
%         set(handles.display,'string',output);



    end

    if(state==0)
    set(hObject,'String','FEED USER DATA');
    set(handles.display,'string',' ');


    %To close the serial port before using it
```

```
    if ~isempty(instrfind)
        fclose(instrfind)
        delete(instrfind)
    end

    end

else
    if(n1<0 | n1>180)
        set(handles.azimuth,'String','Out of Range');
    end

    if(n2<0 | n2>90)
        set(handles.elevation,'String','Out of Range');
    end

    if(n3<0 | n3>64)
        set(handles.attenuation,'String','Out of Range');
    end

end


guidata(hObject,handles)



% --- Executes on button press in plot.
function selftest_Callback(hObject, eventdata, handles)
% hObject    handle to plot (see GCBO)
f = [3 3.25].*1e9;
c = 3e8;
lambda = c/f(end);
phi=[0 90 180];
theta=[0 45 70];
for a=1:3
    for b=1:3
         H = phased.URA ([4 4],'ElementSpacing',lambda/2);
         steer_ang=[phi(a);theta(b)];
         hsv = phased.SteeringVector('SensorArray',H);
         sv = step(hsv,f(end),steer_ang);
         figure;

plotResponse(H,f(end),c,'Format','Polar','RespCut','3D','Weights',sv)
    end
end




% --- Executes on selection change in display.
function display_Callback(hObject, eventdata, handles)
% hObject    handle to display (see GCBO)
```

```
% --- Executes during object creation, after setting all properties.
function display_CreateFcn(hObject, eventdata, handles)
% hObject    handle to display (see GCBO)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function azi_disp_Callback(hObject, eventdata, handles)
% hObject    handle to azi_disp (see GCBO)


% --- Executes during object creation, after setting all properties.
function azi_disp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to azi_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function ele_disp_Callback(hObject, eventdata, handles)
% hObject    handle to ele_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ele_disp as text
%        str2double(get(hObject,'String')) returns contents of ele_disp as a
double


% --- Executes during object creation, after setting all properties.
function ele_disp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ele_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function att_disp_Callback(hObject, eventdata, handles)
% hObject    handle to att_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of att_disp as text
%        str2double(get(hObject,'String')) returns contents of att_disp as a
double


% --- Executes during object creation, after setting all properties.
function att_disp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to att_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in fire.
function fire_Callback(hObject, eventdata, handles)
% hObject    handle to fire (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in plot.
function plot_Callback(hObject, eventdata, handles)
% hObject    handle to plot (see GCBO)
        %3D Array Factor of a 4x4 planar array antenna
        %Elements on the x and y axis
        %Element factor is ignored
        %Power Array factor is calculated and plotted
        %Code can be extended for larger arrays
        data1=guidata(hObject);
        n1=str2num(data1.s1);
%         s1=get(handles.azimuth,'String');
%         n1=str2num(s1);
        data2=guidata(hObject);
        n2=str2num(data2.s2);
%         s2=get(handles.elevation,'String');
%         n2=str2num(s2);
        data3=guidata(hObject);
        n3=str2num(data3.s3);
%         s3=get(handles.attenuation,'String');
%         n3=str2num(s3);
        data4=guidata(hObject);
```

```
        row=str2num(data4.s4);
%         s4=get(handles.row,'String');
%         row=str2num(s4);
        data5=guidata(hObject);
        column=str2num(data5.s5);
%         s5=get(handles.column,'String');
%         column=str2num(s5);

%



% % %         [pattern_phitheta,phi,theta] = helperPatternImport;
% % %         [pattern_azel,az,el] =
phitheta2azelpat(pattern_phitheta,phi,theta);



        %constants
        f = [3 3.25].*1e9;
        c = 3e8;
        lambda = c/f(end);
        k = 2*pi/lambda;
        dx=lambda/2; %distance between elements (X)
        dy=lambda/2;%distance between elements (Y)
        % lengtharray = 4;
        % breadtharray = 4;
        looktheta = n2;
        lookphi = n1;
        temp=10.^(n3/10);

        %% taper


        taper1 = taylorwin(row);
        taper2 = taylorwin(column);
        taper (row,column) = 0;
        for a = 1:row
            for b = 1:column
            taper (a,b) = taper1 (a).*taper2(b);
            end
        end

        figure;
        surf (1:row, 1:column, taper);box 'on';
        shading interp;
         xlabel('X axis');
        ylabel('Y axis');
        title ('Amplitude Excitation Distribution (Taylor Distribution)');
%%
       for a=1:row
           for b=1:column
               AB(a,b)=temp.*taper(a,b); % Array amplitudes
           end
       end
       for a=1:row
           for b=1:column
```

```
                AB_phase(a,b)=0;
            end
        end

         for a=1:row
             for b=1:column
                 AB_phase(a,b) = (a-(row/2))*((-
1*k)*dx*sin(looktheta*pi/180)*cos(lookphi*pi/180)) + (b-(column/2))*((-
1*k)*dy*sin(looktheta*pi/180)* sin(lookphi*pi/180));
             end
         end

        for a=1:row
             for b=1:column
                  if (AB_phase(a,b)< 0)
                   AB_phase(a,b) = mod (AB_phase(a,b),(2*pi));
                  end
             end
        end
         taper

         AB_phase

         AB_phase_deg = mod(AB_phase.*(180/pi), 360)

         AB_phase_matrix = floor(AB_phase_deg/1.4)

         AB_phase_bin = dec2bin(AB_phase_matrix,8)

         H = phased.URA ([row column],'ElementSpacing',lambda/2);
         H.Element.BackBaffled = true;
         figure;
         hplot = viewArray (H,'title','Planar Array');
         %figure;
         AB_coe=AB.*(cos(AB_phase)+1i*sin(AB_phase));

         total_power = (sum(sum(AB.^2)));

         %AF calculation
         %theta0=[(-pi/2)/50:pi/50:pi/2];
         %phi0=[-pi/100:pi/100:pi];
        theta0=[(-pi/100):pi/176:pi/2];
        phi0=[-(pi/100):pi/178:pi];
         [phi,theta]=meshgrid(phi0,theta0);
         sinU=sin(theta).*cos(phi);
         sinV=sin(theta).*sin(phi);

         AF_Field=0;
         for n=1:row
             for m=1:column
                 AF_Field = AB_coe(n,m)*exp((1i*2*pi*(n-
1)/lambda)*dx*(sinU)).*exp((1i*2*pi*(m-1)/lambda)*dy*(sinV)) + AF_Field;
             end
         end
```

```
AF_power=AF_Field.^2;
AF_power_normalized=AF_power/total_power;


%Plotting
%mesh(phi0*180/pi,theta0*180/pi,20*log10(abs(AF)))
figure;
axis([0 360 -90 90 -10 6])
surf(phi*180/pi,theta*180/pi,(abs(AF_power_normalized))), colorbar
xlabel('\phi')
ylabel('\theta')
shading interp
title ('AF Plot (MATLAB SIMULATION)');
view (-10, 15);
H.Element.BackBaffled = true;

figure;
polar (theta,abs(AF_power_normalized)),colorbar;
title ('AF Plot elevation cut');
polar (handles.plot1,theta,abs(AF_power_normalized)),colorbar;

figure;
polar (phi,abs(AF_power_normalized)),colorbar;
title ('AF Plot azimuth cut');
polar (handles.plot2,phi,abs(AF_power_normalized)),colorbar;


%%

betay = AB_phase_deg(1,2)- AB_phase_deg(1,1);
betax = AB_phase_deg(2,1)-AB_phase_deg(1,1);
n= 1:row;
m= 1:column;
AF(90,180)=0;
for phi1 = 1:180
     for theta1 = 1:90

          CHIx  = k*dx*sin (theta1*pi/180)*cos(phi1*pi/180)+
betax*(pi/180);
          CHIy = k*dy*sin (theta1*pi/180)*sin(phi1*pi/180)+
betay*(pi/180);
          AF(theta1, phi1) =
real((sum(exp(1i*n*CHIx))).*(sum(exp(1i*m*CHIy))));
          %AFn (theta1,phi1) = (abs(AF
(theta1,phi1))/max(AF(theta1,phi1)));
     end
end

figure;
surf((1:phi1), (1:theta1),abs(AF)), box on; colorbar;
title ('Verified AF Plot');
xlabel('\phi')
ylabel('\theta')
shading interp
view (-10, 15);
```

```
        size (AF_power_normalized);
        size (abs(AF));

        AF_power_normalized_1 =
AF_power_normalized./max(AF_power_normalized);
        absolute_AF = abs(AF)./max(abs(AF));


        figure;
        surf((1:phi1), (1:theta1),abs(absolute_AF-AF_power_normalized_1)),
box on; colorbar;
        title ('Error Plot wrt \theta - \phi');
        xlabel('\phi')
        ylabel('\theta')
        zlabel ('DB scale');
        shading interp
        RMSE = sqrt(sum(mean(abs(absolute_AF-
AF_power_normalized_1)).^2)/180);
        fprintf('The RMSE between the Simulated and Verified AF plot is = %f
dB', RMSE);
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

APPENDIX 'B'

VHDL CODE

1. DATA PROCESSOR BLOCK

```
----------------------------------------------------------------------------
-- Company: Ashutosh Kaushal
-- Engineer: Ashutosh Kaushal
--
-- Create Date:    01:00:03 06/20/2020
-- Design Name:
-- Module Name:    amp_phase_state - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
---
-- Module: amp_phase_state_fixpt
-- Source Path: amp_phase_state_fixpt
-- Hierarchy Level: 0
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
use ieee.std_logic_signed.all;
USE work.amp_phase_state_fixpt_pkg.ALL;

ENTITY amp_phase_state_fixpt IS
  PORT( looktheta                         :   IN    std_logic_vector(13
DOWNTO 0);  -- ufix8
```

```
        lookphi                          :   IN    std_logic_vector(13
DOWNTO 0);  -- ufix8
        f                                :   IN    std_logic_vector(34
DOWNTO 0):= "0001100000110110111000100001000000000";  -- ufix35
        attenuation                      :   IN    std_logic;  -- ufix1
             mode_selection_TX_RX_BITE              :      in
std_logic_vector (7 downto 0);
             CLK
     :      IN    STD_LOGIC;
             por
     :      in    std_logic;
             DATA_REG                                                  :
OUT std_logic_vector(39 downto 0)          -- ufix1


END amp_phase_state_fixpt;


ARCHITECTURE rtl OF amp_phase_state_fixpt IS

  -- Constants
  CONSTANT nc : vector_of_signed16(0 TO 255) :=
    (to_signed(16#7FFF#, 16), to_signed(16#7FF6#, 16), to_signed(16#7FD9#,
16), to_signed(16#7FA7#, 16),
     to_signed(16#7F62#, 16), to_signed(16#7F0A#, 16), to_signed(16#7E9D#,
16), to_signed(16#7E1E#, 16),
     to_signed(16#7D8A#, 16), to_signed(16#7CE4#, 16), to_signed(16#7C2A#,
16), to_signed(16#7B5D#, 16),
     to_signed(16#7A7D#, 16), to_signed(16#798A#, 16), to_signed(16#7885#,
16), to_signed(16#776C#, 16),
     to_signed(16#7642#, 16), to_signed(16#7505#, 16), to_signed(16#73B6#,
16), to_signed(16#7255#, 16),
     to_signed(16#70E3#, 16), to_signed(16#6F5F#, 16), to_signed(16#6DCA#,
16), to_signed(16#6C24#, 16),
     to_signed(16#6A6E#, 16), to_signed(16#68A7#, 16), to_signed(16#66D0#,
16), to_signed(16#64E9#, 16),
     to_signed(16#62F2#, 16), to_signed(16#60EC#, 16), to_signed(16#5ED7#,
16), to_signed(16#5CB4#, 16),
```

```
     to_signed(16#5A82#, 16), to_signed(16#5843#, 16), to_signed(16#55F6#,
16), to_signed(16#539B#, 16),
     to_signed(16#5134#, 16), to_signed(16#4EC0#, 16), to_signed(16#4C40#,
16), to_signed(16#49B4#, 16),
     to_signed(16#471D#, 16), to_signed(16#447B#, 16), to_signed(16#41CE#,
16), to_signed(16#3F17#, 16),
     to_signed(16#3C57#, 16), to_signed(16#398D#, 16), to_signed(16#36BA#,
16), to_signed(16#33DF#, 16),
     to_signed(16#30FC#, 16), to_signed(16#2E11#, 16), to_signed(16#2B1F#,
16), to_signed(16#2827#, 16),
     to_signed(16#2528#, 16), to_signed(16#2224#, 16), to_signed(16#1F1A#,
16), to_signed(16#1C0C#, 16),
     to_signed(16#18F9#, 16), to_signed(16#15E2#, 16), to_signed(16#12C8#,
16), to_signed(16#0FAB#, 16),
     to_signed(16#0C8C#, 16), to_signed(16#096B#, 16), to_signed(16#0648#,
16), to_signed(16#0324#, 16),
     to_signed(16#0000#, 16), to_signed(-16#0324#, 16), to_signed(-16#0648#,
16), to_signed(-16#096B#, 16),
     to_signed(-16#0C8C#, 16), to_signed(-16#0FAB#, 16), to_signed(-16#12C8#,
16), to_signed(-16#15E2#, 16),
     to_signed(-16#18F9#, 16), to_signed(-16#1C0C#, 16), to_signed(-16#1F1A#,
16), to_signed(-16#2224#, 16),
     to_signed(-16#2528#, 16), to_signed(-16#2827#, 16), to_signed(-16#2B1F#,
16), to_signed(-16#2E11#, 16),
     to_signed(-16#30FC#, 16), to_signed(-16#33DF#, 16), to_signed(-16#36BA#,
16), to_signed(-16#398D#, 16),
     to_signed(-16#3C57#, 16), to_signed(-16#3F17#, 16), to_signed(-16#41CE#,
16), to_signed(-16#447B#, 16),
     to_signed(-16#471D#, 16), to_signed(-16#49B4#, 16), to_signed(-16#4C40#,
16), to_signed(-16#4EC0#, 16),
     to_signed(-16#5134#, 16), to_signed(-16#539B#, 16), to_signed(-16#55F6#,
16), to_signed(-16#5843#, 16),
     to_signed(-16#5A82#, 16), to_signed(-16#5CB4#, 16), to_signed(-16#5ED7#,
16), to_signed(-16#60EC#, 16),
     to_signed(-16#62F2#, 16), to_signed(-16#64E9#, 16), to_signed(-16#66D0#,
16), to_signed(-16#68A7#, 16),
     to_signed(-16#6A6E#, 16), to_signed(-16#6C24#, 16), to_signed(-16#6DCA#,
16), to_signed(-16#6F5F#, 16),
```

```
     to_signed(-16#70E3#, 16), to_signed(-16#7255#, 16), to_signed(-16#73B6#,
16), to_signed(-16#7505#, 16),
     to_signed(-16#7642#, 16), to_signed(-16#776C#, 16), to_signed(-16#7885#,
16), to_signed(-16#798A#, 16),
     to_signed(-16#7A7D#, 16), to_signed(-16#7B5D#, 16), to_signed(-16#7C2A#,
16), to_signed(-16#7CE4#, 16),
     to_signed(-16#7D8A#, 16), to_signed(-16#7E1E#, 16), to_signed(-16#7E9D#,
16), to_signed(-16#7F0A#, 16),
     to_signed(-16#7F62#, 16), to_signed(-16#7FA7#, 16), to_signed(-16#7FD9#,
16), to_signed(-16#7FF6#, 16),
     to_signed(-16#7FFF#, 16), to_signed(-16#7FF6#, 16), to_signed(-16#7FD9#,
16), to_signed(-16#7FA7#, 16),
     to_signed(-16#7F62#, 16), to_signed(-16#7F0A#, 16), to_signed(-16#7E9D#,
16), to_signed(-16#7E1E#, 16),
     to_signed(-16#7D8A#, 16), to_signed(-16#7CE4#, 16), to_signed(-16#7C2A#,
16), to_signed(-16#7B5D#, 16),
     to_signed(-16#7A7D#, 16), to_signed(-16#798A#, 16), to_signed(-16#7885#,
16), to_signed(-16#776C#, 16),
     to_signed(-16#7642#, 16), to_signed(-16#7505#, 16), to_signed(-16#73B6#,
16), to_signed(-16#7255#, 16),
     to_signed(-16#70E3#, 16), to_signed(-16#6F5F#, 16), to_signed(-16#6DCA#,
16), to_signed(-16#6C24#, 16),
     to_signed(-16#6A6E#, 16), to_signed(-16#68A7#, 16), to_signed(-16#66D0#,
16), to_signed(-16#64E9#, 16),
     to_signed(-16#62F2#, 16), to_signed(-16#60EC#, 16), to_signed(-16#5ED7#,
16), to_signed(-16#5CB4#, 16),
     to_signed(-16#5A82#, 16), to_signed(-16#5843#, 16), to_signed(-16#55F6#,
16), to_signed(-16#539B#, 16),
     to_signed(-16#5134#, 16), to_signed(-16#4EC0#, 16), to_signed(-16#4C40#,
16), to_signed(-16#49B4#, 16),
     to_signed(-16#471D#, 16), to_signed(-16#447B#, 16), to_signed(-16#41CE#,
16), to_signed(-16#3F17#, 16),
     to_signed(-16#3C57#, 16), to_signed(-16#398D#, 16), to_signed(-16#36BA#,
16), to_signed(-16#33DF#, 16),
     to_signed(-16#30FC#, 16), to_signed(-16#2E11#, 16), to_signed(-16#2B1F#,
16), to_signed(-16#2827#, 16),
     to_signed(-16#2528#, 16), to_signed(-16#2224#, 16), to_signed(-16#1F1A#,
16), to_signed(-16#1C0C#, 16),
```

```
    to_signed(-16#18F9#, 16), to_signed(-16#15E2#, 16), to_signed(-16#12C8#,
16), to_signed(-16#0FAB#, 16),
    to_signed(-16#0C8C#, 16), to_signed(-16#096B#, 16), to_signed(-16#0648#,
16), to_signed(-16#0324#, 16),
    to_signed(16#0000#, 16), to_signed(16#0324#, 16), to_signed(16#0648#,
16), to_signed(16#096B#, 16),
    to_signed(16#0C8C#, 16), to_signed(16#0FAB#, 16), to_signed(16#12C8#,
16), to_signed(16#15E2#, 16),
    to_signed(16#18F9#, 16), to_signed(16#1C0C#, 16), to_signed(16#1F1A#,
16), to_signed(16#2224#, 16),
    to_signed(16#2528#, 16), to_signed(16#2827#, 16), to_signed(16#2B1F#,
16), to_signed(16#2E11#, 16),
    to_signed(16#30FC#, 16), to_signed(16#33DF#, 16), to_signed(16#36BA#,
16), to_signed(16#398D#, 16),
    to_signed(16#3C57#, 16), to_signed(16#3F17#, 16), to_signed(16#41CE#,
16), to_signed(16#447B#, 16),
    to_signed(16#471D#, 16), to_signed(16#49B4#, 16), to_signed(16#4C40#,
16), to_signed(16#4EC0#, 16),
    to_signed(16#5134#, 16), to_signed(16#539B#, 16), to_signed(16#55F6#,
16), to_signed(16#5843#, 16),
    to_signed(16#5A82#, 16), to_signed(16#5CB4#, 16), to_signed(16#5ED7#,
16), to_signed(16#60EC#, 16),
    to_signed(16#62F2#, 16), to_signed(16#64E9#, 16), to_signed(16#66D0#,
16), to_signed(16#68A7#, 16),
    to_signed(16#6A6E#, 16), to_signed(16#6C24#, 16), to_signed(16#6DCA#,
16), to_signed(16#6F5F#, 16),
    to_signed(16#70E3#, 16), to_signed(16#7255#, 16), to_signed(16#73B6#,
16), to_signed(16#7505#, 16),
    to_signed(16#7642#, 16), to_signed(16#776C#, 16), to_signed(16#7885#,
16), to_signed(16#798A#, 16),
    to_signed(16#7A7D#, 16), to_signed(16#7B5D#, 16), to_signed(16#7C2A#,
16), to_signed(16#7CE4#, 16),
    to_signed(16#7D8A#, 16), to_signed(16#7E1E#, 16), to_signed(16#7E9D#,
16), to_signed(16#7F0A#, 16),
    to_signed(16#7F62#, 16), to_signed(16#7FA7#, 16), to_signed(16#7FD9#,
16), to_signed(16#7FF6#, 16));  -- sfix16 [256]
  CONSTANT nc_0                           : vector_of_signed16(0 TO 255) :=
```

```
    (to_signed(16#0000#, 16), to_signed(16#0324#, 16), to_signed(16#0648#,
16), to_signed(16#096B#, 16),
     to_signed(16#0C8C#, 16), to_signed(16#0FAB#, 16), to_signed(16#12C8#,
16), to_signed(16#15E2#, 16),
     to_signed(16#18F9#, 16), to_signed(16#1C0C#, 16), to_signed(16#1F1A#,
16), to_signed(16#2224#, 16),
     to_signed(16#2528#, 16), to_signed(16#2827#, 16), to_signed(16#2B1F#,
16), to_signed(16#2E11#, 16),
     to_signed(16#30FC#, 16), to_signed(16#33DF#, 16), to_signed(16#36BA#,
16), to_signed(16#398D#, 16),
     to_signed(16#3C57#, 16), to_signed(16#3F17#, 16), to_signed(16#41CE#,
16), to_signed(16#447B#, 16),
     to_signed(16#471D#, 16), to_signed(16#49B4#, 16), to_signed(16#4C40#,
16), to_signed(16#4EC0#, 16),
     to_signed(16#5134#, 16), to_signed(16#539B#, 16), to_signed(16#55F6#,
16), to_signed(16#5843#, 16),
     to_signed(16#5A82#, 16), to_signed(16#5CB4#, 16), to_signed(16#5ED7#,
16), to_signed(16#60EC#, 16),
     to_signed(16#62F2#, 16), to_signed(16#64E9#, 16), to_signed(16#66D0#,
16), to_signed(16#68A7#, 16),
     to_signed(16#6A6E#, 16), to_signed(16#6C24#, 16), to_signed(16#6DCA#,
16), to_signed(16#6F5F#, 16),
     to_signed(16#70E3#, 16), to_signed(16#7255#, 16), to_signed(16#73B6#,
16), to_signed(16#7505#, 16),
     to_signed(16#7642#, 16), to_signed(16#776C#, 16), to_signed(16#7885#,
16), to_signed(16#798A#, 16),
     to_signed(16#7A7D#, 16), to_signed(16#7B5D#, 16), to_signed(16#7C2A#,
16), to_signed(16#7CE4#, 16),
     to_signed(16#7D8A#, 16), to_signed(16#7E1E#, 16), to_signed(16#7E9D#,
16), to_signed(16#7F0A#, 16),
     to_signed(16#7F62#, 16), to_signed(16#7FA7#, 16), to_signed(16#7FD9#,
16), to_signed(16#7FF6#, 16),
     to_signed(16#7FFF#, 16), to_signed(16#7FF6#, 16), to_signed(16#7FD9#,
16), to_signed(16#7FA7#, 16),
     to_signed(16#7F62#, 16), to_signed(16#7F0A#, 16), to_signed(16#7E9D#,
16), to_signed(16#7E1E#, 16),
     to_signed(16#7D8A#, 16), to_signed(16#7CE4#, 16), to_signed(16#7C2A#,
16), to_signed(16#7B5D#, 16),
```

```
     to_signed(16#7A7D#, 16), to_signed(16#798A#, 16), to_signed(16#7885#,
16), to_signed(16#776C#, 16),
     to_signed(16#7642#, 16), to_signed(16#7505#, 16), to_signed(16#73B6#,
16), to_signed(16#7255#, 16),
     to_signed(16#70E3#, 16), to_signed(16#6F5F#, 16), to_signed(16#6DCA#,
16), to_signed(16#6C24#, 16),
     to_signed(16#6A6E#, 16), to_signed(16#68A7#, 16), to_signed(16#66D0#,
16), to_signed(16#64E9#, 16),
     to_signed(16#62F2#, 16), to_signed(16#60EC#, 16), to_signed(16#5ED7#,
16), to_signed(16#5CB4#, 16),
     to_signed(16#5A82#, 16), to_signed(16#5843#, 16), to_signed(16#55F6#,
16), to_signed(16#539B#, 16),
     to_signed(16#5134#, 16), to_signed(16#4EC0#, 16), to_signed(16#4C40#,
16), to_signed(16#49B4#, 16),
     to_signed(16#471D#, 16), to_signed(16#447B#, 16), to_signed(16#41CE#,
16), to_signed(16#3F17#, 16),
     to_signed(16#3C57#, 16), to_signed(16#398D#, 16), to_signed(16#36BA#,
16), to_signed(16#33DF#, 16),
     to_signed(16#30FC#, 16), to_signed(16#2E11#, 16), to_signed(16#2B1F#,
16), to_signed(16#2827#, 16),
     to_signed(16#2528#, 16), to_signed(16#2224#, 16), to_signed(16#1F1A#,
16), to_signed(16#1C0C#, 16),
     to_signed(16#18F9#, 16), to_signed(16#15E2#, 16), to_signed(16#12C8#,
16), to_signed(16#0FAB#, 16),
     to_signed(16#0C8C#, 16), to_signed(16#096B#, 16), to_signed(16#0648#,
16), to_signed(16#0324#, 16),
     to_signed(16#0000#, 16), to_signed(-16#0324#, 16), to_signed(-16#0648#,
16), to_signed(-16#096B#, 16),
     to_signed(-16#0C8C#, 16), to_signed(-16#0FAB#, 16), to_signed(-16#12C8#,
16), to_signed(-16#15E2#, 16),
     to_signed(-16#18F9#, 16), to_signed(-16#1C0C#, 16), to_signed(-16#1F1A#,
16), to_signed(-16#2224#, 16),
     to_signed(-16#2528#, 16), to_signed(-16#2827#, 16), to_signed(-16#2B1F#,
16), to_signed(-16#2E11#, 16),
     to_signed(-16#30FC#, 16), to_signed(-16#33DF#, 16), to_signed(-16#36BA#,
16), to_signed(-16#398D#, 16),
     to_signed(-16#3C57#, 16), to_signed(-16#3F17#, 16), to_signed(-16#41CE#,
16), to_signed(-16#447B#, 16),
```

```
     to_signed(-16#471D#, 16), to_signed(-16#49B4#, 16), to_signed(-16#4C40#,
16), to_signed(-16#4EC0#, 16),
     to_signed(-16#5134#, 16), to_signed(-16#539B#, 16), to_signed(-16#55F6#,
16), to_signed(-16#5843#, 16),
     to_signed(-16#5A82#, 16), to_signed(-16#5CB4#, 16), to_signed(-16#5ED7#,
16), to_signed(-16#60EC#, 16),
     to_signed(-16#62F2#, 16), to_signed(-16#64E9#, 16), to_signed(-16#66D0#,
16), to_signed(-16#68A7#, 16),
     to_signed(-16#6A6E#, 16), to_signed(-16#6C24#, 16), to_signed(-16#6DCA#,
16), to_signed(-16#6F5F#, 16),
     to_signed(-16#70E3#, 16), to_signed(-16#7255#, 16), to_signed(-16#73B6#,
16), to_signed(-16#7505#, 16),
     to_signed(-16#7642#, 16), to_signed(-16#776C#, 16), to_signed(-16#7885#,
16), to_signed(-16#798A#, 16),
     to_signed(-16#7A7D#, 16), to_signed(-16#7B5D#, 16), to_signed(-16#7C2A#,
16), to_signed(-16#7CE4#, 16),
     to_signed(-16#7D8A#, 16), to_signed(-16#7E1E#, 16), to_signed(-16#7E9D#,
16), to_signed(-16#7F0A#, 16),
     to_signed(-16#7F62#, 16), to_signed(-16#7FA7#, 16), to_signed(-16#7FD9#,
16), to_signed(-16#7FF6#, 16),
     to_signed(-16#7FFF#, 16), to_signed(-16#7FF6#, 16), to_signed(-16#7FD9#,
16), to_signed(-16#7FA7#, 16),
     to_signed(-16#7F62#, 16), to_signed(-16#7F0A#, 16), to_signed(-16#7E9D#,
16), to_signed(-16#7E1E#, 16),
     to_signed(-16#7D8A#, 16), to_signed(-16#7CE4#, 16), to_signed(-16#7C2A#,
16), to_signed(-16#7B5D#, 16),
     to_signed(-16#7A7D#, 16), to_signed(-16#798A#, 16), to_signed(-16#7885#,
16), to_signed(-16#776C#, 16),
     to_signed(-16#7642#, 16), to_signed(-16#7505#, 16), to_signed(-16#73B6#,
16), to_signed(-16#7255#, 16),
     to_signed(-16#70E3#, 16), to_signed(-16#6F5F#, 16), to_signed(-16#6DCA#,
16), to_signed(-16#6C24#, 16),
     to_signed(-16#6A6E#, 16), to_signed(-16#68A7#, 16), to_signed(-16#66D0#,
16), to_signed(-16#64E9#, 16),
     to_signed(-16#62F2#, 16), to_signed(-16#60EC#, 16), to_signed(-16#5ED7#,
16), to_signed(-16#5CB4#, 16),
     to_signed(-16#5A82#, 16), to_signed(-16#5843#, 16), to_signed(-16#55F6#,
16), to_signed(-16#539B#, 16),
```

```
     to_signed(-16#5134#, 16), to_signed(-16#4EC0#, 16), to_signed(-16#4C40#,
16), to_signed(-16#49B4#, 16),
     to_signed(-16#471D#, 16), to_signed(-16#447B#, 16), to_signed(-16#41CE#,
16), to_signed(-16#3F17#, 16),
     to_signed(-16#3C57#, 16), to_signed(-16#398D#, 16), to_signed(-16#36BA#,
16), to_signed(-16#33DF#, 16),
     to_signed(-16#30FC#, 16), to_signed(-16#2E11#, 16), to_signed(-16#2B1F#,
16), to_signed(-16#2827#, 16),
     to_signed(-16#2528#, 16), to_signed(-16#2224#, 16), to_signed(-16#1F1A#,
16), to_signed(-16#1C0C#, 16),
     to_signed(-16#18F9#, 16), to_signed(-16#15E2#, 16), to_signed(-16#12C8#,
16), to_signed(-16#0FAB#, 16),
     to_signed(-16#0C8C#, 16), to_signed(-16#096B#, 16), to_signed(-16#0648#,
16), to_signed(-16#0324#, 16));  -- sfix16 [256]
  CONSTANT One                            : unsigned(46 DOWNTO 0) :=
    unsigned'("11001001000011000000000000000000000000000000000000");  -- ufix47
  CONSTANT C_divbyzero_p                  : unsigned(46 DOWNTO 0) :=
    unsigned'("11111111111111111111111111111111111111111111111111");  -- ufix47
  CONSTANT one_0                          : unsigned(52 DOWNTO 0) :=
    unsigned'("1000111100001101000110000000000000000000000000000000000");  --
ufix53
  CONSTANT c_divbyzero_p_0                : unsigned(52 DOWNTO 0) :=
    unsigned'("1111111111111111111111111111111111111111111111111111");  --
ufix53

  -- Signals

  signal clk_25MHZ,clki_25mhz                       : std_logic;
  signal starttogglecounter                         : std_logic_vector(3
downto 0) :="1111";
  SIGNAL looktheta_unsigned               : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL lookphi_unsigned                 : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL f_unsigned                       : unsigned(34 DOWNTO 0);  -- ufix35
  SIGNAL AB1_tmp                          : unsigned(5 DOWNTO 0);  -- ufix6
  SIGNAL AB2_tmp                          : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB3_tmp                          : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB4_tmp                          : unsigned(5 DOWNTO 0);  -- ufix6
  SIGNAL AB5_tmp                          : unsigned(6 DOWNTO 0);  -- ufix7
```

```
  SIGNAL AB6_tmp                          : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB7_tmp                          : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB8_tmp                          : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB9_tmp                          : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB10_tmp                         : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB11_tmp                         : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB12_tmp                         : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB13_tmp                         : unsigned(5 DOWNTO 0);  -- ufix6
  SIGNAL AB14_tmp                         : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB15_tmp                         : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB16_tmp                         : unsigned(5 DOWNTO 0);  -- ufix6
  SIGNAL AB1_phase_tmp                    : unsigned(7 DOWNTO 0):=
"00000000";  -- ufix7
  SIGNAL AB2_phase_tmp                    : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB3_phase_tmp                    : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB4_phase_tmp                    : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB5_phase_tmp                    : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB6_phase_tmp                    : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB7_phase_tmp                    : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB8_phase_tmp                    : unsigned(6 DOWNTO 0);  -- ufix7
  SIGNAL AB9_phase_tmp                    : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB10_phase_tmp                   : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB11_phase_tmp                   : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB12_phase_tmp                   : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB13_phase_tmp                   : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB14_phase_tmp                   : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB15_phase_tmp                   : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AB16_phase_tmp                   : unsigned(7 DOWNTO 0);  -- ufix8

BEGIN
  looktheta_unsigned <= unsigned(looktheta (7 downto 0));

  lookphi_unsigned <= unsigned(lookphi(7 downto 0));

  f_unsigned <= unsigned(f);

  amp_phase_state_fixpt_1_output : PROCESS (looktheta_unsigned,
lookphi_unsigned, f_unsigned, attenuation)
```

```
VARIABLE FI_SIN_COS_LUT : vector_of_signed16(0 TO 255);
VARIABLE fi_sin_cos_lut_0 : vector_of_signed16(0 TO 255);
VARIABLE tmp : unsigned(63 DOWNTO 0);
VARIABLE lambda : unsigned(13 DOWNTO 0);
VARIABLE tmp_0 : unsigned(37 DOWNTO 0);
VARIABLE k : unsigned(13 DOWNTO 0);
VARIABLE x : unsigned(35 DOWNTO 0);
VARIABLE fullScaleIndex : unsigned(15 DOWNTO 0);
VARIABLE idxLUTLoZero : unsigned(7 DOWNTO 0);
VARIABLE c1 : signed(15 DOWNTO 0);
VARIABLE x_0 : unsigned(35 DOWNTO 0);
VARIABLE fullscaleindex_0 : unsigned(15 DOWNTO 0);
VARIABLE idxlutlozero_0 : unsigned(7 DOWNTO 0);
VARIABLE betax : signed(13 DOWNTO 0);
VARIABLE x_1 : unsigned(35 DOWNTO 0);
VARIABLE fullscaleindex_1 : unsigned(15 DOWNTO 0);
VARIABLE idxlutlozero_1 : unsigned(7 DOWNTO 0);
VARIABLE c1_0 : signed(15 DOWNTO 0);
VARIABLE x_2 : unsigned(35 DOWNTO 0);
VARIABLE fullscaleindex_2 : unsigned(15 DOWNTO 0);
VARIABLE idxlutlozero_2 : unsigned(7 DOWNTO 0);
VARIABLE betay : signed(13 DOWNTO 0);
VARIABLE AB3_rad : signed(13 DOWNTO 0);
VARIABLE AB4_rad : signed(13 DOWNTO 0);
VARIABLE AB6_rad : signed(13 DOWNTO 0);
VARIABLE AB7_rad : signed(13 DOWNTO 0);
VARIABLE AB8_rad : signed(13 DOWNTO 0);
VARIABLE AB9_rad : signed(13 DOWNTO 0);
VARIABLE AB10_rad : signed(13 DOWNTO 0);
VARIABLE AB11_rad : signed(13 DOWNTO 0);
VARIABLE AB12_rad : signed(13 DOWNTO 0);
VARIABLE AB13_rad : signed(13 DOWNTO 0);
VARIABLE AB14_rad : signed(13 DOWNTO 0);
VARIABLE AB15_rad : signed(13 DOWNTO 0);
VARIABLE AB16_rad : signed(13 DOWNTO 0);
VARIABLE tmp_1 : unsigned(13 DOWNTO 0);
VARIABLE AB_deg : unsigned(13 DOWNTO 0);
VARIABLE tmp_2 : unsigned(13 DOWNTO 0);
```

```
VARIABLE ab_deg_0 : unsigned(13 DOWNTO 0);
VARIABLE tmp_3 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_1 : unsigned(13 DOWNTO 0);
VARIABLE tmp_4 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_2 : unsigned(13 DOWNTO 0);
VARIABLE tmp_5 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_3 : unsigned(13 DOWNTO 0);
VARIABLE tmp_6 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_4 : unsigned(13 DOWNTO 0);
VARIABLE tmp_7 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_5 : unsigned(13 DOWNTO 0);
VARIABLE tmp_8 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_6 : unsigned(13 DOWNTO 0);
VARIABLE tmp_9 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_7 : unsigned(13 DOWNTO 0);
VARIABLE tmp_10 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_8 : unsigned(13 DOWNTO 0);
VARIABLE tmp_11 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_9 : unsigned(13 DOWNTO 0);
VARIABLE tmp_12 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_10 : unsigned(13 DOWNTO 0);
VARIABLE tmp_13 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_11 : unsigned(13 DOWNTO 0);
VARIABLE tmp_14 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_12 : unsigned(13 DOWNTO 0);
VARIABLE tmp_15 : unsigned(13 DOWNTO 0);
VARIABLE ab_deg_13 : unsigned(13 DOWNTO 0);
VARIABLE div_temp : unsigned(52 DOWNTO 0);
VARIABLE div_temp_0 : unsigned(46 DOWNTO 0);
VARIABLE div_temp_1 : unsigned(64 DOWNTO 0);
VARIABLE div_temp_2 : unsigned(64 DOWNTO 0);
VARIABLE div_temp_3 : unsigned(64 DOWNTO 0);
VARIABLE div_temp_4 : unsigned(64 DOWNTO 0);
VARIABLE mul_temp : unsigned(21 DOWNTO 0);
VARIABLE cast : unsigned(64 DOWNTO 0);
VARIABLE cast_0 : unsigned(35 DOWNTO 0);
VARIABLE cast_1 : unsigned(2 DOWNTO 0);
VARIABLE mul_temp_0 : unsigned(34 DOWNTO 0);
```

```
VARIABLE sub_cast : unsigned(35 DOWNTO 0);
VARIABLE sub_temp : unsigned(35 DOWNTO 0);
VARIABLE cast_2 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_1 : unsigned(47 DOWNTO 0);
VARIABLE cast_3 : unsigned(31 DOWNTO 0);
VARIABLE add_cast : signed(15 DOWNTO 0);
VARIABLE mul_temp_2 : unsigned(21 DOWNTO 0);
VARIABLE cast_4 : unsigned(64 DOWNTO 0);
VARIABLE cast_5 : unsigned(35 DOWNTO 0);
VARIABLE cast_6 : unsigned(2 DOWNTO 0);
VARIABLE mul_temp_3 : unsigned(34 DOWNTO 0);
VARIABLE sub_cast_0 : unsigned(35 DOWNTO 0);
VARIABLE sub_temp_0 : unsigned(35 DOWNTO 0);
VARIABLE cast_7 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_4 : unsigned(47 DOWNTO 0);
VARIABLE cast_8 : unsigned(31 DOWNTO 0);
VARIABLE cast_9 : signed(14 DOWNTO 0);
VARIABLE cast_10 : signed(14 DOWNTO 0);
VARIABLE cast_11 : unsigned(14 DOWNTO 0);
VARIABLE cast_12 : unsigned(14 DOWNTO 0);
VARIABLE cast_13 : unsigned(13 DOWNTO 0);
VARIABLE cast_14 : signed(14 DOWNTO 0);
VARIABLE mul_temp_5 : signed(29 DOWNTO 0);
VARIABLE add_cast_0 : signed(31 DOWNTO 0);
VARIABLE cast_15 : signed(8 DOWNTO 0);
VARIABLE add_temp : unsigned(7 DOWNTO 0);
VARIABLE add_cast_1 : signed(15 DOWNTO 0);
VARIABLE sub_cast_1 : signed(31 DOWNTO 0);
VARIABLE sub_cast_2 : signed(31 DOWNTO 0);
VARIABLE sub_temp_1 : signed(31 DOWNTO 0);
VARIABLE mul_temp_6 : signed(40 DOWNTO 0);
VARIABLE add_cast_2 : signed(31 DOWNTO 0);
VARIABLE add_cast_3 : signed(15 DOWNTO 0);
VARIABLE add_cast_4 : signed(31 DOWNTO 0);
VARIABLE add_temp_0 : signed(31 DOWNTO 0);
VARIABLE cast_16 : signed(15 DOWNTO 0);
VARIABLE mul_temp_7 : signed(45 DOWNTO 0);
VARIABLE add_cast_5 : signed(31 DOWNTO 0);
```

```
VARIABLE cast_17 : signed(8 DOWNTO 0);
VARIABLE add_temp_1 : unsigned(7 DOWNTO 0);
VARIABLE add_cast_6 : signed(15 DOWNTO 0);
VARIABLE sub_cast_3 : signed(31 DOWNTO 0);
VARIABLE sub_cast_4 : signed(31 DOWNTO 0);
VARIABLE sub_temp_2 : signed(31 DOWNTO 0);
VARIABLE mul_temp_8 : signed(40 DOWNTO 0);
VARIABLE add_cast_7 : signed(31 DOWNTO 0);
VARIABLE add_cast_8 : signed(15 DOWNTO 0);
VARIABLE add_cast_9 : signed(31 DOWNTO 0);
VARIABLE add_temp_2 : signed(31 DOWNTO 0);
VARIABLE cast_18 : signed(15 DOWNTO 0);
VARIABLE mul_temp_9 : signed(61 DOWNTO 0);
VARIABLE mul_temp_10 : unsigned(21 DOWNTO 0);
VARIABLE cast_19 : unsigned(64 DOWNTO 0);
VARIABLE cast_20 : unsigned(35 DOWNTO 0);
VARIABLE cast_21 : unsigned(2 DOWNTO 0);
VARIABLE mul_temp_11 : unsigned(34 DOWNTO 0);
VARIABLE sub_cast_5 : unsigned(35 DOWNTO 0);
VARIABLE sub_temp_3 : unsigned(35 DOWNTO 0);
VARIABLE cast_22 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_12 : unsigned(47 DOWNTO 0);
VARIABLE cast_23 : unsigned(31 DOWNTO 0);
VARIABLE add_cast_10 : signed(15 DOWNTO 0);
VARIABLE mul_temp_13 : unsigned(21 DOWNTO 0);
VARIABLE cast_24 : unsigned(64 DOWNTO 0);
VARIABLE cast_25 : unsigned(35 DOWNTO 0);
VARIABLE cast_26 : unsigned(2 DOWNTO 0);
VARIABLE mul_temp_14 : unsigned(34 DOWNTO 0);
VARIABLE sub_cast_6 : unsigned(35 DOWNTO 0);
VARIABLE sub_temp_4 : unsigned(35 DOWNTO 0);
VARIABLE cast_27 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_15 : unsigned(47 DOWNTO 0);
VARIABLE cast_28 : unsigned(31 DOWNTO 0);
VARIABLE cast_29 : signed(14 DOWNTO 0);
VARIABLE cast_30 : signed(14 DOWNTO 0);
VARIABLE cast_31 : unsigned(14 DOWNTO 0);
VARIABLE cast_32 : unsigned(14 DOWNTO 0);
```

```
VARIABLE cast_33 : unsigned(13 DOWNTO 0);
VARIABLE cast_34 : signed(14 DOWNTO 0);
VARIABLE mul_temp_16 : signed(29 DOWNTO 0);
VARIABLE add_cast_11 : signed(31 DOWNTO 0);
VARIABLE cast_35 : signed(8 DOWNTO 0);
VARIABLE add_temp_3 : unsigned(7 DOWNTO 0);
VARIABLE add_cast_12 : signed(15 DOWNTO 0);
VARIABLE sub_cast_7 : signed(31 DOWNTO 0);
VARIABLE sub_cast_8 : signed(31 DOWNTO 0);
VARIABLE sub_temp_5 : signed(31 DOWNTO 0);
VARIABLE mul_temp_17 : signed(40 DOWNTO 0);
VARIABLE add_cast_13 : signed(31 DOWNTO 0);
VARIABLE add_cast_14 : signed(15 DOWNTO 0);
VARIABLE add_cast_15 : signed(31 DOWNTO 0);
VARIABLE add_temp_4 : signed(31 DOWNTO 0);
VARIABLE cast_36 : signed(15 DOWNTO 0);
VARIABLE mul_temp_18 : signed(45 DOWNTO 0);
VARIABLE add_cast_16 : signed(31 DOWNTO 0);
VARIABLE cast_37 : signed(8 DOWNTO 0);
VARIABLE add_temp_5 : unsigned(7 DOWNTO 0);
VARIABLE add_cast_17 : signed(15 DOWNTO 0);
VARIABLE sub_cast_9 : signed(31 DOWNTO 0);
VARIABLE sub_cast_10 : signed(31 DOWNTO 0);
VARIABLE sub_temp_6 : signed(31 DOWNTO 0);
VARIABLE mul_temp_19 : signed(40 DOWNTO 0);
VARIABLE add_cast_18 : signed(31 DOWNTO 0);
VARIABLE add_cast_19 : signed(15 DOWNTO 0);
VARIABLE add_cast_20 : signed(31 DOWNTO 0);
VARIABLE add_temp_6 : signed(31 DOWNTO 0);
VARIABLE cast_38 : signed(15 DOWNTO 0);
VARIABLE mul_temp_20 : signed(61 DOWNTO 0);
VARIABLE cast_39 : signed(16 DOWNTO 0);
VARIABLE cast_40 : signed(15 DOWNTO 0);
VARIABLE mul_temp_21 : signed(16 DOWNTO 0);
VARIABLE cast_41 : signed(15 DOWNTO 0);
VARIABLE add_cast_21 : signed(14 DOWNTO 0);
VARIABLE add_cast_22 : signed(14 DOWNTO 0);
VARIABLE add_temp_7 : signed(14 DOWNTO 0);
```

```
VARIABLE add_cast_23 : signed(16 DOWNTO 0);
VARIABLE add_cast_24 : signed(15 DOWNTO 0);
VARIABLE add_cast_25 : signed(16 DOWNTO 0);
VARIABLE add_cast_26 : signed(16 DOWNTO 0);
VARIABLE add_temp_8 : signed(16 DOWNTO 0);
VARIABLE mul_temp_22 : signed(16 DOWNTO 0);
VARIABLE add_cast_27 : signed(15 DOWNTO 0);
VARIABLE add_cast_28 : signed(16 DOWNTO 0);
VARIABLE add_cast_29 : signed(16 DOWNTO 0);
VARIABLE add_temp_9 : signed(16 DOWNTO 0);
VARIABLE cast_42 : signed(16 DOWNTO 0);
VARIABLE cast_43 : signed(15 DOWNTO 0);
VARIABLE add_cast_30 : signed(16 DOWNTO 0);
VARIABLE add_cast_31 : signed(16 DOWNTO 0);
VARIABLE add_cast_32 : signed(15 DOWNTO 0);
VARIABLE add_cast_33 : signed(16 DOWNTO 0);
VARIABLE add_temp_10 : signed(16 DOWNTO 0);
VARIABLE add_cast_34 : signed(16 DOWNTO 0);
VARIABLE add_cast_35 : signed(15 DOWNTO 0);
VARIABLE add_cast_36 : signed(16 DOWNTO 0);
VARIABLE add_cast_37 : signed(16 DOWNTO 0);
VARIABLE add_cast_38 : signed(15 DOWNTO 0);
VARIABLE add_cast_39 : signed(16 DOWNTO 0);
VARIABLE add_temp_11 : signed(16 DOWNTO 0);
VARIABLE mul_temp_23 : signed(16 DOWNTO 0);
VARIABLE add_cast_40 : signed(15 DOWNTO 0);
VARIABLE add_cast_41 : signed(16 DOWNTO 0);
VARIABLE add_cast_42 : signed(16 DOWNTO 0);
VARIABLE add_cast_43 : signed(15 DOWNTO 0);
VARIABLE add_cast_44 : signed(16 DOWNTO 0);
VARIABLE add_temp_12 : signed(16 DOWNTO 0);
VARIABLE mul_temp_24 : signed(16 DOWNTO 0);
VARIABLE cast_44 : signed(15 DOWNTO 0);
VARIABLE add_cast_45 : signed(16 DOWNTO 0);
VARIABLE mul_temp_25 : signed(16 DOWNTO 0);
VARIABLE add_cast_46 : signed(15 DOWNTO 0);
VARIABLE add_cast_47 : signed(16 DOWNTO 0);
VARIABLE add_temp_13 : signed(16 DOWNTO 0);
```

```
VARIABLE add_cast_48 : signed(16 DOWNTO 0);
VARIABLE add_cast_49 : signed(15 DOWNTO 0);
VARIABLE add_cast_50 : signed(16 DOWNTO 0);
VARIABLE mul_temp_26 : signed(16 DOWNTO 0);
VARIABLE add_cast_51 : signed(15 DOWNTO 0);
VARIABLE add_cast_52 : signed(16 DOWNTO 0);
VARIABLE add_temp_14 : signed(16 DOWNTO 0);
VARIABLE mul_temp_27 : signed(16 DOWNTO 0);
VARIABLE add_cast_53 : signed(15 DOWNTO 0);
VARIABLE add_cast_54 : signed(16 DOWNTO 0);
VARIABLE mul_temp_28 : signed(16 DOWNTO 0);
VARIABLE add_cast_55 : signed(15 DOWNTO 0);
VARIABLE add_cast_56 : signed(16 DOWNTO 0);
VARIABLE add_temp_15 : signed(16 DOWNTO 0);
VARIABLE cast_45 : signed(14 DOWNTO 0);
VARIABLE cast_46 : signed(14 DOWNTO 0);
VARIABLE cast_47 : signed(15 DOWNTO 0);
VARIABLE mul_temp_29 : unsigned(27 DOWNTO 0);
VARIABLE cast_48 : signed(14 DOWNTO 0);
VARIABLE cast_49 : signed(14 DOWNTO 0);
VARIABLE cast_50 : signed(15 DOWNTO 0);
VARIABLE mul_temp_30 : unsigned(27 DOWNTO 0);
VARIABLE cast_51 : signed(14 DOWNTO 0);
VARIABLE cast_52 : signed(14 DOWNTO 0);
VARIABLE cast_53 : signed(15 DOWNTO 0);
VARIABLE mul_temp_31 : unsigned(27 DOWNTO 0);
VARIABLE cast_54 : signed(14 DOWNTO 0);
VARIABLE cast_55 : signed(14 DOWNTO 0);
VARIABLE cast_56 : signed(15 DOWNTO 0);
VARIABLE mul_temp_32 : unsigned(27 DOWNTO 0);
VARIABLE cast_57 : signed(14 DOWNTO 0);
VARIABLE cast_58 : signed(14 DOWNTO 0);
VARIABLE cast_59 : signed(15 DOWNTO 0);
VARIABLE mul_temp_33 : unsigned(27 DOWNTO 0);
VARIABLE cast_60 : signed(14 DOWNTO 0);
VARIABLE cast_61 : signed(14 DOWNTO 0);
VARIABLE cast_62 : signed(15 DOWNTO 0);
VARIABLE mul_temp_34 : unsigned(27 DOWNTO 0);
```

```
VARIABLE cast_63 : signed(14 DOWNTO 0);
VARIABLE cast_64 : signed(14 DOWNTO 0);
VARIABLE cast_65 : signed(15 DOWNTO 0);
VARIABLE mul_temp_35 : unsigned(27 DOWNTO 0);
VARIABLE cast_66 : signed(14 DOWNTO 0);
VARIABLE cast_67 : signed(14 DOWNTO 0);
VARIABLE cast_68 : signed(15 DOWNTO 0);
VARIABLE mul_temp_36 : unsigned(27 DOWNTO 0);
VARIABLE cast_69 : signed(14 DOWNTO 0);
VARIABLE cast_70 : signed(14 DOWNTO 0);
VARIABLE cast_71 : signed(15 DOWNTO 0);
VARIABLE mul_temp_37 : unsigned(27 DOWNTO 0);
VARIABLE cast_72 : signed(14 DOWNTO 0);
VARIABLE cast_73 : signed(14 DOWNTO 0);
VARIABLE cast_74 : signed(15 DOWNTO 0);
VARIABLE mul_temp_38 : unsigned(27 DOWNTO 0);
VARIABLE cast_75 : signed(14 DOWNTO 0);
VARIABLE cast_76 : signed(14 DOWNTO 0);
VARIABLE cast_77 : signed(15 DOWNTO 0);
VARIABLE mul_temp_39 : unsigned(27 DOWNTO 0);
VARIABLE cast_78 : signed(14 DOWNTO 0);
VARIABLE cast_79 : signed(14 DOWNTO 0);
VARIABLE cast_80 : signed(15 DOWNTO 0);
VARIABLE mul_temp_40 : unsigned(27 DOWNTO 0);
VARIABLE cast_81 : signed(14 DOWNTO 0);
VARIABLE cast_82 : signed(14 DOWNTO 0);
VARIABLE cast_83 : signed(15 DOWNTO 0);
VARIABLE mul_temp_41 : unsigned(27 DOWNTO 0);
VARIABLE cast_84 : signed(14 DOWNTO 0);
VARIABLE cast_85 : signed(14 DOWNTO 0);
VARIABLE cast_86 : signed(15 DOWNTO 0);
VARIABLE mul_temp_42 : unsigned(27 DOWNTO 0);
VARIABLE cast_87 : signed(14 DOWNTO 0);
VARIABLE cast_88 : signed(14 DOWNTO 0);
VARIABLE cast_89 : signed(15 DOWNTO 0);
VARIABLE mul_temp_43 : unsigned(27 DOWNTO 0);
VARIABLE sub_cast_11 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_44 : unsigned(27 DOWNTO 0);
```

```
VARIABLE cast_90 : std_logic;
VARIABLE sub_cast_12 : unsigned(15 DOWNTO 0);
VARIABLE cast_91 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_7 : unsigned(15 DOWNTO 0);
VARIABLE cast_92 : unsigned(10 DOWNTO 0);
VARIABLE cast_93 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_45 : unsigned(22 DOWNTO 0);
VARIABLE cast_94 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_13 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_46 : unsigned(27 DOWNTO 0);
VARIABLE cast_95 : std_logic;
VARIABLE sub_cast_14 : unsigned(15 DOWNTO 0);
VARIABLE cast_96 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_8 : unsigned(15 DOWNTO 0);
VARIABLE cast_97 : unsigned(10 DOWNTO 0);
VARIABLE cast_98 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_47 : unsigned(22 DOWNTO 0);
VARIABLE cast_99 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_15 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_48 : unsigned(27 DOWNTO 0);
VARIABLE cast_100 : std_logic;
VARIABLE sub_cast_16 : unsigned(15 DOWNTO 0);
VARIABLE cast_101 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_9 : unsigned(15 DOWNTO 0);
VARIABLE cast_102 : unsigned(10 DOWNTO 0);
VARIABLE cast_103 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_49 : unsigned(22 DOWNTO 0);
VARIABLE cast_104 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_17 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_50 : unsigned(27 DOWNTO 0);
VARIABLE cast_105 : std_logic;
VARIABLE sub_cast_18 : unsigned(15 DOWNTO 0);
VARIABLE cast_106 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_10 : unsigned(15 DOWNTO 0);
VARIABLE cast_107 : unsigned(10 DOWNTO 0);
VARIABLE cast_108 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_51 : unsigned(22 DOWNTO 0);
VARIABLE cast_109 : unsigned(8 DOWNTO 0);
```

```
VARIABLE sub_cast_19 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_52 : unsigned(27 DOWNTO 0);
VARIABLE cast_110 : std_logic;
VARIABLE sub_cast_20 : unsigned(15 DOWNTO 0);
VARIABLE cast_111 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_11 : unsigned(15 DOWNTO 0);
VARIABLE cast_112 : unsigned(10 DOWNTO 0);
VARIABLE cast_113 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_53 : unsigned(22 DOWNTO 0);
VARIABLE cast_114 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_21 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_54 : unsigned(27 DOWNTO 0);
VARIABLE cast_115 : std_logic;
VARIABLE sub_cast_22 : unsigned(15 DOWNTO 0);
VARIABLE cast_116 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_12 : unsigned(15 DOWNTO 0);
VARIABLE cast_117 : unsigned(10 DOWNTO 0);
VARIABLE cast_118 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_55 : unsigned(22 DOWNTO 0);
VARIABLE cast_119 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_23 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_56 : unsigned(27 DOWNTO 0);
VARIABLE cast_120 : std_logic;
VARIABLE sub_cast_24 : unsigned(15 DOWNTO 0);
VARIABLE cast_121 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_13 : unsigned(15 DOWNTO 0);
VARIABLE cast_122 : unsigned(10 DOWNTO 0);
VARIABLE cast_123 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_57 : unsigned(22 DOWNTO 0);
VARIABLE cast_124 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_25 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_58 : unsigned(27 DOWNTO 0);
VARIABLE cast_125 : std_logic;
VARIABLE sub_cast_26 : unsigned(15 DOWNTO 0);
VARIABLE cast_126 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_14 : unsigned(15 DOWNTO 0);
VARIABLE cast_127 : unsigned(10 DOWNTO 0);
VARIABLE cast_128 : unsigned(8 DOWNTO 0);
```

```
VARIABLE mul_temp_59 : unsigned(22 DOWNTO 0);
VARIABLE cast_129 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_27 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_60 : unsigned(27 DOWNTO 0);
VARIABLE cast_130 : std_logic;
VARIABLE sub_cast_28 : unsigned(15 DOWNTO 0);
VARIABLE cast_131 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_15 : unsigned(15 DOWNTO 0);
VARIABLE cast_132 : unsigned(10 DOWNTO 0);
VARIABLE cast_133 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_61 : unsigned(22 DOWNTO 0);
VARIABLE cast_134 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_29 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_62 : unsigned(27 DOWNTO 0);
VARIABLE cast_135 : std_logic;
VARIABLE sub_cast_30 : unsigned(15 DOWNTO 0);
VARIABLE cast_136 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_16 : unsigned(15 DOWNTO 0);
VARIABLE cast_137 : unsigned(10 DOWNTO 0);
VARIABLE cast_138 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_63 : unsigned(22 DOWNTO 0);
VARIABLE cast_139 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_31 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_64 : unsigned(27 DOWNTO 0);
VARIABLE cast_140 : std_logic;
VARIABLE sub_cast_32 : unsigned(15 DOWNTO 0);
VARIABLE cast_141 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_17 : unsigned(15 DOWNTO 0);
VARIABLE cast_142 : unsigned(10 DOWNTO 0);
VARIABLE cast_143 : unsigned(8 DOWNTO 0);
VARIABLE mul_temp_65 : unsigned(22 DOWNTO 0);
VARIABLE cast_144 : unsigned(8 DOWNTO 0);
VARIABLE sub_cast_33 : unsigned(15 DOWNTO 0);
VARIABLE mul_temp_66 : unsigned(27 DOWNTO 0);
VARIABLE cast_145 : std_logic;
VARIABLE sub_cast_34 : unsigned(15 DOWNTO 0);
VARIABLE cast_146 : unsigned(9 DOWNTO 0);
VARIABLE sub_temp_18 : unsigned(15 DOWNTO 0);
```

```
  VARIABLE cast_147 : unsigned(10 DOWNTO 0);
  VARIABLE cast_148 : unsigned(8 DOWNTO 0);
  VARIABLE mul_temp_67 : unsigned(22 DOWNTO 0);
  VARIABLE cast_149 : unsigned(8 DOWNTO 0);
  VARIABLE sub_cast_35 : unsigned(15 DOWNTO 0);
  VARIABLE mul_temp_68 : unsigned(27 DOWNTO 0);
  VARIABLE cast_150 : std_logic;
  VARIABLE sub_cast_36 : unsigned(15 DOWNTO 0);
  VARIABLE cast_151 : unsigned(9 DOWNTO 0);
  VARIABLE sub_temp_19 : unsigned(15 DOWNTO 0);
  VARIABLE cast_152 : unsigned(10 DOWNTO 0);
  VARIABLE cast_153 : unsigned(8 DOWNTO 0);
  VARIABLE mul_temp_69 : unsigned(22 DOWNTO 0);
  VARIABLE cast_154 : unsigned(8 DOWNTO 0);
  VARIABLE sub_cast_37 : unsigned(15 DOWNTO 0);
  VARIABLE mul_temp_70 : unsigned(27 DOWNTO 0);
  VARIABLE cast_155 : std_logic;
  VARIABLE sub_cast_38 : unsigned(15 DOWNTO 0);
  VARIABLE cast_156 : unsigned(9 DOWNTO 0);
  VARIABLE sub_temp_20 : unsigned(15 DOWNTO 0);
  VARIABLE cast_157 : unsigned(10 DOWNTO 0);
  VARIABLE cast_158 : unsigned(8 DOWNTO 0);
  VARIABLE mul_temp_71 : unsigned(22 DOWNTO 0);
  VARIABLE cast_159 : unsigned(8 DOWNTO 0);
  VARIABLE sub_cast_39 : unsigned(15 DOWNTO 0);
  VARIABLE mul_temp_72 : unsigned(27 DOWNTO 0);
  VARIABLE cast_160 : std_logic;
  VARIABLE sub_cast_40 : unsigned(15 DOWNTO 0);
  VARIABLE cast_161 : unsigned(9 DOWNTO 0);
  VARIABLE sub_temp_21 : unsigned(15 DOWNTO 0);
  VARIABLE cast_162 : unsigned(10 DOWNTO 0);
  VARIABLE cast_163 : unsigned(8 DOWNTO 0);
  VARIABLE mul_temp_73 : unsigned(22 DOWNTO 0);
  VARIABLE cast_164 : unsigned(8 DOWNTO 0);
BEGIN
  --HDL code generation from MATLAB function: amp_phase_state_fixpt
  FI_SIN_COS_LUT := nc;
  fi_sin_cos_lut_0 := nc_0;
```

```
    --
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --
%
    --            Generated by MATLAB 9.4 and Fixed-Point Designer 6.1
%
    --
%
    --
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    -- function [betax,betay] = betaxbetay(looktheta,lookphi,f)
    --function
[AB_phase1,AB_phase2,AB_phase3,AB_phase4,AB_phase5,AB_phase6,AB_phase7,AB_pha
se8,AB_phase9,AB_phase10,AB_phase11,AB_phase12,AB_phase13,AB_phase14,AB_phase
15,AB_phase16] = betaxbetay(looktheta,lookphi,f)
    IF f_unsigned = to_unsigned(0, 35) THEN
      tmp := unsigned'(X"FFFFFFFFFFFFFFFF");
    ELSE
      IF f_unsigned = 0 THEN
        div_temp := c_divbyzero_p_0;
      ELSE
        div_temp := one_0 / f_unsigned;
      END IF;
      tmp := resize(div_temp, 64);
    END IF;
    lambda := tmp(17 DOWNTO 4);
    IF lambda = to_unsigned(16#0000#, 14) THEN
      tmp_0 := unsigned'("11111111111111111111111111111111111111");
    ELSE
      IF lambda = 0 THEN
        div_temp_0 := C_divbyzero_p;
      ELSE
        div_temp_0 := One / lambda;
      END IF;
      tmp_0 := div_temp_0(37 DOWNTO 0);
    END IF;
    k := tmp_0(33 DOWNTO 20);
    --distance between elements (X)
```

```
    --distance between elements (Y)
    mul_temp := looktheta_unsigned * to_unsigned(16#3243#, 14);
    x := mul_temp * to_unsigned(16#2D82#, 14);
    cast := x & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0';
    div_temp_1 := cast / unsigned'(X"C90FDAA2");
    cast_0 := div_temp_1(35 DOWNTO 0);
    cast_1 := cast_0(35 DOWNTO 33);
    mul_temp_0 := cast_1 * unsigned'(X"C90FDAA2");
    sub_cast := mul_temp_0(31 DOWNTO 0) & '0' & '0' & '0' & '0';
    sub_temp := x - sub_cast;
    cast_2 := sub_temp(35 DOWNTO 20);
    mul_temp_1 := unsigned'(X"A2F96524") * cast_2;
    cast_3 := mul_temp_1(46 DOWNTO 15);
    fullScaleIndex := cast_3(31 DOWNTO 16);
    idxLUTLoZero := fullScaleIndex(15 DOWNTO 8);
    add_cast := signed(resize(idxLUTLoZero, 16));
    c1 := add_cast + to_signed(16#0001#, 16);
    mul_temp_2 := lookphi_unsigned * to_unsigned(16#3243#, 14);
    x_0 := mul_temp_2 * to_unsigned(16#2D82#, 14);
    cast_4 := x_0 & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0';
    div_temp_2 := cast_4 / unsigned'(X"C90FDAA2");
    cast_5 := div_temp_2(35 DOWNTO 0);
    cast_6 := cast_5(35 DOWNTO 33);
    mul_temp_3 := cast_6 * unsigned'(X"C90FDAA2");
    sub_cast_0 := mul_temp_3(31 DOWNTO 0) & '0' & '0' & '0' & '0';
    sub_temp_0 := x_0 - sub_cast_0;
    cast_7 := sub_temp_0(35 DOWNTO 20);
    mul_temp_4 := unsigned'(X"A2F96524") * cast_7;
    cast_8 := mul_temp_4(46 DOWNTO 15);
    fullscaleindex_0 := cast_8(31 DOWNTO 16);
    idxlutlozero_0 := fullscaleindex_0(15 DOWNTO 8);
    cast_9 := signed(resize(k, 15));
    cast_10 :=  - (cast_9);
    cast_11 := lambda & '0';
```

```
    cast_12 := SHIFT_RIGHT(cast_11, 1);
    cast_13 := cast_12(13 DOWNTO 0);
    cast_14 := signed(resize(cast_13, 15));
    mul_temp_5 := cast_10 * cast_14;
    add_cast_0 := resize(fi_sin_cos_lut_0(to_integer(resize(c1, 32) - 1)) &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0', 32);
    cast_15 := signed(resize(fullScaleIndex(7 DOWNTO 0), 9));
    add_temp := idxLUTLoZero + to_unsigned(16#01#, 8);
    add_cast_1 := signed(resize(add_temp, 16));
    sub_cast_1 := resize(fi_sin_cos_lut_0(to_integer(resize(add_cast_1 +
to_signed(16#0001#, 16), 32) - 1)) & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    sub_cast_2 := resize(fi_sin_cos_lut_0(to_integer(resize(c1, 32) - 1)) &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0', 32);
    sub_temp_1 := sub_cast_1 - sub_cast_2;
    mul_temp_6 := cast_15 * sub_temp_1;
    add_cast_2 := mul_temp_6(39 DOWNTO 8);
    add_cast_3 := add_cast_2(30 DOWNTO 15);
    add_cast_4 := resize(add_cast_3 & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    add_temp_0 := add_cast_0 + add_cast_4;
    cast_16 := add_temp_0(30 DOWNTO 15);
    mul_temp_7 := mul_temp_5 * cast_16;
    add_cast_5 := resize(FI_SIN_COS_LUT(to_integer(idxlutlozero_0)) & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0', 32);
    cast_17 := signed(resize(fullscaleindex_0(7 DOWNTO 0), 9));
    add_temp_1 := idxlutlozero_0 + to_unsigned(16#01#, 8);
    add_cast_6 := signed(resize(add_temp_1, 16));
    sub_cast_3 := resize(FI_SIN_COS_LUT(to_integer(resize(add_cast_6 +
to_signed(16#0001#, 16), 32) - 1)) & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    sub_cast_4 := resize(FI_SIN_COS_LUT(to_integer(idxlutlozero_0)) & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0', 32);
    sub_temp_2 := sub_cast_3 - sub_cast_4;
```

```
    mul_temp_8 := cast_17 * sub_temp_2;
    add_cast_7 := mul_temp_8(39 DOWNTO 8);
    add_cast_8 := add_cast_7(30 DOWNTO 15);
    add_cast_9 := resize(add_cast_8 & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    add_temp_2 := add_cast_5 + add_cast_9;
    cast_18 := add_temp_2(30 DOWNTO 15);
    mul_temp_9 := mul_temp_7 * cast_18;
    betax := mul_temp_9(57 DOWNTO 44);
    mul_temp_10 := looktheta_unsigned * to_unsigned(16#3243#, 14);
    x_1 := mul_temp_10 * to_unsigned(16#2D82#, 14);
    cast_19 := x_1 & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0';
    div_temp_3 := cast_19 / unsigned'(X"C90FDAA2");
    cast_20 := div_temp_3(35 DOWNTO 0);
    cast_21 := cast_20(35 DOWNTO 33);
    mul_temp_11 := cast_21 * unsigned'(X"C90FDAA2");
    sub_cast_5 := mul_temp_11(31 DOWNTO 0) & '0' & '0' & '0' & '0';
    sub_temp_3 := x_1 - sub_cast_5;
    cast_22 := sub_temp_3(35 DOWNTO 20);
    mul_temp_12 := unsigned'(X"A2F96524") * cast_22;
    cast_23 := mul_temp_12(46 DOWNTO 15);
    fullscaleindex_1 := cast_23(31 DOWNTO 16);
    idxlutlozero_1 := fullscaleindex_1(15 DOWNTO 8);
    add_cast_10 := signed(resize(idxlutlozero_1, 16));
    c1_0 := add_cast_10 + to_signed(16#0001#, 16);
    mul_temp_13 := lookphi_unsigned * to_unsigned(16#3243#, 14);
    x_2 := mul_temp_13 * to_unsigned(16#2D82#, 14);
    cast_24 := x_2 & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0';
    div_temp_4 := cast_24 / unsigned'(X"C90FDAA2");
    cast_25 := div_temp_4(35 DOWNTO 0);
    cast_26 := cast_25(35 DOWNTO 33);
    mul_temp_14 := cast_26 * unsigned'(X"C90FDAA2");
    sub_cast_6 := mul_temp_14(31 DOWNTO 0) & '0' & '0' & '0' & '0';
    sub_temp_4 := x_2 - sub_cast_6;
```

```
    cast_27 := sub_temp_4(35 DOWNTO 20);
    mul_temp_15 := unsigned'(X"A2F96524") * cast_27;
    cast_28 := mul_temp_15(46 DOWNTO 15);
    fullscaleindex_2 := cast_28(31 DOWNTO 16);
    idxlutlozero_2 := fullscaleindex_2(15 DOWNTO 8);
    cast_29 := signed(resize(k, 15));
    cast_30 :=  - (cast_29);
    cast_31 := lambda & '0';
    cast_32 := SHIFT_RIGHT(cast_31, 1);
    cast_33 := cast_32(13 DOWNTO 0);
    cast_34 := signed(resize(cast_33, 15));
    mul_temp_16 := cast_30 * cast_34;
    add_cast_11 := resize(fi_sin_cos_lut_0(to_integer(resize(c1_0, 32) - 1))
& '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0', 32);
    cast_35 := signed(resize(fullscaleindex_1(7 DOWNTO 0), 9));
    add_temp_3 := idxlutlozero_1 + to_unsigned(16#01#, 8);
    add_cast_12 := signed(resize(add_temp_3, 16));
    sub_cast_7 := resize(fi_sin_cos_lut_0(to_integer(resize(add_cast_12 +
to_signed(16#0001#, 16), 32) - 1)) & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    sub_cast_8 := resize(fi_sin_cos_lut_0(to_integer(resize(c1_0, 32) - 1)) &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0', 32);
    sub_temp_5 := sub_cast_7 - sub_cast_8;
    mul_temp_17 := cast_35 * sub_temp_5;
    add_cast_13 := mul_temp_17(39 DOWNTO 8);
    add_cast_14 := add_cast_13(30 DOWNTO 15);
    add_cast_15 := resize(add_cast_14 & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    add_temp_4 := add_cast_11 + add_cast_15;
    cast_36 := add_temp_4(30 DOWNTO 15);
    mul_temp_18 := mul_temp_16 * cast_36;
    add_cast_16 := resize(fi_sin_cos_lut_0(to_integer(idxlutlozero_2)) & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0', 32);
    cast_37 := signed(resize(fullscaleindex_2(7 DOWNTO 0), 9));
    add_temp_5 := idxlutlozero_2 + to_unsigned(16#01#, 8);
```

```
    add_cast_17 := signed(resize(add_temp_5, 16));
    sub_cast_9 := resize(fi_sin_cos_lut_0(to_integer(resize(add_cast_17 +
to_signed(16#0001#, 16), 32) - 1)) & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    sub_cast_10 := resize(fi_sin_cos_lut_0(to_integer(idxlutlozero_2)) & '0'
& '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
& '0', 32);
    sub_temp_6 := sub_cast_9 - sub_cast_10;
    mul_temp_19 := cast_37 * sub_temp_6;
    add_cast_18 := mul_temp_19(39 DOWNTO 8);
    add_cast_19 := add_cast_18(30 DOWNTO 15);
    add_cast_20 := resize(add_cast_19 & '0' & '0' & '0' & '0' & '0' & '0' &
'0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0', 32);
    add_temp_6 := add_cast_16 + add_cast_20;
    cast_38 := add_temp_6(30 DOWNTO 15);
    mul_temp_20 := mul_temp_18 * cast_38;
    betay := mul_temp_20(57 DOWNTO 44);
    --          AB_phase = zeros(4,4);
    --        for a=1:row
    --             for b=1:column
    --                 AB_phase(a,b) = (a-(row/2))*betax + (b-
(column/2))*betay;
    --             end
    --         end
    --temp=10^(attenuation/10);
    cast_39 := resize(betay & '0', 17);
    cast_40 := cast_39(15 DOWNTO 0);
    AB3_rad := cast_40(14 DOWNTO 1);
    mul_temp_21 := to_signed(16#3#, 3) * betay;
    cast_41 := mul_temp_21(15 DOWNTO 0);
    AB4_rad := cast_41(14 DOWNTO 1);
    add_cast_21 := resize(betay, 15);
    add_cast_22 := resize(betax, 15);
    add_temp_7 := add_cast_21 + add_cast_22;
    AB6_rad := add_temp_7(14 DOWNTO 1);
    add_cast_23 := resize(betay & '0', 17);
    add_cast_24 := add_cast_23(15 DOWNTO 0);
    add_cast_25 := resize(add_cast_24, 17);
```

```
add_cast_26 := resize(betax, 17);
add_temp_8 := add_cast_25 + add_cast_26;
AB7_rad := add_temp_8(14 DOWNTO 1);
mul_temp_22 := to_signed(16#3#, 3) * betay;
add_cast_27 := mul_temp_22(15 DOWNTO 0);
add_cast_28 := resize(add_cast_27, 17);
add_cast_29 := resize(betax, 17);
add_temp_9 := add_cast_28 + add_cast_29;
AB8_rad := add_temp_9(15 DOWNTO 2);
cast_42 := resize(betax & '0', 17);
cast_43 := cast_42(15 DOWNTO 0);
AB9_rad := cast_43(14 DOWNTO 1);
add_cast_30 := resize(betay, 17);
add_cast_31 := resize(betax & '0', 17);
add_cast_32 := add_cast_31(15 DOWNTO 0);
add_cast_33 := resize(add_cast_32, 17);
add_temp_10 := add_cast_30 + add_cast_33;
AB10_rad := add_temp_10(14 DOWNTO 1);
add_cast_34 := resize(betay & '0', 17);
add_cast_35 := add_cast_34(15 DOWNTO 0);
add_cast_36 := resize(add_cast_35, 17);
add_cast_37 := resize(betax & '0', 17);
add_cast_38 := add_cast_37(15 DOWNTO 0);
add_cast_39 := resize(add_cast_38, 17);
add_temp_11 := add_cast_36 + add_cast_39;
AB11_rad := add_temp_11(15 DOWNTO 2);
mul_temp_23 := to_signed(16#3#, 3) * betay;
add_cast_40 := mul_temp_23(15 DOWNTO 0);
add_cast_41 := resize(add_cast_40, 17);
add_cast_42 := resize(betax & '0', 17);
add_cast_43 := add_cast_42(15 DOWNTO 0);
add_cast_44 := resize(add_cast_43, 17);
add_temp_12 := add_cast_41 + add_cast_44;
AB12_rad := add_temp_12(15 DOWNTO 2);
mul_temp_24 := to_signed(16#3#, 3) * betax;
cast_44 := mul_temp_24(15 DOWNTO 0);
AB13_rad := cast_44(14 DOWNTO 1);
add_cast_45 := resize(betay, 17);
```

```
mul_temp_25 := to_signed(16#3#, 3) * betax;
add_cast_46 := mul_temp_25(15 DOWNTO 0);
add_cast_47 := resize(add_cast_46, 17);
add_temp_13 := add_cast_45 + add_cast_47;
AB14_rad := add_temp_13(15 DOWNTO 2);
add_cast_48 := resize(betay & '0', 17);
add_cast_49 := add_cast_48(15 DOWNTO 0);
add_cast_50 := resize(add_cast_49, 17);
mul_temp_26 := to_signed(16#3#, 3) * betax;
add_cast_51 := mul_temp_26(15 DOWNTO 0);
add_cast_52 := resize(add_cast_51, 17);
add_temp_14 := add_cast_50 + add_cast_52;
AB15_rad := add_temp_14(15 DOWNTO 2);
mul_temp_27 := to_signed(16#3#, 3) * betay;
add_cast_53 := mul_temp_27(15 DOWNTO 0);
add_cast_54 := resize(add_cast_53, 17);
mul_temp_28 := to_signed(16#3#, 3) * betax;
add_cast_55 := mul_temp_28(15 DOWNTO 0);
add_cast_56 := resize(add_cast_55, 17);
add_temp_15 := add_cast_54 + add_cast_56;
AB16_rad := add_temp_15(15 DOWNTO 2);
--AB_mod = mod (AB,(2*pi));
IF betay < to_signed(16#0000#, 14) THEN
  cast_45 := resize(betay, 15);
  cast_46 :=  - (cast_45);
  cast_47 := resize(cast_46, 16);
  tmp_1 := unsigned(cast_47(13 DOWNTO 0));
ELSE
  tmp_1 := unsigned(betay);
END IF;
mul_temp_29 := tmp_1 * to_unsigned(16#394B#, 14);
AB_deg := mul_temp_29(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB3_rad < to_signed(16#0000#, 14) THEN
  cast_48 := resize(AB3_rad, 15);
  cast_49 :=  - (cast_48);
  cast_50 := resize(cast_49, 16);
  tmp_2 := unsigned(cast_50(12 DOWNTO 0) & '0');
```

```
ELSE
  tmp_2 := unsigned(AB3_rad(12 DOWNTO 0) & '0');
END IF;
mul_temp_30 := tmp_2 * to_unsigned(16#394B#, 14);
ab_deg_0 := mul_temp_30(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB4_rad < to_signed(16#0000#, 14) THEN
  cast_51 := resize(AB4_rad, 15);
  cast_52 :=  - (cast_51);
  cast_53 := resize(cast_52, 16);
  tmp_3 := unsigned(cast_53(12 DOWNTO 0) & '0');
ELSE
  tmp_3 := unsigned(AB4_rad(12 DOWNTO 0) & '0');
END IF;
mul_temp_31 := tmp_3 * to_unsigned(16#394B#, 14);
ab_deg_1 := mul_temp_31(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF betax < to_signed(16#0000#, 14) THEN
  cast_54 := resize(betax, 15);
  cast_55 :=  - (cast_54);
  cast_56 := resize(cast_55, 16);
  tmp_4 := unsigned(cast_56(13 DOWNTO 0));
ELSE
  tmp_4 := unsigned(betax);
END IF;
mul_temp_32 := tmp_4 * to_unsigned(16#394B#, 14);
ab_deg_2 := mul_temp_32(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB6_rad < to_signed(16#0000#, 14) THEN
  cast_57 := resize(AB6_rad, 15);
  cast_58 :=  - (cast_57);
  cast_59 := resize(cast_58, 16);
  tmp_5 := unsigned(cast_59(12 DOWNTO 0) & '0');
ELSE
  tmp_5 := unsigned(AB6_rad(12 DOWNTO 0) & '0');
END IF;
mul_temp_33 := tmp_5 * to_unsigned(16#394B#, 14);
ab_deg_3 := mul_temp_33(27 DOWNTO 14);
```

```
--AB_mod = mod (AB,(2*pi));
IF AB7_rad < to_signed(16#0000#, 14) THEN
  cast_60 := resize(AB7_rad, 15);
  cast_61 :=  - (cast_60);
  cast_62 := resize(cast_61, 16);
  tmp_6 := unsigned(cast_62(12 DOWNTO 0) & '0');
ELSE
  tmp_6 := unsigned(AB7_rad(12 DOWNTO 0) & '0');
END IF;
mul_temp_34 := tmp_6 * to_unsigned(16#394B#, 14);
ab_deg_4 := mul_temp_34(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB8_rad < to_signed(16#0000#, 14) THEN
  cast_63 := resize(AB8_rad, 15);
  cast_64 :=  - (cast_63);
  cast_65 := resize(cast_64, 16);
  tmp_7 := unsigned(cast_65(11 DOWNTO 0) & '0' & '0');
ELSE
  tmp_7 := unsigned(AB8_rad(11 DOWNTO 0) & '0' & '0');
END IF;
mul_temp_35 := tmp_7 * to_unsigned(16#394B#, 14);
ab_deg_5 := mul_temp_35(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB9_rad < to_signed(16#0000#, 14) THEN
  cast_66 := resize(AB9_rad, 15);
  cast_67 :=  - (cast_66);
  cast_68 := resize(cast_67, 16);
  tmp_8 := unsigned(cast_68(12 DOWNTO 0) & '0');
ELSE
  tmp_8 := unsigned(AB9_rad(12 DOWNTO 0) & '0');
END IF;
mul_temp_36 := tmp_8 * to_unsigned(16#394B#, 14);
ab_deg_6 := mul_temp_36(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB10_rad < to_signed(16#0000#, 14) THEN
  cast_69 := resize(AB10_rad, 15);
  cast_70 :=  - (cast_69);
  cast_71 := resize(cast_70, 16);
```

```
  tmp_9 := unsigned(cast_71(12 DOWNTO 0) & '0');
ELSE
  tmp_9 := unsigned(AB10_rad(12 DOWNTO 0) & '0');
END IF;
mul_temp_37 := tmp_9 * to_unsigned(16#394B#, 14);
ab_deg_7 := mul_temp_37(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB11_rad < to_signed(16#0000#, 14) THEN
  cast_72 := resize(AB11_rad, 15);
  cast_73 :=  - (cast_72);
  cast_74 := resize(cast_73, 16);
  tmp_10 := unsigned(cast_74(11 DOWNTO 0) & '0' & '0');
ELSE
  tmp_10 := unsigned(AB11_rad(11 DOWNTO 0) & '0' & '0');
END IF;
mul_temp_38 := tmp_10 * to_unsigned(16#394B#, 14);
ab_deg_8 := mul_temp_38(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB12_rad < to_signed(16#0000#, 14) THEN
  cast_75 := resize(AB12_rad, 15);
  cast_76 :=  - (cast_75);
  cast_77 := resize(cast_76, 16);
  tmp_11 := unsigned(cast_77(11 DOWNTO 0) & '0' & '0');
ELSE
  tmp_11 := unsigned(AB12_rad(11 DOWNTO 0) & '0' & '0');
END IF;
mul_temp_39 := tmp_11 * to_unsigned(16#394B#, 14);
ab_deg_9 := mul_temp_39(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB13_rad < to_signed(16#0000#, 14) THEN
  cast_78 := resize(AB13_rad, 15);
  cast_79 :=  - (cast_78);
  cast_80 := resize(cast_79, 16);
  tmp_12 := unsigned(cast_80(12 DOWNTO 0) & '0');
ELSE
  tmp_12 := unsigned(AB13_rad(12 DOWNTO 0) & '0');
END IF;
mul_temp_40 := tmp_12 * to_unsigned(16#394B#, 14);
```

```
ab_deg_10 := mul_temp_40(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB14_rad < to_signed(16#0000#, 14) THEN
  cast_81 := resize(AB14_rad, 15);
  cast_82 :=  - (cast_81);
  cast_83 := resize(cast_82, 16);
  tmp_13 := unsigned(cast_83(11 DOWNTO 0) & '0' & '0');
ELSE
  tmp_13 := unsigned(AB14_rad(11 DOWNTO 0) & '0' & '0');
END IF;
mul_temp_41 := tmp_13 * to_unsigned(16#394B#, 14);
ab_deg_11 := mul_temp_41(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB15_rad < to_signed(16#0000#, 14) THEN
  cast_84 := resize(AB15_rad, 15);
  cast_85 :=  - (cast_84);
  cast_86 := resize(cast_85, 16);
  tmp_14 := unsigned(cast_86(11 DOWNTO 0) & '0' & '0');
ELSE
  tmp_14 := unsigned(AB15_rad(11 DOWNTO 0) & '0' & '0');
END IF;
mul_temp_42 := tmp_14 * to_unsigned(16#394B#, 14);
ab_deg_12 := mul_temp_42(27 DOWNTO 14);
--AB_mod = mod (AB,(2*pi));
IF AB16_rad < to_signed(16#0000#, 14) THEN
  cast_87 := resize(AB16_rad, 15);
  cast_88 :=  - (cast_87);
  cast_89 := resize(cast_88, 16);
  tmp_15 := unsigned(cast_89(11 DOWNTO 0) & '0' & '0');
ELSE
  tmp_15 := unsigned(AB16_rad(11 DOWNTO 0) & '0' & '0');
END IF;
mul_temp_43 := tmp_15 * to_unsigned(16#394B#, 14);
ab_deg_13 := mul_temp_43(27 DOWNTO 14);
IF attenuation = '1' THEN
  AB1_tmp <= to_unsigned(16#22#, 6);
ELSE
  AB1_tmp <= to_unsigned(16#00#, 6);
```

```
END IF;
IF attenuation = '1' THEN
  AB2_tmp <= to_unsigned(16#53#, 7);
ELSE
  AB2_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB3_tmp <= to_unsigned(16#53#, 7);
ELSE
  AB3_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB4_tmp <= to_unsigned(16#22#, 6);
ELSE
  AB4_tmp <= to_unsigned(16#00#, 6);
END IF;
IF attenuation = '1' THEN
  AB5_tmp <= to_unsigned(16#53#, 7);
ELSE
  AB5_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB6_tmp <= to_unsigned(16#C7#, 8);
ELSE
  AB6_tmp <= to_unsigned(16#00#, 8);
END IF;
IF attenuation = '1' THEN
  AB7_tmp <= to_unsigned(16#C7#, 8);
ELSE
  AB7_tmp <= to_unsigned(16#00#, 8);
END IF;
IF attenuation = '1' THEN
  AB8_tmp <= to_unsigned(16#53#, 7);
ELSE
  AB8_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB9_tmp <= to_unsigned(16#53#, 7);
```

```
ELSE
  AB9_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB10_tmp <= to_unsigned(16#C7#, 8);
ELSE
  AB10_tmp <= to_unsigned(16#00#, 8);
END IF;
IF attenuation = '1' THEN
  AB11_tmp <= to_unsigned(16#C7#, 8);
ELSE
  AB11_tmp <= to_unsigned(16#00#, 8);
END IF;
IF attenuation = '1' THEN
  AB12_tmp <= to_unsigned(16#53#, 7);
ELSE
  AB12_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB13_tmp <= to_unsigned(16#22#, 6);
ELSE
  AB13_tmp <= to_unsigned(16#00#, 6);
END IF;
IF attenuation = '1' THEN
  AB14_tmp <= to_unsigned(16#53#, 7);
ELSE
  AB14_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB15_tmp <= to_unsigned(16#53#, 7);
ELSE
  AB15_tmp <= to_unsigned(16#00#, 7);
END IF;
IF attenuation = '1' THEN
  AB16_tmp <= to_unsigned(16#22#, 6);
ELSE
  AB16_tmp <= to_unsigned(16#00#, 6);
END IF;
```

```
-- AB1_phase <= '0';
 sub_cast_11 := resize(AB_deg, 16);
 mul_temp_44 := AB_deg * to_unsigned(16#2D82#, 14);
 cast_90 := mul_temp_44(27);
 IF cast_90 = '1' THEN
   cast_91 := to_unsigned(16#168#, 10);
 ELSE
   cast_91 := to_unsigned(16#000#, 10);
 END IF;
 sub_cast_12 := resize(cast_91 & '0' & '0' & '0' & '0' & '0', 16);
 sub_temp_7 := sub_cast_11 - sub_cast_12;
 cast_92 := sub_temp_7(15 DOWNTO 5);
 cast_93 := cast_92(8 DOWNTO 0);
 mul_temp_45 := cast_93 * to_unsigned(16#2DB6#, 14);
 cast_94 := mul_temp_45(22 DOWNTO 14);
 AB2_phase_tmp <= cast_94(6 DOWNTO 0);
 sub_cast_13 := resize(ab_deg_0, 16);
 mul_temp_46 := ab_deg_0 * to_unsigned(16#2D82#, 14);
 cast_95 := mul_temp_46(27);
 IF cast_95 = '1' THEN
   cast_96 := to_unsigned(16#168#, 10);
 ELSE
   cast_96 := to_unsigned(16#000#, 10);
 END IF;
 sub_cast_14 := resize(cast_96 & '0' & '0' & '0' & '0' & '0', 16);
 sub_temp_8 := sub_cast_13 - sub_cast_14;
 cast_97 := sub_temp_8(15 DOWNTO 5);
 cast_98 := cast_97(8 DOWNTO 0);
 mul_temp_47 := cast_98 * to_unsigned(16#2DB6#, 14);
 cast_99 := mul_temp_47(22 DOWNTO 14);
 AB3_phase_tmp <= cast_99(7 DOWNTO 0);
 sub_cast_15 := resize(ab_deg_1, 16);
 mul_temp_48 := ab_deg_1 * to_unsigned(16#2D82#, 14);
 cast_100 := mul_temp_48(27);
 IF cast_100 = '1' THEN
   cast_101 := to_unsigned(16#168#, 10);
 ELSE
   cast_101 := to_unsigned(16#000#, 10);
```

```
END IF;
sub_cast_16 := resize(cast_101 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_9 := sub_cast_15 - sub_cast_16;
cast_102 := sub_temp_9(15 DOWNTO 5);
cast_103 := cast_102(8 DOWNTO 0);
mul_temp_49 := cast_103 * to_unsigned(16#2DB6#, 14);
cast_104 := mul_temp_49(22 DOWNTO 14);
AB4_phase_tmp <= cast_104(7 DOWNTO 0);
sub_cast_17 := resize(ab_deg_2, 16);
mul_temp_50 := ab_deg_2 * to_unsigned(16#2D82#, 14);
cast_105 := mul_temp_50(27);
IF cast_105 = '1' THEN
  cast_106 := to_unsigned(16#168#, 10);
ELSE
  cast_106 := to_unsigned(16#000#, 10);
END IF;
sub_cast_18 := resize(cast_106 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_10 := sub_cast_17 - sub_cast_18;
cast_107 := sub_temp_10(15 DOWNTO 5);
cast_108 := cast_107(8 DOWNTO 0);
mul_temp_51 := cast_108 * to_unsigned(16#2DB6#, 14);
cast_109 := mul_temp_51(22 DOWNTO 14);
AB5_phase_tmp <= cast_109(6 DOWNTO 0);
sub_cast_19 := resize(ab_deg_3, 16);
mul_temp_52 := ab_deg_3 * to_unsigned(16#2D82#, 14);
cast_110 := mul_temp_52(27);
IF cast_110 = '1' THEN
  cast_111 := to_unsigned(16#168#, 10);
ELSE
  cast_111 := to_unsigned(16#000#, 10);
END IF;
sub_cast_20 := resize(cast_111 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_11 := sub_cast_19 - sub_cast_20;
cast_112 := sub_temp_11(15 DOWNTO 5);
cast_113 := cast_112(8 DOWNTO 0);
mul_temp_53 := cast_113 * to_unsigned(16#2DB6#, 14);
cast_114 := mul_temp_53(22 DOWNTO 14);
AB6_phase_tmp <= cast_114(7 DOWNTO 0);
```

```
sub_cast_21 := resize(ab_deg_4, 16);
mul_temp_54 := ab_deg_4 * to_unsigned(16#2D82#, 14);
cast_115 := mul_temp_54(27);
IF cast_115 = '1' THEN
  cast_116 := to_unsigned(16#168#, 10);
ELSE
  cast_116 := to_unsigned(16#000#, 10);
END IF;
sub_cast_22 := resize(cast_116 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_12 := sub_cast_21 - sub_cast_22;
cast_117 := sub_temp_12(15 DOWNTO 5);
cast_118 := cast_117(8 DOWNTO 0);
mul_temp_55 := cast_118 * to_unsigned(16#2DB6#, 14);
cast_119 := mul_temp_55(22 DOWNTO 14);
AB7_phase_tmp <= cast_119(7 DOWNTO 0);
sub_cast_23 := resize(ab_deg_5, 16);
mul_temp_56 := ab_deg_5 * to_unsigned(16#2D82#, 14);
cast_120 := mul_temp_56(27);
IF cast_120 = '1' THEN
  cast_121 := to_unsigned(16#168#, 10);
ELSE
  cast_121 := to_unsigned(16#000#, 10);
END IF;
sub_cast_24 := resize(cast_121 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_13 := sub_cast_23 - sub_cast_24;
cast_122 := sub_temp_13(15 DOWNTO 5);
cast_123 := cast_122(8 DOWNTO 0);
mul_temp_57 := cast_123 * to_unsigned(16#2DB6#, 14);
cast_124 := mul_temp_57(22 DOWNTO 14);
AB8_phase_tmp <= cast_124(6 DOWNTO 0);
sub_cast_25 := resize(ab_deg_6, 16);
mul_temp_58 := ab_deg_6 * to_unsigned(16#2D82#, 14);
cast_125 := mul_temp_58(27);
IF cast_125 = '1' THEN
  cast_126 := to_unsigned(16#168#, 10);
ELSE
  cast_126 := to_unsigned(16#000#, 10);
END IF;
```

```
sub_cast_26 := resize(cast_126 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_14 := sub_cast_25 - sub_cast_26;
cast_127 := sub_temp_14(15 DOWNTO 5);
cast_128 := cast_127(8 DOWNTO 0);
mul_temp_59 := cast_128 * to_unsigned(16#2DB6#, 14);
cast_129 := mul_temp_59(22 DOWNTO 14);
AB9_phase_tmp <= cast_129(7 DOWNTO 0);
sub_cast_27 := resize(ab_deg_7, 16);
mul_temp_60 := ab_deg_7 * to_unsigned(16#2D82#, 14);
cast_130 := mul_temp_60(27);
IF cast_130 = '1' THEN
  cast_131 := to_unsigned(16#168#, 10);
ELSE
  cast_131 := to_unsigned(16#000#, 10);
END IF;
sub_cast_28 := resize(cast_131 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_15 := sub_cast_27 - sub_cast_28;
cast_132 := sub_temp_15(15 DOWNTO 5);
cast_133 := cast_132(8 DOWNTO 0);
mul_temp_61 := cast_133 * to_unsigned(16#2DB6#, 14);
cast_134 := mul_temp_61(22 DOWNTO 14);
AB10_phase_tmp <= cast_134(7 DOWNTO 0);
sub_cast_29 := resize(ab_deg_8, 16);
mul_temp_62 := ab_deg_8 * to_unsigned(16#2D82#, 14);
cast_135 := mul_temp_62(27);
IF cast_135 = '1' THEN
  cast_136 := to_unsigned(16#168#, 10);
ELSE
  cast_136 := to_unsigned(16#000#, 10);
END IF;
sub_cast_30 := resize(cast_136 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_16 := sub_cast_29 - sub_cast_30;
cast_137 := sub_temp_16(15 DOWNTO 5);
cast_138 := cast_137(8 DOWNTO 0);
mul_temp_63 := cast_138 * to_unsigned(16#2DB6#, 14);
cast_139 := mul_temp_63(22 DOWNTO 14);
AB11_phase_tmp <= cast_139(7 DOWNTO 0);
sub_cast_31 := resize(ab_deg_9, 16);
```

```
mul_temp_64 := ab_deg_9 * to_unsigned(16#2D82#, 14);
cast_140 := mul_temp_64(27);
IF cast_140 = '1' THEN
  cast_141 := to_unsigned(16#168#, 10);
ELSE
  cast_141 := to_unsigned(16#000#, 10);
END IF;
sub_cast_32 := resize(cast_141 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_17 := sub_cast_31 - sub_cast_32;
cast_142 := sub_temp_17(15 DOWNTO 5);
cast_143 := cast_142(8 DOWNTO 0);
mul_temp_65 := cast_143 * to_unsigned(16#2DB6#, 14);
cast_144 := mul_temp_65(22 DOWNTO 14);
AB12_phase_tmp <= cast_144(7 DOWNTO 0);
sub_cast_33 := resize(ab_deg_10, 16);
mul_temp_66 := ab_deg_10 * to_unsigned(16#2D82#, 14);
cast_145 := mul_temp_66(27);
IF cast_145 = '1' THEN
  cast_146 := to_unsigned(16#168#, 10);
ELSE
  cast_146 := to_unsigned(16#000#, 10);
END IF;
sub_cast_34 := resize(cast_146 & '0' & '0' & '0' & '0' & '0', 16);
sub_temp_18 := sub_cast_33 - sub_cast_34;
cast_147 := sub_temp_18(15 DOWNTO 5);
cast_148 := cast_147(8 DOWNTO 0);
mul_temp_67 := cast_148 * to_unsigned(16#2DB6#, 14);
cast_149 := mul_temp_67(22 DOWNTO 14);
AB13_phase_tmp <= cast_149(7 DOWNTO 0);
sub_cast_35 := resize(ab_deg_11, 16);
mul_temp_68 := ab_deg_11 * to_unsigned(16#2D82#, 14);
cast_150 := mul_temp_68(27);
IF cast_150 = '1' THEN
  cast_151 := to_unsigned(16#168#, 10);
ELSE
  cast_151 := to_unsigned(16#000#, 10);
END IF;
sub_cast_36 := resize(cast_151 & '0' & '0' & '0' & '0' & '0', 16);
```

```
  sub_temp_19 := sub_cast_35 - sub_cast_36;
  cast_152 := sub_temp_19(15 DOWNTO 5);
  cast_153 := cast_152(8 DOWNTO 0);
  mul_temp_69 := cast_153 * to_unsigned(16#2DB6#, 14);
  cast_154 := mul_temp_69(22 DOWNTO 14);
  AB14_phase_tmp <= cast_154(7 DOWNTO 0);
  sub_cast_37 := resize(ab_deg_12, 16);
  mul_temp_70 := ab_deg_12 * to_unsigned(16#2D82#, 14);
  cast_155 := mul_temp_70(27);
  IF cast_155 = '1' THEN
    cast_156 := to_unsigned(16#168#, 10);
  ELSE
    cast_156 := to_unsigned(16#000#, 10);
  END IF;
  sub_cast_38 := resize(cast_156 & '0' & '0' & '0' & '0' & '0', 16);
  sub_temp_20 := sub_cast_37 - sub_cast_38;
  cast_157 := sub_temp_20(15 DOWNTO 5);
  cast_158 := cast_157(8 DOWNTO 0);
  mul_temp_71 := cast_158 * to_unsigned(16#2DB6#, 14);
  cast_159 := mul_temp_71(22 DOWNTO 14);
  AB15_phase_tmp <= cast_159(7 DOWNTO 0);
  sub_cast_39 := resize(ab_deg_13, 16);
  mul_temp_72 := ab_deg_13 * to_unsigned(16#2D82#, 14);
  cast_160 := mul_temp_72(27);
  IF cast_160 = '1' THEN
    cast_161 := to_unsigned(16#168#, 10);
  ELSE
    cast_161 := to_unsigned(16#000#, 10);
  END IF;
  sub_cast_40 := resize(cast_161 & '0' & '0' & '0' & '0' & '0', 16);
  sub_temp_21 := sub_cast_39 - sub_cast_40;
  cast_162 := sub_temp_21(15 DOWNTO 5);
  cast_163 := cast_162(8 DOWNTO 0);
  mul_temp_73 := cast_163 * to_unsigned(16#2DB6#, 14);
  cast_164 := mul_temp_73(22 DOWNTO 14);
  AB16_phase_tmp <= cast_164(7 DOWNTO 0);
END PROCESS amp_phase_state_fixpt_1_output;
```

```
--  AB1 <= std_logic_vector(AB1_tmp);
--
--  AB2 <= std_logic_vector(AB2_tmp);
--
--  AB3 <= std_logic_vector(AB3_tmp);
--
--  AB4 <= std_logic_vector(AB4_tmp);
--
--  AB5 <= std_logic_vector(AB5_tmp);
--
--  AB6 <= std_logic_vector(AB6_tmp);
--
--  AB7 <= std_logic_vector(AB7_tmp);
--
--  AB8 <= std_logic_vector(AB8_tmp);
--
--  AB9 <= std_logic_vector(AB9_tmp);
--
--  AB10 <= std_logic_vector(AB10_tmp);
--
--  AB11 <= std_logic_vector(AB11_tmp);
--
--  AB12 <= std_logic_vector(AB12_tmp);
--
--  AB13 <= std_logic_vector(AB13_tmp);
--
--  AB14 <= std_logic_vector(AB14_tmp);
--
--  AB15 <= std_logic_vector(AB15_tmp);
--
--  AB16 <= std_logic_vector(AB16_tmp);
--
--  AB2_phase <= std_logic_vector(AB2_phase_tmp);
--
--  AB3_phase <= std_logic_vector(AB3_phase_tmp);
--
--  AB4_phase <= std_logic_vector(AB4_phase_tmp);
```

```
--
--  AB5_phase <= std_logic_vector(AB5_phase_tmp);
--
--  AB6_phase <= std_logic_vector(AB6_phase_tmp);
--
--  AB7_phase <= std_logic_vector(AB7_phase_tmp);
--
--  AB8_phase <= std_logic_vector(AB8_phase_tmp);
--
--  AB9_phase <= std_logic_vector(AB9_phase_tmp);
--
--  AB10_phase <= std_logic_vector(AB10_phase_tmp);
--
--  AB11_phase <= std_logic_vector(AB11_phase_tmp);
--
--  AB12_phase <= std_logic_vector(AB12_phase_tmp);
--
--  AB13_phase <= std_logic_vector(AB13_phase_tmp);
--
--  AB14_phase <= std_logic_vector(AB14_phase_tmp);
--
--  AB15_phase <= std_logic_vector(AB15_phase_tmp);
--
--  AB16_phase <= std_logic_vector(AB16_phase_tmp);



---------------------- clock divide by 2------------
process(CLK,por,clk_25mhz)
begin
if(por='1')then
clk_25mhz<='0';
elsif(CLK ='1' and CLK'event)then
 clk_25mhz<=not clk_25mhz;
end if;
clki_25mhz <= not clk_25mhz;
end process;
```

```
process (clki_25Mhz,por,starttogglecounter)
begin
if (por = '1' or starttogglecounter <= "1111") then
starttogglecounter <= "0000";
elsif (por = '0') then
     if(clki_25Mhz'event and clki_25Mhz = '1') then
     starttogglecounter <= starttogglecounter + "0001";
     end if;
else
null;
end if;
end process;


PROCESS
(starttogglecounter,AB1_tmp,AB2_tmp,AB3_tmp,AB4_tmp,AB5_tmp,AB6_tmp,AB7_tmp,A
B8_tmp,AB9_tmp,AB1_tmp,AB10_tmp,AB11_tmp,AB12_tmp,AB13_tmp,AB14_tmp,AB15_tmp,
AB16_tmp,AB2_phase_tmp,AB3_phase_tmp,AB4_phase_tmp,AB5_phase_tmp,AB6_phase_tm
p,AB7_phase_tmp,AB8_phase_tmp,AB9_phase_tmp,AB10_phase_tmp,AB11_phase_tmp,AB1
2_phase_tmp,AB13_phase_tmp,AB14_phase_tmp,AB15_phase_tmp,AB16_phase_tmp,mode_
selection_TX_RX_BITE)
begin

if (starttogglecounter = "0000") then

DATA_REG (39 DOWNTO 32) <= "00000000";
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & '0' & AB1_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & '0' & AB1_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & '0' & AB1_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;


ELSIF (starttogglecounter = "0001") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector('0' & AB2_phase_tmp);
```

```
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB2_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB2_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB2_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "0010") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB3_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB3_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB3_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB3_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "0011") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB4_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & '0' & AB4_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & '0' & AB4_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & '0' & AB4_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "0100") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector('0' & AB5_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB5_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB5_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB5_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "0101") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB6_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector(AB6_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector(AB6_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector(AB6_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "0110") then
```

```
DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB7_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector(AB7_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector(AB7_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector(AB7_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "0111") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector('0' & AB8_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB8_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB8_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB8_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "1000") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB9_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB9_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB9_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB9_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "1001") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB10_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector(AB10_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector(AB10_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector(AB10_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "1010") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB11_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector(AB11_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector(AB11_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector(AB11_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;
```

```
ELSIF (starttogglecounter = "1011") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB12_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB12_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB12_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB12_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "1100") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB13_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & '0' & AB13_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & '0' & AB13_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & '0' & AB13_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "1101") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB14_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB14_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB14_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB14_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "1110") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB15_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & AB15_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & AB15_tmp);
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & AB15_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSIF (starttogglecounter = "1111") then

DATA_REG (39 DOWNTO 32) <= std_logic_vector(AB16_phase_tmp);
DATA_REG (31 DOWNTO 24) <= std_logic_vector('0' & '0' & AB16_tmp);
DATA_REG (23 DOWNTO 16) <= std_logic_vector('0' & '0' & AB16_tmp);
```

```
DATA_REG (15 DOWNTO 8) <= std_logic_vector('0' & '0' & AB16_tmp);
DATA_REG (7 DOWNTO 0) <= mode_selection_TX_RX_BITE;

ELSE
NULL;
END IF;

END PROCESS;

END rtl;
```

2. **Customized Interface Code**.

```
----------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:45:47 11/28/2019
-- Design Name:
-- Module Name:    spicode - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity spicode is

port (CLK:IN STD_LOGIC;
por:in std_logic;
start: in std_logic;
fire: in std_logic;
reset_in: in std_logic;
--dataack: in std_logic;
--

selectline: out std_logic_vector(3 downto 0);
burstclk:out std_logic;
enable_a:out std_logic;
enable_b: out std_logic;
data_out0:out std_logic;
data_out1:out std_logic;
data_out2:out std_logic;
data_out3:out std_logic;
data_out4:out std_logic;
reset_out:out std_logic
);

end spicode;

architecture Behavioral of spicode is

signal datareg : std_logic_vector(39 downto 0):=x"0000000000";
signal shift_reg0: std_logic_vector(7 downto 0):=x"00";
```

```
signal shift_reg1: std_logic_vector(7 downto 0):=x"00";
signal shift_reg2: std_logic_vector(7 downto 0):=x"00";
signal shift_reg3: std_logic_vector(7 downto 0):=x"00";
signal shift_reg4: std_logic_vector(7 downto 0):=x"00";
signal clk_25MHZ,clki_25mhz : std_logic;
signal clk_burst: std_logic := '0';
signal enablecounter : std_logic_vector(3 downto 0) :="0000";
signal starttogglecounter : std_logic_vector(4 downto 0) :="00000";
signal enablesel : std_logic_vector(3 downto 0) :="1111";
signal enable_a_logic : std_logic := '0';
signal startsig, execute: STD_LOGIC := '0';
signal latch_s1,latch_s2, trigger : STD_LOGIC := '0';
signal qa,qb,qc,qd:std_logic := '0';
signal fire_sig : std_logic := '0';
signal enable_b_logic : std_logic := '0';
signal reset_sig : std_logic := '0';
begin

process (por)
begin
if (por = '1') then
datareg <=  x"0000000000" ;
end if;
end process;



------------------------- clock divide by 2------------
process(CLK,por)
begin
if(por='1')then
clk_25mhz<='0';
elsif(CLK ='1' and CLK'event)then
 clk_25mhz<=not clk_25mhz;
end if;

end process;

clki_25mhz <= not clk_25mhz;
```

```
----------execute----------

startsig <= not start;
process (clk_25Mhz,por,startsig,starttogglecounter,execute)
begin
if (por = '1' or starttogglecounter <= "11111" or startsig = '0') then
starttogglecounter <= "00000";
elsif (execute = '1')then
 if (clk_25Mhz'event and clk_25Mhz = '1') then
starttogglecounter <= starttogglecounter + "00001";
     end if;
end if;
end process;

process (clki_25Mhz,por,starttogglecounter,start,startsig,enable_b_logic,
reset_sig)
begin
if (enable_b_logic = '1' or reset_sig = '1') then
execute<= '0';
elsif (enable_b_logic = '0') then
execute <= startsig;
if(starttogglecounter <="11111") then
execute <= '0';
else
execute<='1';
end if;
end if;
end process;
-------------latch pulse generation----------------


process (clk_25mhz,clki_25mhz,startsig,qc,qb,qa,qd,por,execute)
begin

if (por = '1' or qd ='1') then
qa <= '0';
elsif (execute'event and execute='1') then
```

```
qa<= '1';
else
null;
end if;

if (por = '1' or qd ='1') then
qb<='0';
elsif (clk_25mhz'event and clk_25mhz='1') then
qb <= qa;
end if;

if(por='1')then
qc<='0';
elsif (clk_25mhz'event and clk_25mhz ='1') then
qc <= qb;
end if;

if(por='1')then
qd<='0';
elsif (clki_25mhz'event and clki_25mhz ='1') then
qd <= qc;
end if;

end process;
latch_s1 <= qa;
latch_s2 <= qb;
trigger <= qc;

--qa,qb,qc,qd <= '0';

---- 16 cycles counter for keeping enable_a high----

--process(clki_25mhz,enablecounter,latch_s1,latch_s2)
--begin
--
--IF (latch_s1'EVENT and latch_s1 = '1') THEN
----ENABLE_A_LOGIC <= '1'; ---- enable A signal  high---
```

```
--shift_reg4 <= datareg(39 downto 32);------------------ 32 bit data load
into shift register---------
--shift_reg3 <= datareg(31 downto 24);
--shift_reg2 <= datareg(23 downto 16);
--shift_reg1 <= datareg(15 downto 8);
--else
--null;
--END IF;
--
--IF (latch_s2'EVENT and latch_s2 = '1') THEN
--shift_reg0 <= datareg(7 downto 0);
--else
--null;
--END IF;
--
--end process;


process(por,clki_25mhz,clk_25mhz,enablecounter,enable_a_logic,trigger,startsi
g)
begin

if (por ='1') then
enable_a_logic <= '0';
elsif ( enablecounter <= "1000" or startsig = '0') then
   if(clki_25mhz'event and clki_25mhz ='1') then
   enable_a_logic <= '0';
     end if;
elsif(trigger'event and trigger ='1') then
enable_a_logic <= '1';
else
null;
end if;
end process;

process(por,clki_25mhz,clk_25mhz,enablecounter,enable_a_logic,trigger)
begin
```

```
if (por ='1')then
enablecounter <= "0000";
elsif(clk_25mhz'event and clk_25mhz ='1') then
if( enable_a_logic ='0' ) then
enablecounter <= "0000";
else
enablecounter <= enablecounter +"0001";
end if;
end if;

end process;


--- selection line ---
process(por,latch_s2,start)
begin

if (por = '1' or start = '1') then
enablesel <= "1111";
elsif (latch_s2'event and latch_s2 ='1')then
enablesel <= enablesel + '1';
end if;
end process;
selectline <= enablesel;


 clk_burst <= enable_a_logic and clki_25Mhz; -- burst clock is only high when
the enable 'a' is high--
 enable_a <= enable_a_logic; -- enable_a_logic signal connected to port--
 burstclk <= clk_burst; --- clk_burst signal connected to port--

------ serial data out ----

process
(clki_25Mhz,clk_burst,enable_a_logic,enablecounter,por,latch_s1,latch_s2)
begin

if(por='1')then
```

```
shift_reg0 <= x"00";
shift_reg1 <= x"00";
shift_reg2 <= x"00";
shift_reg3 <= x"00";
shift_reg4 <= x"00";
elsIF (latch_s1'EVENT and latch_s1 = '1') THEN
--ENABLE_A_LOGIC <= '1'; ---- enable A signal  high---
shift_reg4 <= datareg(39 downto 32);------------------ 32 bit data load into
shift register---------
shift_reg3 <= datareg(31 downto 24);
shift_reg2 <= datareg(23 downto 16);
shift_reg1 <= datareg(15 downto 8);
END IF;
IF (latch_s2'EVENT and latch_s2 = '1') THEN
shift_reg0 <= datareg(7 downto 0);
else
null;
end if;

--if(por='1')then
--data_out0 <= '0';
--data_out1 <= '0';
--data_out2 <= '0';
--data_out3 <= '0';
--data_out4 <= '0';
--end if;

if(enable_a_logic='1') then
if (clk_burst'event and clk_burst ='1')then

data_out0 <= shift_reg0(7);
shift_reg0 <= shift_reg0(6 downto 0) & shift_reg0(7);

data_out1 <= shift_reg1(7);
shift_reg1 <= shift_reg1(6 downto 0) & shift_reg1(7);

data_out2 <= shift_reg2(7);
shift_reg2 <= shift_reg2(6 downto 0) & shift_reg2(7);
```

```
data_out3 <= shift_reg3(7);
shift_reg3 <= shift_reg3(6 downto 0) & shift_reg3(7);

data_out4 <= shift_reg4(7);
shift_reg4 <= shift_reg4(6 downto 0) & shift_reg4(7);
else
null;
end if;
end if;
end process;


-- FIRE/ENABLE_B---
process (fire, fire_sig,enable_b_logic,clk_25Mhz,enablesel)
begin
fire_sig <= fire;
if (fire_sig = '1') then
     if (enablesel = "1111") then
     enable_b_logic <= '1';
end if;
end if;
     if (fire_sig = '0') then
enable_b_logic <= '0';
 end if;
enable_b <= enable_b_logic;
end process;


--- RESET---

process(reset_in, reset_sig)
begin

reset_sig <=  reset_in;

reset_out <= reset_sig;
end process;
```

```
---  Process for selection lines-- done
---- process for Data flag read--
--- reset--
--- enable'b'-- done


end Behavioral;
```