A Project Report

On

Toroidal Mode Computation in Real Time Using Field-Programmable Gate Arrays (FPGA)

Submitted in the partial fulfillment of requirement for the award

of Master of Science

Submitted by: Jalal F. Sulaiman

Advisor: Prof. Katherine Compton

University of Wisconsin-Madison

December, 2010

CONTENTS

III. References

List of Figures

List of Tables

1 Introduction

This report documents a project to create an FPGA-based hardware implementation of a matrix-vector multiplication procedure specifically designed for a particular physics experimentation system. This system requires that the matrix operation be performed in real-time to find the amplitude and phase of the input signals. This report explains the multiplication algorithm and how it is implemented in hardware. Also, it shows how the design can be used with other models in other applications.

1.1 Project Objective

Modern real-time applications with increasing computing power and design complexity have revolutionized embedded systems design. Furthermore, there is increasing demand for real-time signal processing algorithms in diverse areas such as audio processing, video, and medical imaging. The performance requirements and inherent parallelism of this and most other Digital Signal Processing (DSP) algorithms makes them ideal candidates for hardware implementation [1].

There are two popular technologies for implementing computation in real-time system on hardware. First is the Application Specific Integrated Circuits (ASICs) which are custom-designed for specific applications. The second technology is programmable logic such as Field Programmable Gate Arrays (FPGAs). Differences between these technologies will be discussed in section 3.

In this project, we develop a real time matrix multiplication system to analyze and calculate the amplitude and phase of particular electromagnetic fields generated by coils in a specific test system: the Madison Symmetric Torus (MST). These fields are known

as “toroidal modes”. This system assists physicists at the University of Wisconsin–Madison to examine magnetic modes. This design, which must be implemented within an existing FPGA system, will calculate parameters to be used for feedback during the experiment.

Currently in the MST, the electron temperature is measured during the plasma shot-which lasts for 70 milliseconds. However, the value of the electron temperature is evaluated after the shot is over. A real time system would help to determine the electron temperature in real time. Such a capacity can then be used for plasma feed-back.

1.2 Report Organizations

This report is organized as follows. Chapter two discusses the motive behind implementing the matrix multiplication algorithm on an FPGA. It states in detail the mathematical algorithm that the MST group uses to analyze the toriodal modes. It presents also the implementation of the technique in Matlab. Chapter three briefly explains FPGA architecture concepts. Chapter four includes a very detailed discussion of the hardware design of this project, including design implementation. The fifth chapter contains the results of simulation and synthesis using ISE. The last chapter concludes the project result analysis and presents future work.

2 Problem Statement

Magnetic fields in magnetically-confined plasmas consist of equilibrium and spatially-varying components. The equilibrium component is the axisymmetric part. The equilibrium magnetic fields, as well as other equilibrium plasma parameters, are independent of the toroidal direction. In addition to the equilibrium components, there are spatially-varying parts. In toroidally-confined plasma, the system is naturally periodic in the toroidal direction. The Fourier theorem allows us to expand the periodic function into sine and cosine functions with some arbitrary (user-chosen) amplitudes. These amplitudes and their associated periods are important quantities in the description of many plasma phenomena such as turbulence, magnetohydrodynamics instabilities, etc.

The structures (plasma shapes) can be measured with a magnetic sensor when those structures are rotating in the lab frame. If we place N magnetic sensors at nearly equal spaces in the toroidal direction to fully cover the circumference of the torus, we will be able to obtain a measure of the magnetic field at N locations around that circumference. Note that these sensors must be outside the plasma volume. These N measurements can be converted into an amplitude and phase of all Fourier modes from n = 0 to n = N/2 -1 as prescribed by Fourier and Nyquist theorem.

2.1 Spatial Fourier Decomposition Principles

The following describes briefly the experimental setup and how the data is measured from the toroidal array of magnetic coils in the Madison Symmetric Torus (MST). The MST has a set of 64 coil-locations, at each location there is a three-coil set: one coil measures the poloidal component, the second measures the toroidal component,

and the third measures the radial component of the magnetic field. These 64 coils are grouped into two groups. Each group consists of 32 coils that are equally spaced and has full coverage. These two groups are named as even coil-set and odd coil-set. Thus every set contains 32 coils. In MST only 32 coils are needed to fully resolve the spectrum without ant aliasing. Figure.1 shows the toroidal coordinate system used. Where θ is the coil poloidal angle and ϕ is the coil toroidal angle, r is the radial location of these coils.

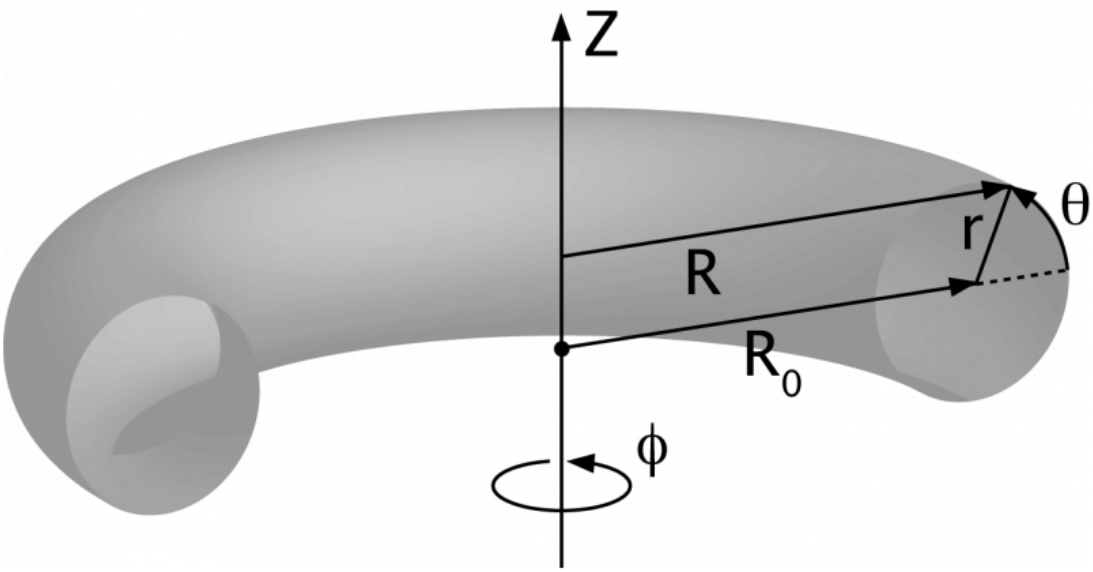


Figure 2.1: The toroidal coordinate system (r, θ, ϕ) [10]

In plasma physics the magnetic fluctuations are described in terms of their mode number content. Usually, “m” is used for the poloidal mode number (number of wavelengths in one poloidal circumference) and “n” is used for the toroidal number (number of wavelengths in one toroidal circumference). For a finite mode number spectrum, the spatial Fourier transform is used on a set of measurements at different toroidal locations to distinguish between different modes. In this experiment, all coils are at the same poloidal angle of 241 degrees and at the same radial location—at the vessel wall. Since the coil locations are equally-spaced and have full coverage in a periodic

device, then the measured magnetic field at N toroidal locations can be described using the spatial Fourier decomposition principle:

$$B(\phi) = \frac{a_0}{2} + \sum_{n=1}^{\left(\frac{N-1}{2}\right)} \left(a_n \cos(n\phi) + b_n \sin(n\phi) \right) + b_{\frac{N}{2}} \sin\left(\frac{N}{2}\phi\right) \tag{2.1}$$

The series sum is truncated at $n_{\max} = N/2 - 1$, where N is the number of coils in use. The above equation can be written for each coil at toroidal angle φ_i. With some algebraic manipulations these N equations with N unknowns can be written in the matrix form shown below [4].

$$\begin{bmatrix} B(\phi_1,t) \\ B(\phi_2,t) \\ . \\ . \\ B(\phi_N,t) \end{bmatrix} = \overbrace{\begin{bmatrix} 0.5 & \cos(\phi_1) & \cos(2\phi_1) & .... \cos\left(\left(\frac{N}{2}-1\right)\phi_1\right) & \sin(\phi_1) & \sin(2\phi_1) & .... & \sin\left(\left(\frac{N}{2}\right)\phi_1\right) \\ 0.5 & \cos(\phi_2) & \cos(2\phi_2) & .... \cos\left(\left(\frac{N}{2}-1\right)\phi_2\right) & \sin(\phi_2) & \sin(2\phi_2) & .... & \sin\left(\left(\frac{N}{2}\right)\phi_2\right) \\ & . & & & & & & \\ & . & & & & & & \\ 0.5 & \cos(\phi_N) & \cos(2\phi_N) & .... \cos\left(\left(\frac{N}{2}-1\right)\phi_N\right) & \sin(\phi_N) & \sin(2\phi_N) & .... & \sin\left(\left(\frac{N}{2}\right)\phi_N\right) \end{bmatrix}}^{A} \begin{bmatrix} a_0(t) \\ a_1(t) \\ . \\ a_{\left(\frac{N-1}{2}\right)}(t) \\ b_1(t) \\ b_2 \\ . \\ b_{\left(\frac{N}{2}\right)}(t) \end{bmatrix}$$

Figure 2.2: Magnetic field at N coils' locations

The left hand side represents the Nx1 input matrix *B*. On the right side is the NxN matrix *A* and the Nx1 output matrix *M*. Matrix *A* depends only on the geometry of the coils. This matrix is fixed once the coil locations are fixed. The output matrix *M* is describes the mode amplitude and phase, and contains the sine and cosine coefficients know as a_n's and b_n's. The mode amplitude is given by $c_n = \sqrt{a_n^2 + b_n^2}$ and the phase is given by $\Phi_n = a\tan(b_n / a_n)$. The output matrix *M* can be found using the equation:

$$M_{Nx1} = {}^{-1}A_{NxN} x B_{Nx1}$$

The input matrix B_{nx1} is measured at multiple time points during the plasma. The above procedure can be repeated at all time points to obtain the time evolutions of the mode amplitudes and phases. The multiplication procedure is shown below. The dot product matrix depends on time.

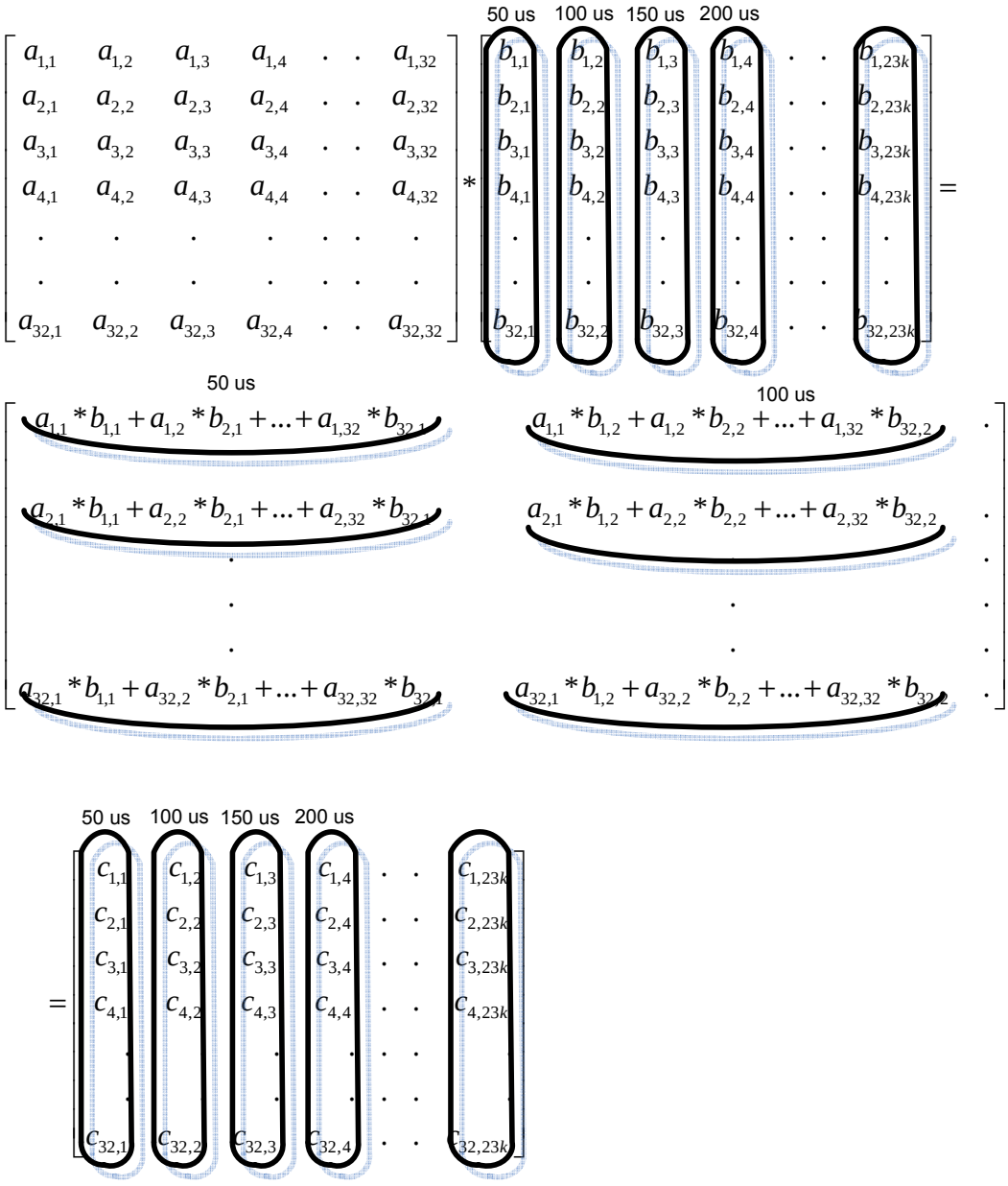


Figure 2.3: Multiplication procedure of matrix coefficients and magnetic field

2.2 Mathematical Model in Matlab

The mathematical magnetic mode analysis model based on the MST IDL code was implemented in Matlab. The purpose of the model is to calculate the toroidal mode structure of the magnetic fluctuations using toroidal field component and poloidal field component coils of toroidal array. The model is based on the n-equations, n-unknowns matrix method to find the sin and the cosine amplitudes. The model is tested using both synthetic and real data.

2.2.1 Model Testing on Synthetic and Real Data

The above technique (see figure 2.2) was first tested using synthetic data. A toroidal structure is assumed, then we calculate the magnetic field values at the toroidal locations of our magnetic coils. From these data points the matrix multiplication is made to calculate the mode contents of the magnetic data. If the methods are correct, the calculated mode structure should be the same as the initial structure. As shown in table 2.1, the initial structure consists of a cosine function of period of 8 cycles and amplitude of 15. Next the magnetic field at all the toroidal locations of the sense coils is calculated to determine the input matrix *B*. The matrix *A* coefficients are known since they only depend on the toroidal locations. The matrix *B* is multiplied by the inverse of matrix *A* and the results are compared with the original *M* values.

Table 2.1: Parameters that characterize the mode shape

Input	Values
Cosine cycles number, n	8
Cosine cycles number, m	0
Number of coils, N	32
Amplitude cosine wave, a	15
Amplitude sine wave, b	0

Table 2.2: Output Fourier coefficients

Expected Output Fourier Coefficients:	Values
Fourier coefficients (a and b) will be zero except	$a_8 = 15$

Figure 2.4 shows the magnetic field of the coils. This plot does not tell us the exact mode shape. If we connect the points in the plot, the mode shape can be a step function or triangle shape (or any shape). Therefore we plot the magnetic field values at different toroidal angles.

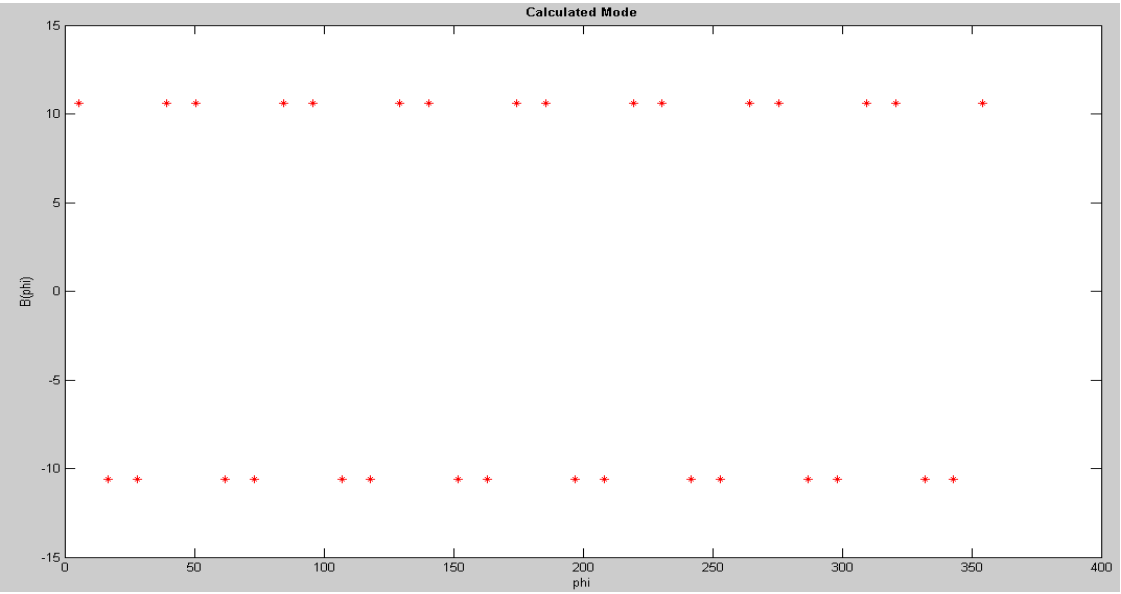


Figure 2.4: Calculated magnetic field values in different coil locations

In Figure 2.5, the blue color shows the magnetic field at all toroidal angles from 0° to 360°. The stars show the magnetic field values at the different 32 coils' locations. This plot also describes the entire mode characteristics. Table 2.3 shows the resulting *M* matrix whose all values are zeros except $a_8 = 15$, the same as the initial structure.

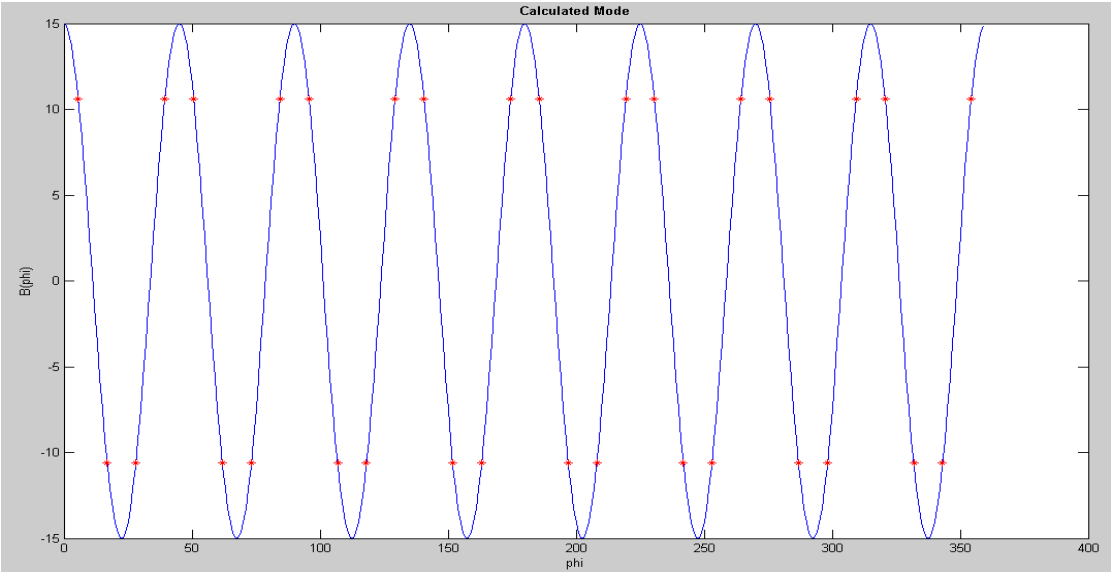


Figure 2.5: Calculated a cosine single mode

Table 2.3: Output of all Fourier coefficients

Cosine coefficients' values "a"	Cosine coefficients' values "b"
$a_0 = 0$	$b_1 = 0$
$a_1 = 0$	$b_2 = 0$
$a_2 = 0$	$b_3 = 0$
$a_3 = 0$	$b_4 = 0$
$a_4 = 0$	$b_5 = 0$
$a_5 = 0$	$b_6 = 0$
$a_6 = 0$	$b_7 = 0$
$a_7 = 0$	$b_8 = 8$

$a_8 = 15$	$b_9 = 0$
$a_9 = 0$	$b_{10} = 0$
$a_{10} = 0$	$b_{11} = 0$
$a_{11} = 0$	$b_{12} = 0$
$a_{12} = 0$	$b_{13} = 0$
$a_{13} = 0$	$b_{14} = 0$
$a_{14} = 0$	$b_{15} = 0$
$a_{15} = 0$	$b_{16} = 0$

This technique is also applied to real MST data, where the Matlab results were compared with the same results from the original implementation that used the Interface Description Language (IDL). The shape of the calculated amplitudes and phases using Matlab and IDL were identical without applying the normalization in both environments. The normalization procedure was not implemented in Matlab technique at this stage. Figure 2.5 shows the calculated amplitude of the mode after for a typical MST shot.

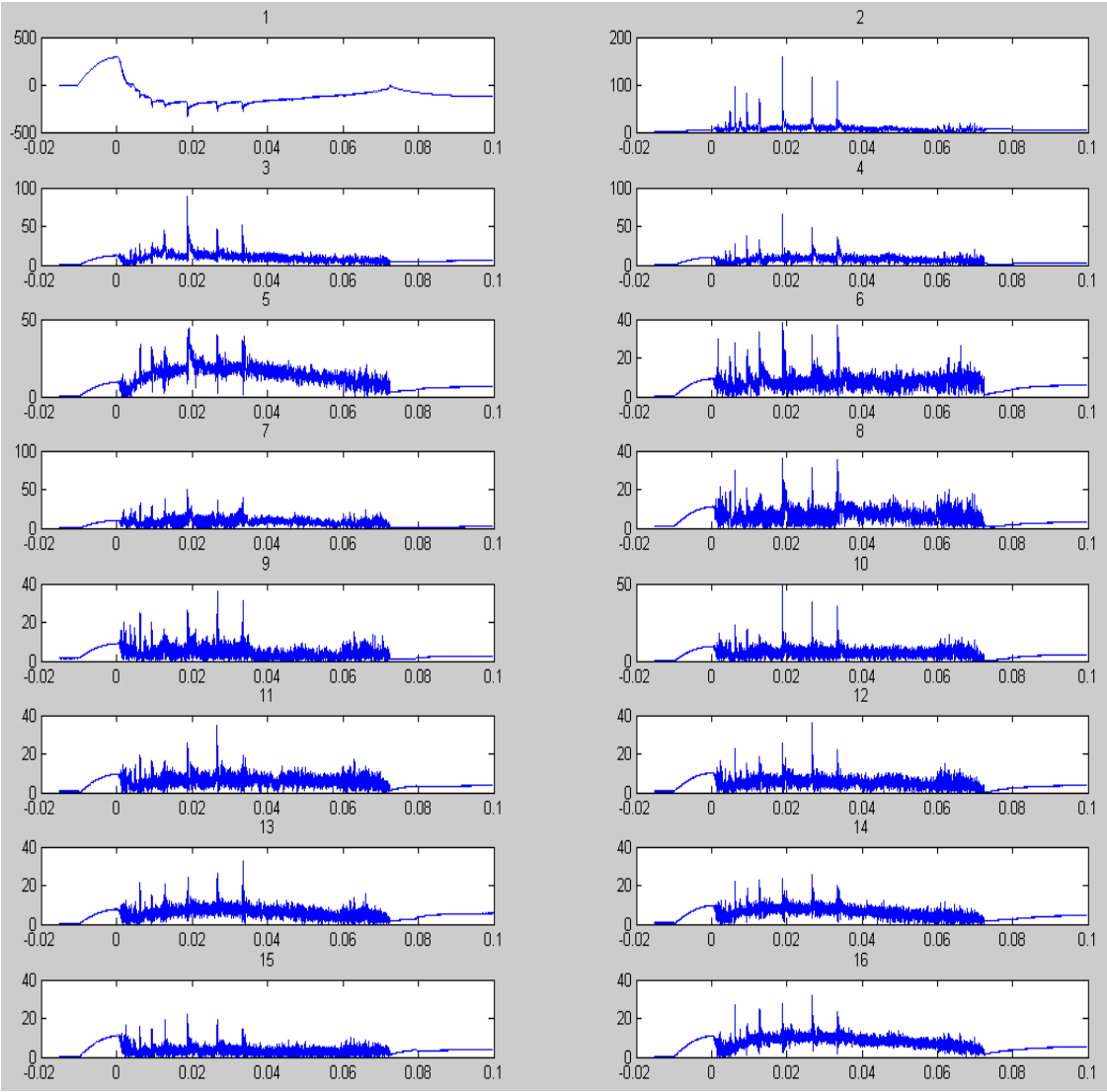


Figure 2.5: Mode amplitude found using Matlab

3 – D-TACQ Board

3.1 Board Specifications

The data acquisition board that is used in this experiment is ACQ196. It is designed by D-TACQ Solutions and has the following specifications:

- 16 bit 500 kSPS/channel
- 96, 64, 32 channels options
- 64bit/66MHz PCI interface to backplane
- High performance, low power microprocessor (400 MHz, XSCALE/ARM arch)
- Internal FIFO buffers in the FPGA
- The system is fully firmware controlled-both operating microprocessor and FPGA images are stored in flash memory and may be upgraded in the system without difficulty.
- Very high speed 200MHz DDR bus to local memory

The board uses a high performance Xilinx Spartan 3E FPGA to process the acquired data in real time. An array of hardware multipliers within the FPGA facilitate implementing different Digital Signal Processing (DSP) algorithms such as filtering. Figure 3.1 shows the DSP platform review of the data path on the board [5]. After the analog input data is digitized using the input digitizers at 200 kHz, it will be sorted in the "cold" FIFOs. Then a flag will be sent to the FPGA indicating that there is data available to start processing using the DSP algorithm. A subset of the calculated data will be sent to AO16 output of the system. More information about the inputs and outputs of the board can be found in the D-TACQ user guide [5].

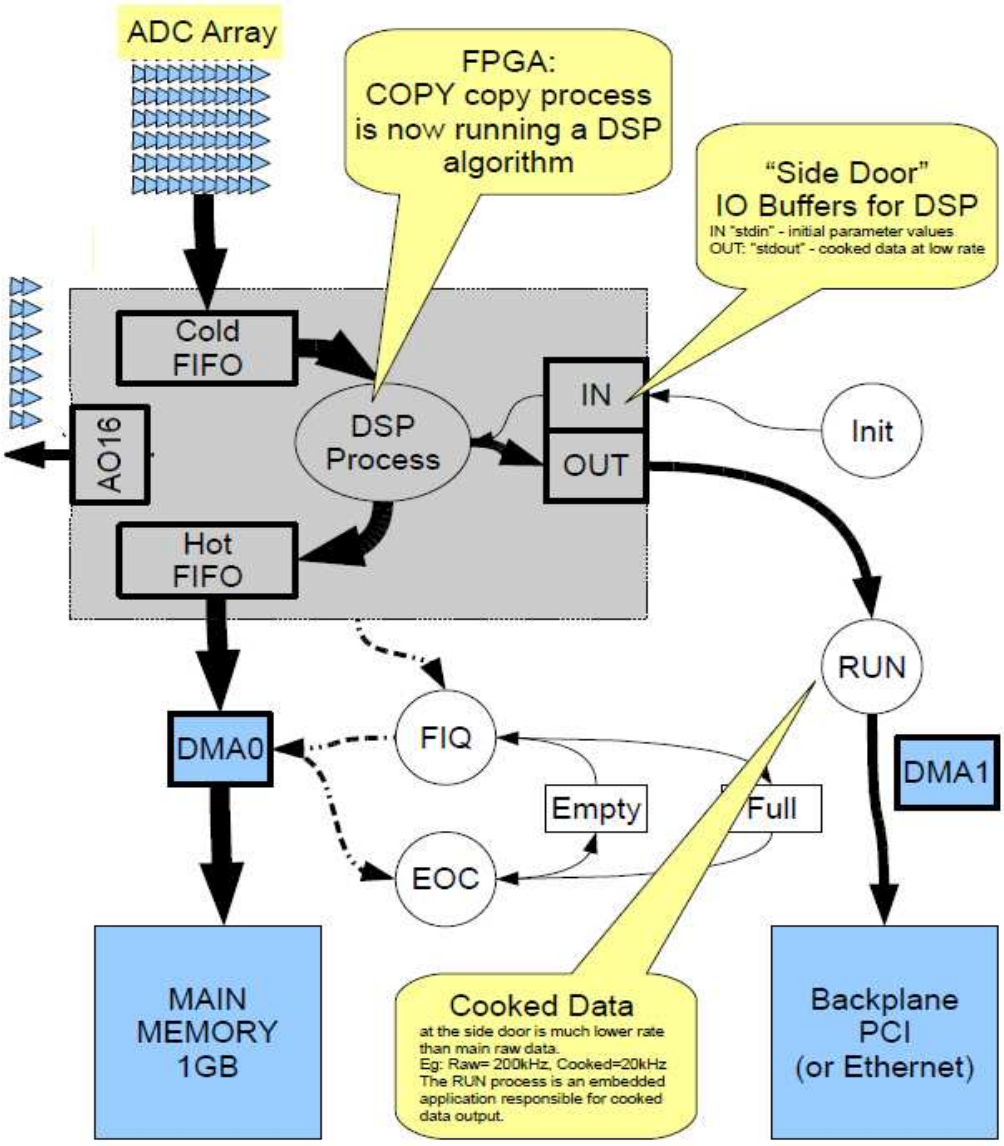


Figure 3.1: DSP Platform Overview [5]

3.2 Field-Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) represent one form of implementation platform for hardware-based computation. Unlike Application-Specific Integrated Circuits (ASICs), the functionality of FPGAs can be modified after fabrication to load new or modified hardware circuits [2].

Table 3.1: Comparison of ASICs, FPGAs, and microprocessors [1,2]

	ASIC	Microprocessors	FPGAs
Cost	High	Medium	Medium
Performance	High	Low	Medium
Flexibility	Low	Very High	High
Application Design Complexity	Very High	Low	High

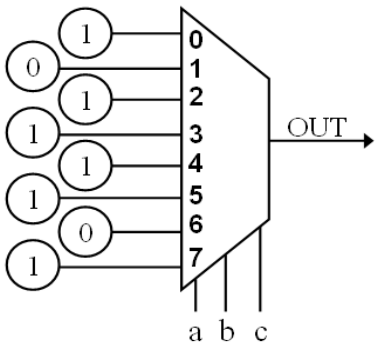
The high initial costs for designing and fabricating an ASIC make it expensive unless it is produced in large volumes, plus ASICs are inflexible. ASICs can only be used for a specific computation's algorithm that is specified before the designing process. Microprocessors are the other popular means for computing. They execute programs, composed of a series of small operations called instructions. However, a drawback of this technique is the time taken for reading each instruction form memory and decoding its meaning is overhead for execution of every instruction [2].

FPGAs not only have more flexibility than ASICs, but also have better performance than microprocessors for compute-intensive applications. Reconfigurable Computing devices have logic blocks which are different from one device to another. The example in Table 3.1 describes the implementation of a logical equation on a reconfigurable device. It shows how to implement this logical equation using LUTs which work as multiplexers to allow the choice from a number of possible inputs.

Assume we have the equation: OUT = ab + ac +bc [2, 3, 6]

Table 3.1: Look Up Table of the logic equation (left) and its implementation on Multiplexers (right) [2,3,6]

abc	OUT
000	1
001	0
010	1
011	1
100	1
101	1
110	0
111	**1**



These logic blocks are arranged in a grid, interconnected with a flexible (programmable) routing network, as shown in Figure 3.2. The actual logic blocks of an FPGA are far more sophisticated than the 3-LUT shown in Table 3.1. Furthermore, many FPGAs also include more coarse-grained hard computation cores such as memories and multipliers. Programmable input/output blocks connect the chip with external interface [2, 11, 3]. Loading a bitstream into the logic and routing locations in the FPGA configures that FPGA to implement different circuits. The devices can be reprogrammed with a new circuit as needed.

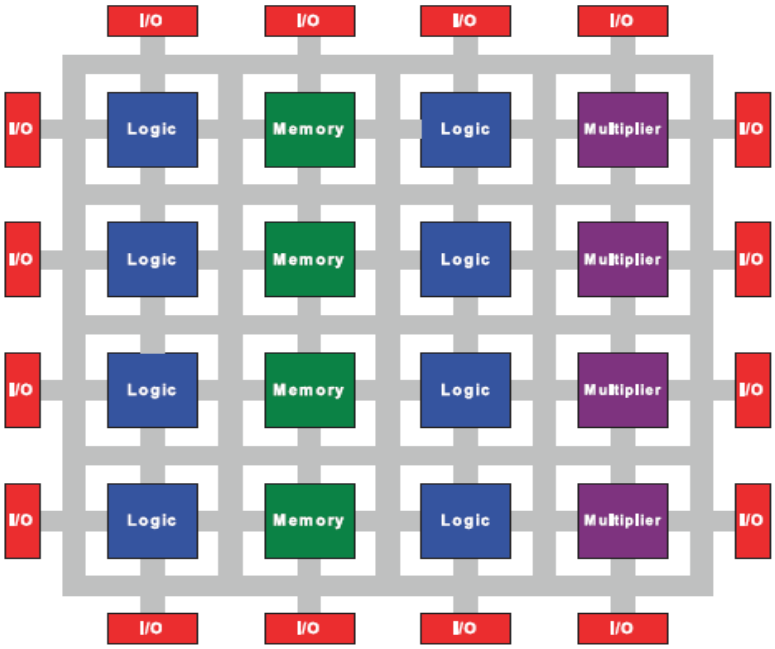


Figure 3.2: Basic FPGA structure [11]

4 Matrix Multiplication Hardware Design

4.1 Design Specification:

The main functionality of this design is to multiply a 32x32 matrix of coefficients with a 32x1 vector of data. The 32x32 matrix of 16-bit coefficients is stored in the FPGA's internal RAM blocks (but is treated as a Read Only Memory, or ROM). The 32x1 vector of 32-bit data values is received from First In First Out buffers (FIFOs) which store the data after it is received from the digitizer at a rate of 20 kHz. These FIFOs are considered to be an external module for the system presented in this document. The vector of input data is stored in a register file until the multiplication of the matrix with the vector is finished. The output dot products are stored in a Random-Access Memory (RAM) for later retrieval by another external circuit.

The D-TAQ board's Spartan 3E FPGA has 20 dedicated multiplier blocks, which operate at a maximum clock frequency of 270 MHz. Since the frequency rate of the vector input data is 20 kHz, one multiplier can be used 270/0.02 = 13,500 times in 50 microseconds if it is used at its maximum frequency. Multiplying a 32x32 matrix by 32x1 vector will need 1024 multiplications per 20 kHz. Thus one multiplier will be enough in the system to perform all needed multiplications.

There are two outputs of the system. One output is the multiplication result of one row of the matrix coefficients by the vector input data. The data width of the result is 128-bit. The second output is a flag signal to start writing the result to a text file. The system has two flag signals. A handshake is asserted from the external modules to indicate there is data available. The handshake output is used as an input signal to show that the system is done writing the results to the text file.

4.2 High Level Design Description

This section explains the design and how it can be used with other modules. It also provides an illustration of the design of the matrix multiplication on a small scale (a 4x4 matrix instead of 32x32, with a corresponding 4x1 input data vector). There are three main parts/stages of the design. The first stage registers the input vector data that is received from the three FIFOs. The current design registers the input data before the computation. When the computation unit is ready to take in data, it will send the read signal to the three FIFOs to start outputting data. The readInRF module keeps importing data from the FIFO as long as none of the ADC FIFOs is *empty* and *Busy* is low until the Register File (RF) is filled. Thus the design receives three inputs at every clock cycle. The signal *empty* is an output of the FIFO module and *Busy* is a signal that is generated in computation stage to indicate that a calculation is in progress

After the RF is filled with the input data, a control signal *InRdy* is sent to the next stage (the computation stage) to indicate that the input is ready and start computation. The multiplication of the input vector with the coefficients will proceed until the last dot product is stored in Random-Access Memory (RAM). For testing purposes, a signal is sent to the read out stage to indicate that the multiplication results are ready and they can be read and stored in an external file.

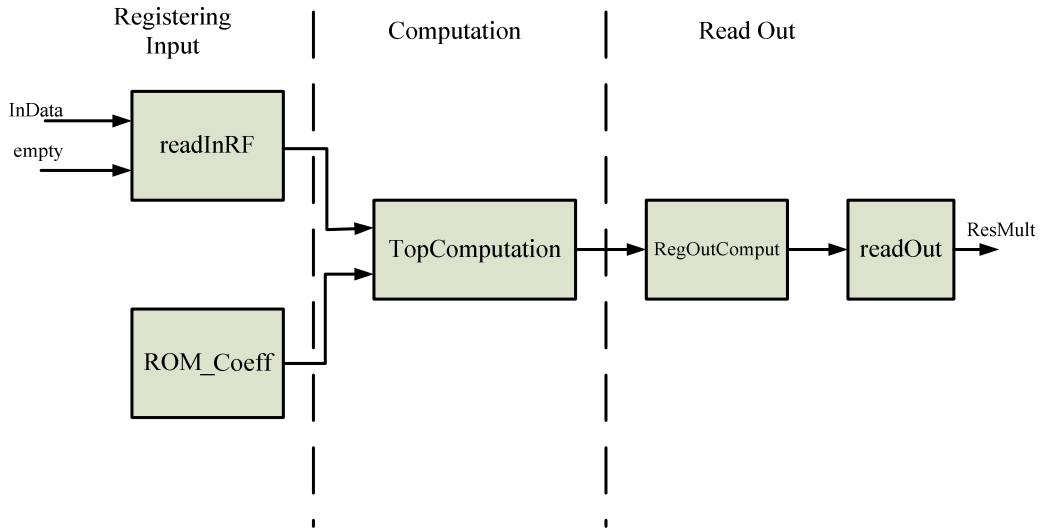


Figure 4.1: Block diagram of the matrix multiplication design

The number of cycles that the system needs to execute the multiplication is described as the following. With the assumption that there are three FIFOs that are used to fill the RF, if there are L locations in a RF, it will need C=L/3 cycles to fill the RF. C is rounded to the highest integer. For example, if there are 32 locations in the RF, the number of cycles that will be needed to fill the RF is 11 cycles. The following example illustrates the number of cycles that the system needs to produce the dot products and output the results.

First, we implemented the multiplication of matrix coefficients of size 4x4 with a vector data of size 4x1 which is a subset of matrix multiplication. The following example demonstrates the matrix multiplication procedure:

$$\begin{bmatrix} K_{A0} & K_{A1} & K_{A2} & K_{A3} \\ K_{B0} & K_{B1} & K_{B2} & K_{B3} \\ K_{C0} & K_{C1} & K_{C2} & K_{C3} \\ K_{D0} & K_{D1} & K_{D2} & K_{D3} \end{bmatrix} * \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} X \\ Y \\ W \\ Z \end{bmatrix}$$

$$X = K_{A0} * A + K_{A1} * B + K_{A2} * C + K_{A3} * D$$
$$Y = K_{B0} * A + K_{B1} * B + K_{B2} * C + K_{B3} * D$$
$$W = K_{C0} * A + K_{C1} * B + K_{C2} * C + K_{C3} * D$$
$$Z = K_{D0} * A + K_{D1} * B + K_{D2} * C + K_{D3} * D$$

The computation steps to produce one dot product X by multiplying the first row of the matrix by the vector data are the following:

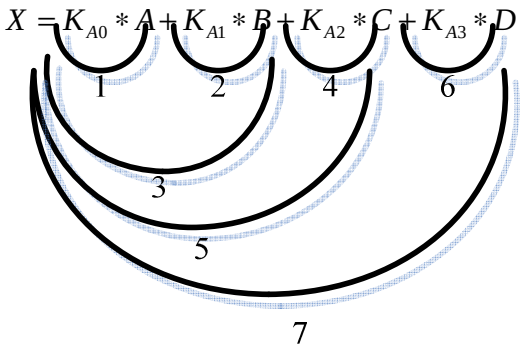


Figure 4.2: Multiplication steps of one row with the input vector

Step 1: Multiplying the first coefficient of the first row in the matrix with the first element in the vector

Step 2: Multiplying the second coefficient of the first row in the matrix with the second element in the vector

Step 3: Adding the multiplication results in step 1 and step 2

Step 4: Multiplying the third coefficient of the first row in the matrix with the third element in the vector

Step 5: Adding the addition result in step 3 to the multiplication result in step 4

Step 6: Multiplying the fourth coefficient of the first row in the matrix with the fourth element in the vector

Step 7: Adding the addition result in step 5 to the multiplication result in step 6

To produce the dot product X from, we needed four multiplications and three additions. The system performs one MAC pre cycle. It takes four cycles to produce one dot product (X.) In order to produce the four dot products (X,Y,W,Z), it will take 16

cycles. In general case, if there is a matrix coefficients of size NxN and a vector data Nx1, it takes N^2 to produce the dot products. The following diagram shows the multiplication procedure order of 4x4 matrix multiplication example [9, 12, 13].

$$X = K_{A0} * A + K_{A1} * B + K_{A2} * C + K_{A3} * D$$

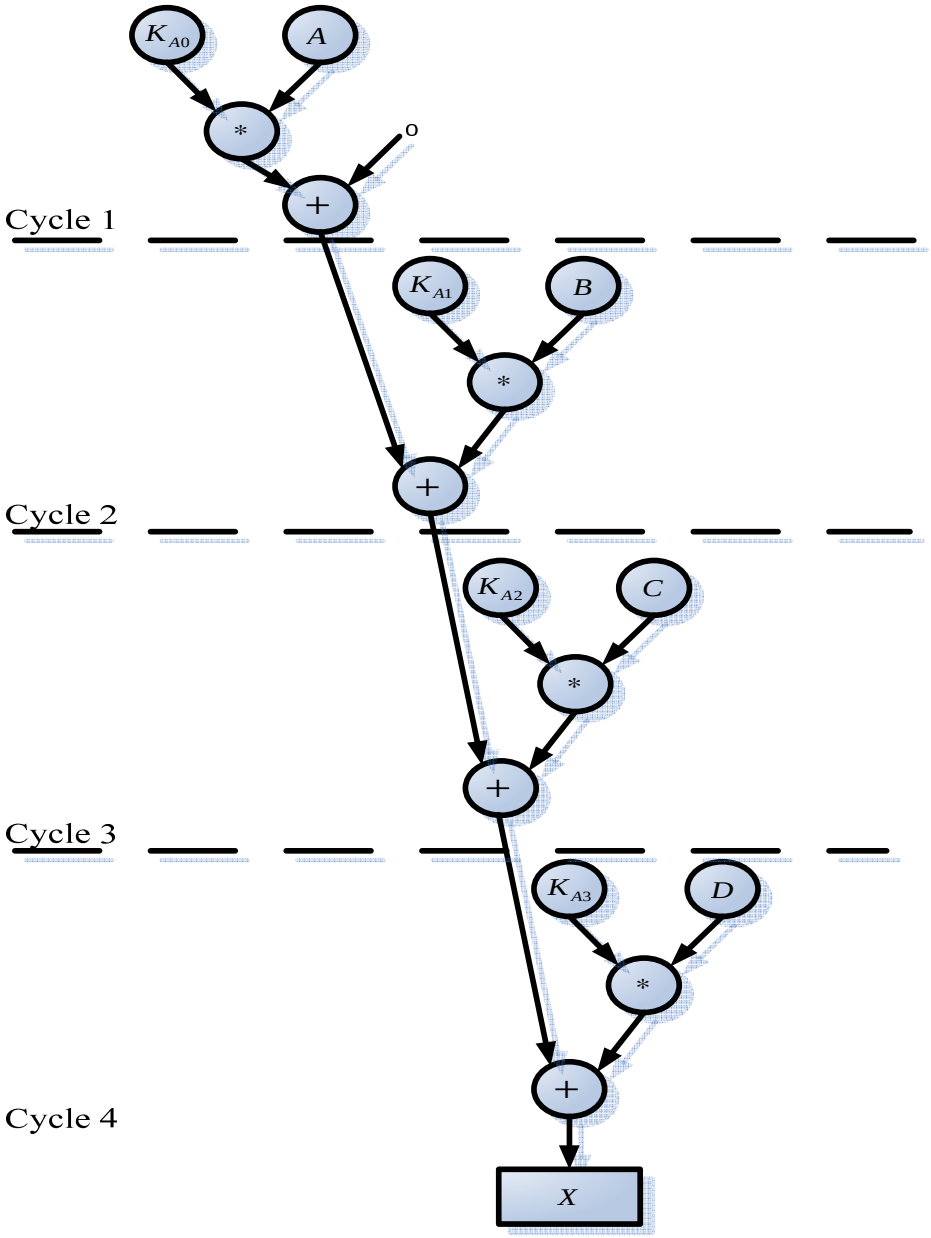


Figure 4.3: The number of cycles to produce dot product

Using the previous diagram as a guide to perform the order of multiplications and additions, the bit width of the dot product will be the following. Note that the final summation does not result in a bitwidth increase. This is because the total possible sum will not exceed 50 bits. In other words, if we used a summation tree, the C and D products would first be summed to create a 49-bit intermediate, which would then be added to the 49-bit sum of the A and B products, resulting in a 50-bit result.

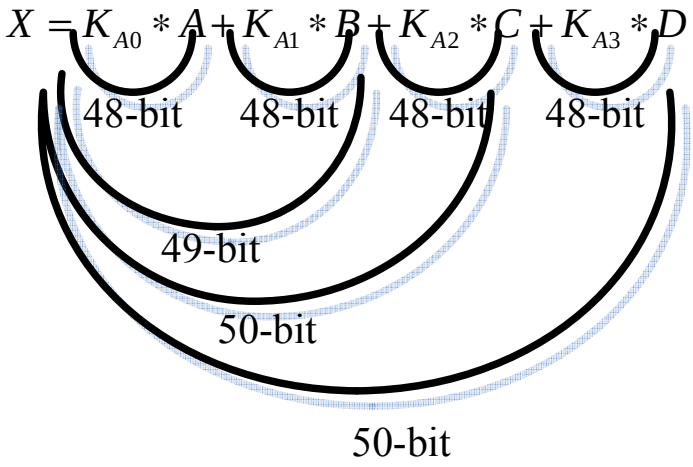


Figure 4.4: The bit width of dot product

4.3 Design Implementation Discussion

For an overview of the design, the following block diagram shows the breakdown of the main Verilog logic hierarchy.

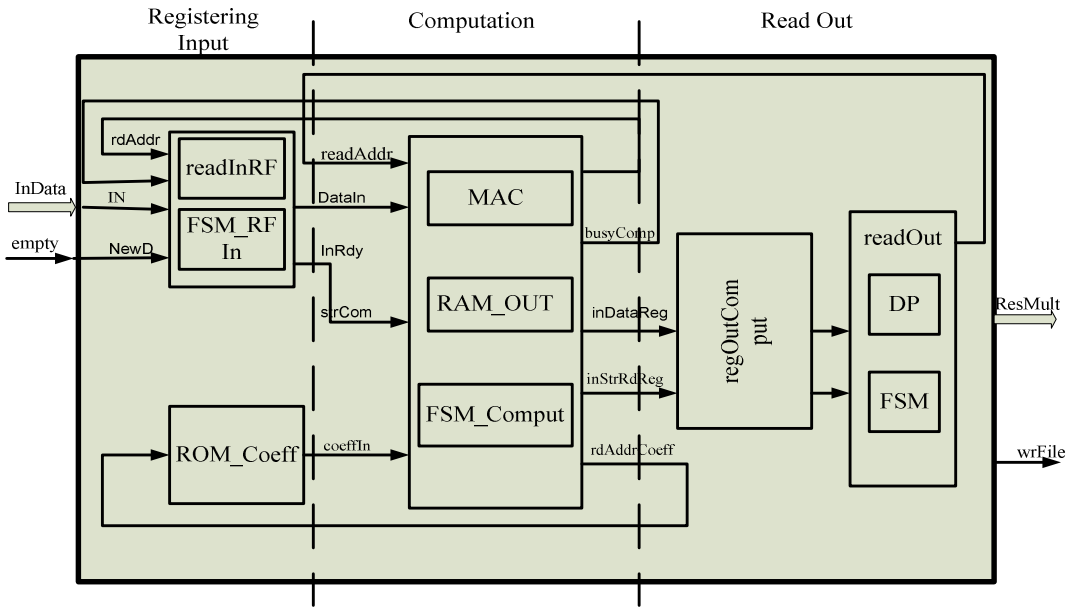


Figure 4.5: Detailed block diagram of the matrix multiplication design

As mentioned above, the design can be divided into three parts/stages: register the input, compute, and read out the results. Stage one contains two main modules: readInRF and ROM_Coeff. readInRF is connected with other external modules to import the input vector data. It has its own control system to manage storing the input data in the right location. It is also connected to the computation to send the data for multiplication. This module receives two control signals from the computation stage: rdAddr and busy Comp. The ROM_Coeff will send the coefficients to the next stage for computation.

The second main part in the design is the computation stage. The main module in this stage is called topComputaoin. This module communicates directly with the registering input stage and the read out stage. It has three internal modules that are connected to each other: MAC, FSM_Comput and RAM_OUT. Every module has its own

functionality. The MAC module multiples one matrix coefficients row by the input vector. The dot product will be stored in RAM_OUT. FSM_Comput controls the input data and coefficients that are selected for multiplication. Also, it assigns an address to store the dot product in the RAM.

After the computation stage finished multiplying the matrix coefficients by the vector input and storing the results in the RAM, the readout stage start reading the stored data for verification. This module has a counter to generate an address to read the RAM locations in the computation stage.

The following subsection discusses every module in detail by illustrating the input and output ports and brief description of the module functionality.

4.3.1 Parameter List

- COEFFSIZE 16 - Coefficient size is 16-bit
- INDATASIZE 32 - Input size is 32-bit
- OUTDATASIZE 128 - Output size
- NUMCHANNELS 32 - Number of input channels
- SIZEINPUTADDR 5 - Size input data address
- SIZECOEFFADDR 10 - Size ROM coefficients address
- SIZEOUTADDR 5 - Size RAM out address. This RAM stores the calculated results
- NUMMATRIXROWS 32 – The number of the matrix rows

4.3.2 Datapath

MatrixMult Module (top module)

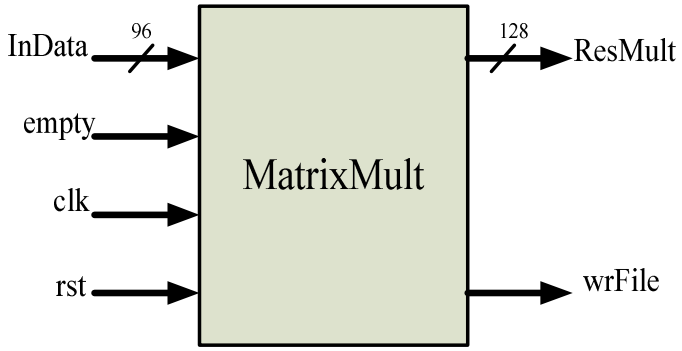


Figure 4.6: Top module interface

Module functionality:

This module is the top module that is used to instantiate the major sub-modules and to implement a matrix multiplication. A detailed description of this module was explained in the previous sections.

Input ports:

- InData - the port width is 96-bit. It is used to read the vector data from the FIFOs to store inputs values in the RF.
- empty – this is a control signal that is generated by the FIFOs module and indicates whether any of the FIFOs is empty.
- clk- input signal for the clock input
- rst- global rst

Output ports:

- ResMult – the post width is 128-bit. It is used to read the computation results that are stored in the RAM. It reads the result from every location in one cycle.
- wrFile – control signal to enable the text file for writing the read results.

readInRF Module

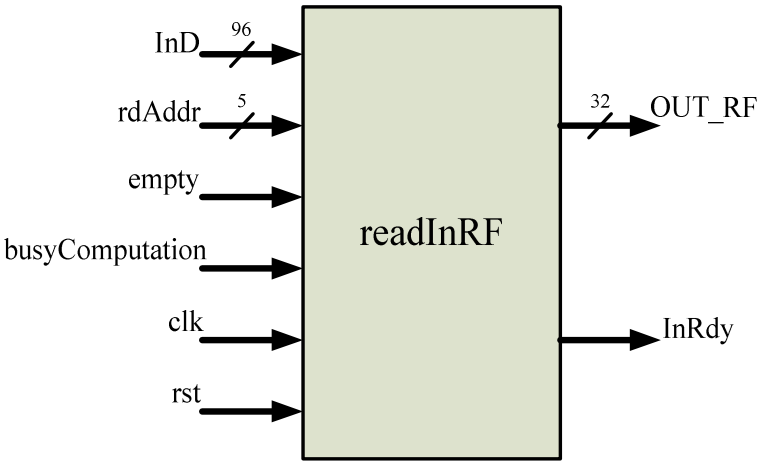


Figure 4.7: Input register file interface

Module functionality:

The main function of this module is to store the input vector data from the FIFOs. The data will not change until the entire calculations are completed.

Input ports:

- InD – this bus is used to write the input data from the FIFOs to the RF. It is wired with the input of the top module.
- rdAddr – the width of this port is 5-bit. It is used as an address to read from the locations of the RF. It is generated the computation module.
- empty - it is wired with the input signal of the top module.
- busyComputaion – this signal is generated from the computation module and indicates that the computation module is still busy computing the previous set of data

Output ports

- OUT_RF – the width of this port is 32-bit. It outputs the data from every RF location at every clock cycle.
- InRdy – a control signal that is generated to indicated that the RF is full and the next module is ready to start the computation.

ROM_Coeff module

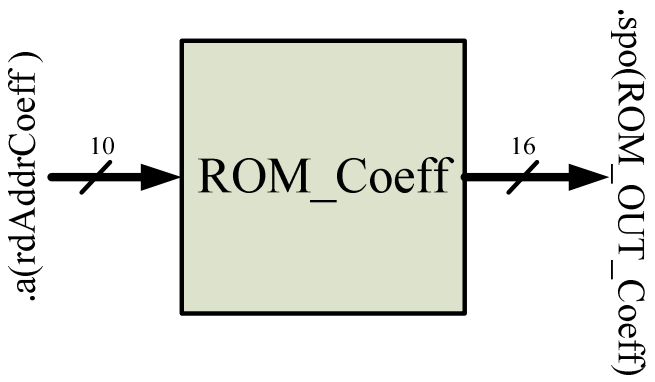


Figure 4.7: ROM interface using IP core generator

Module functionality:

It stores the matrix coefficients from a file.

Input ports:

- rdAddrCoeff – the width of this port is 10-bit. It is used to indicate what location should be read. It is generated in the computation module.

Output ports:

- ROM_OUT_Coeff – this port is used to output the coefficient to be sent to the next stage.

TopComputation Module

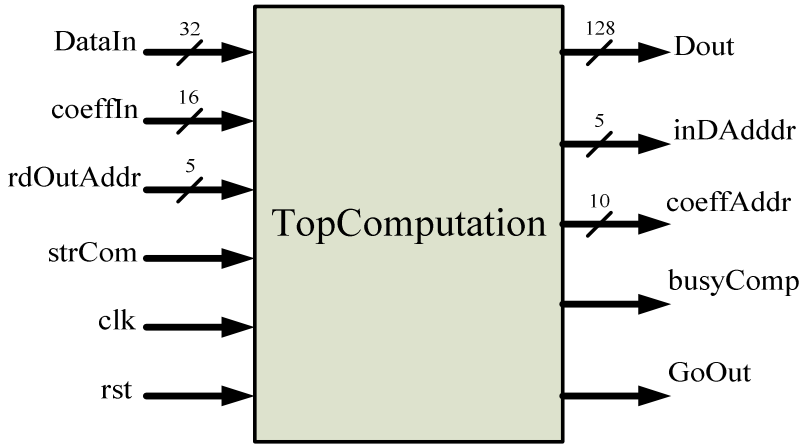


Figure 4.8: Top module computation stage interface

Module functionality:

This module is a wrapper module for three modules: MAC, RAM_OUT and FSM_Comput. All computations and storing results happen in this module.

Input ports:

- DataIn: 32-bit port is wired with the OUT_RF. This value is one input data from the RF
- coeffIn: 16-bit port is wired with the coefficient value from the ROM.
- rdOutAddr: This is 5-bit port to address the RAM locations where the multiplication results are stored. It is generated in the readout module.
- strCom: This control signal is wired with the signal InRdy from readInRF module. It indicates that the RF is full and the vector input data is available for computation.

Output ports:

- Dout: 128-bit port to read the multiplication results from the RAM.
- inDAddr: 5-bit port to address the locations of the RF where the vector input data is stored.
- coeffAddr: 10-bit port to address the locations of the ROM where the coefficients are stored.
- busyComp: This control signal is sent to readInRF module to indicate the computation is still in progress so it will not accept new input data from the FIFOs
- GoOut: This signal goes high when the computation is done and to start reading the computation results from the RAM. It is wired with the regOutComput module.

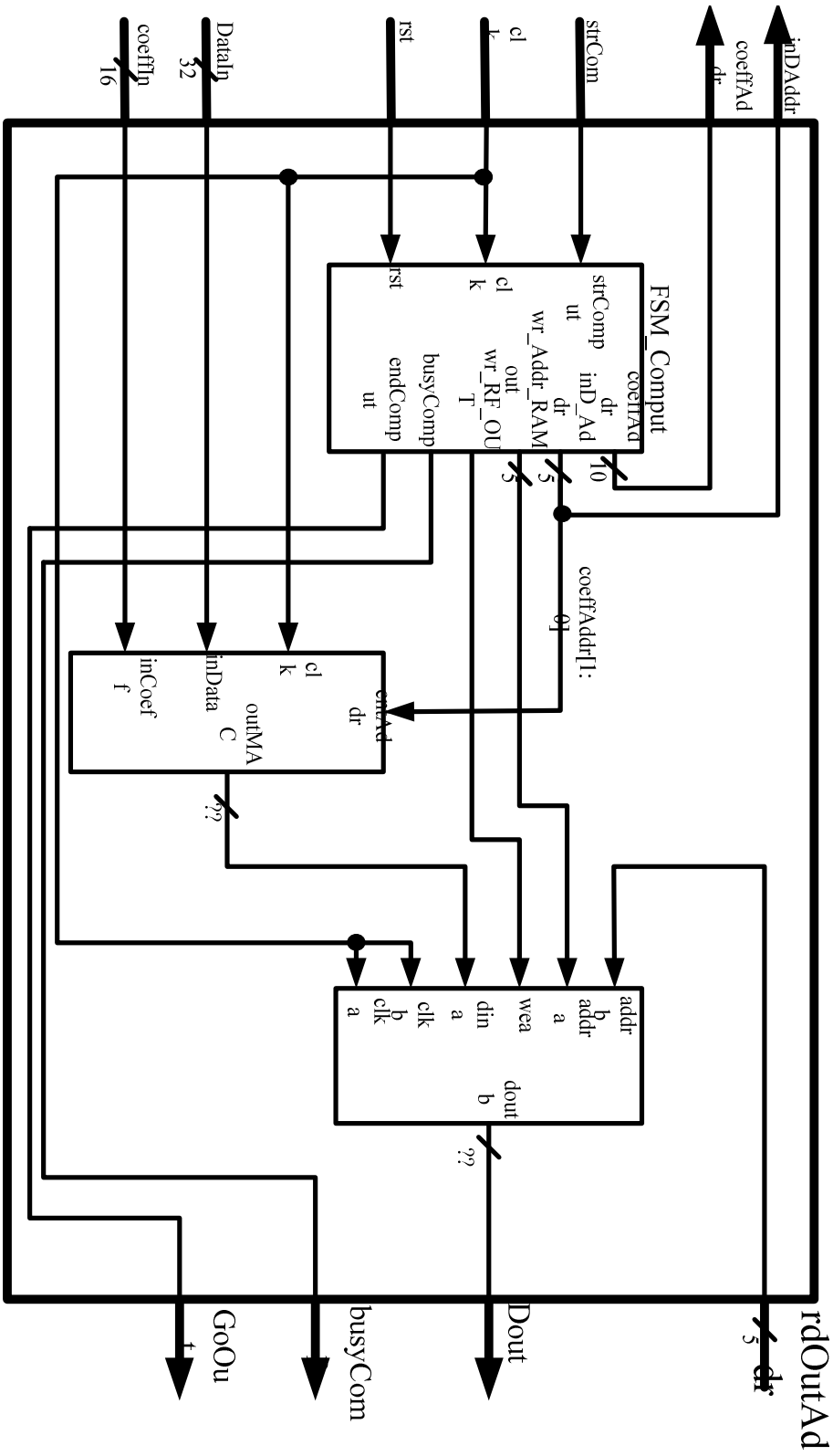


Figure 4.9: Detailed block diagram of the top computation module

MAC Module

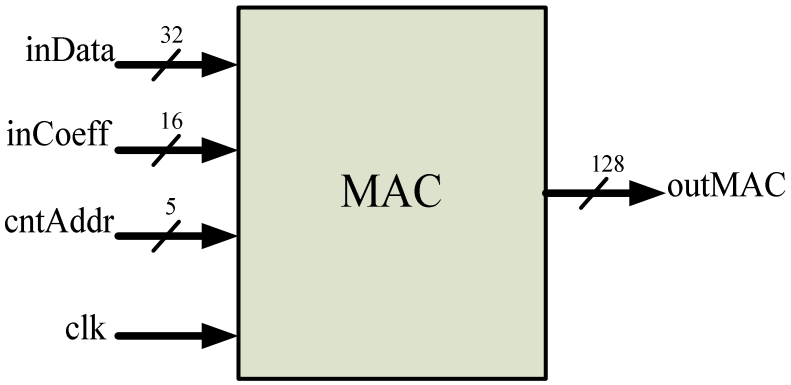


Figure 4.10: MAC module interface

Module functionality:

It computes the product of two numbers (one coefficient and a single data from RF) and adds the product to an accumulator.

Input ports:

- InData: 32-bit port is wired with the DataIn in topComputation module. This is the multiplicand
- inCoeff : 16-bit port is wired with coeffIn in topComputaion module. This is the multiplier.
- cntAddr: 5-bit port to the enable the mux in the mac unit.

Output ports:

- outMAC: 128-bit port to output the results of the matrix computation

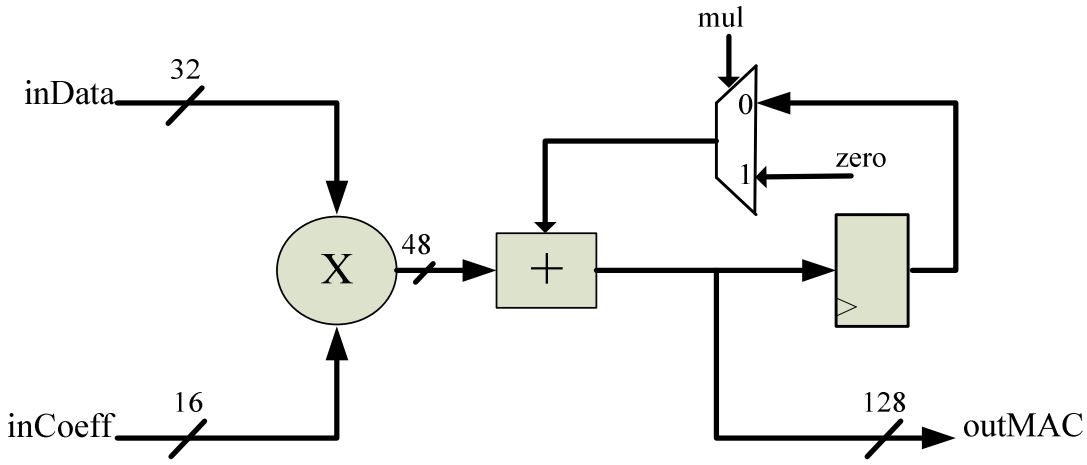


Figure 4.11: MAC block diagram with the bit width

RAM_OUT Module

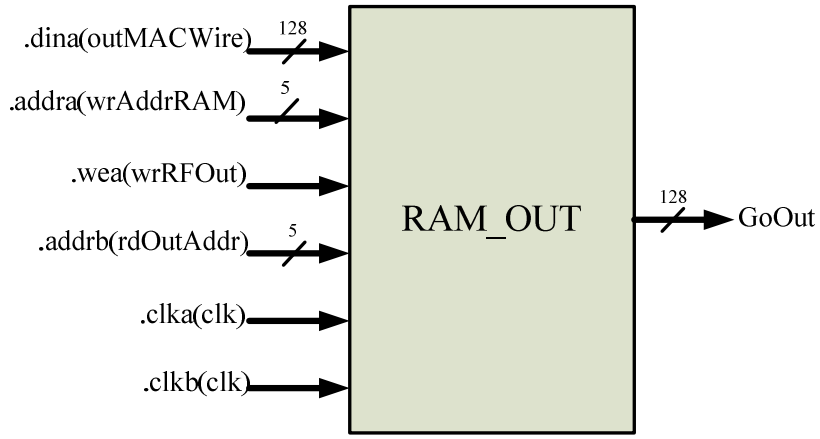


Figure 4.12: RAM IP core generator interface

Module functionality:

To store one set multiplication results

Input ports:

- dina – 128-bit port is wired with outMACWire to store the multiplication results
- addra – 5-bit port to address write RAM location
- wea – enable write
- addrb – 5-bit port to read RAM location
- clka and clkb – clocks to write and read. They are connected to the same clk

Output ports:

- GoOut – 128-bit port to output the results

regOutComput Module

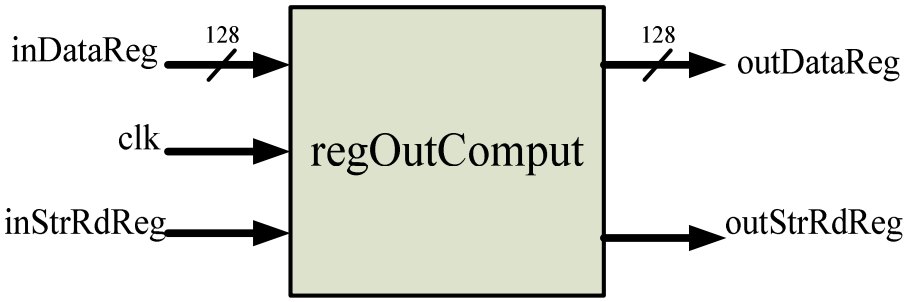


Figure 4.13: regOutComput module interface to pipleline the output

Module functionality:

To register the outputs of the TopComputation module before they are sent to readout module.

Input ports:

- inDataReg: 128-bit port is wired with the TopComputation output Dout. This is the result of the multiplication one matrix row with the input vector.
- inStrRdReg: a control signal is wired with GoOut that indicates that the computation is finished and the results is ready to be read.

Output ports:

- outStrRdReg- This is the control signal inStrRdReg with delay one cycle.
- outDataReg: 128-bit port to output the input with delay one cycle.

readout Module

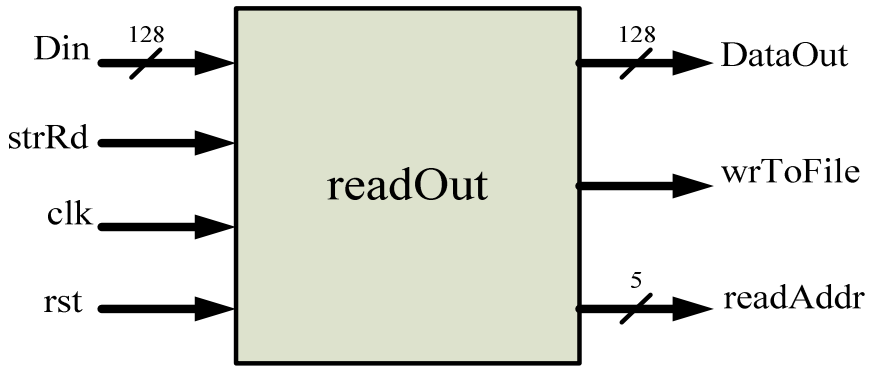


Figure 4.14: readOut module interface to read the outputs

Module functionality:

The main function of this module is to test the design by reading the stored computation results in the RAM. The output will be send to a file.

Input ports:

- Din – 128-bit port is wired with the output outDataReg of *regOutComput* module
- strRd – this is a control signal that trigger the readOut module to start reading the results from the RAM in topComputation mdule

Output port:

- DataOut – 128-bit port that outputs the computation results
- wrToFile- a control signal to start writing the output DataOut to a text file
- readAddr – 5-bit port is wired with the input rdOutAddr in topComputaion module to address the RAM locations to read the multiplication result.

4.3.3 Control

FSM_RFIn module

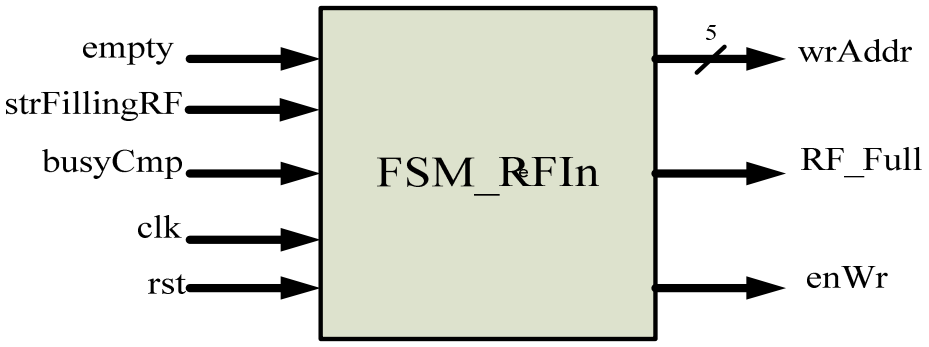


Figure 4.15: FSM_RFIn module interface to control the RF

Module functionality:

The purpose of this FSM is to control the filling process of the RF from external modules. In our case, three FIFOs are used to fill the RF.

Input ports:

- empty - this is a control signal that is provided by the FIFOs module to indicate weather the FIFOs are empty or not.
- busyCmp - this signal is generated in the topComputaion module to indicate if the computation is still in progress
- strFillingRF – this control signal is wired with the output signal doneRd (Add this signal to the design)

Output ports:

- wrAddr – 5-bit port to address the RF location to write the input data
- RF_Full – this signal indicates that the RF is full and next module should start the computation
- enWr- set signal high to start writing to the RF

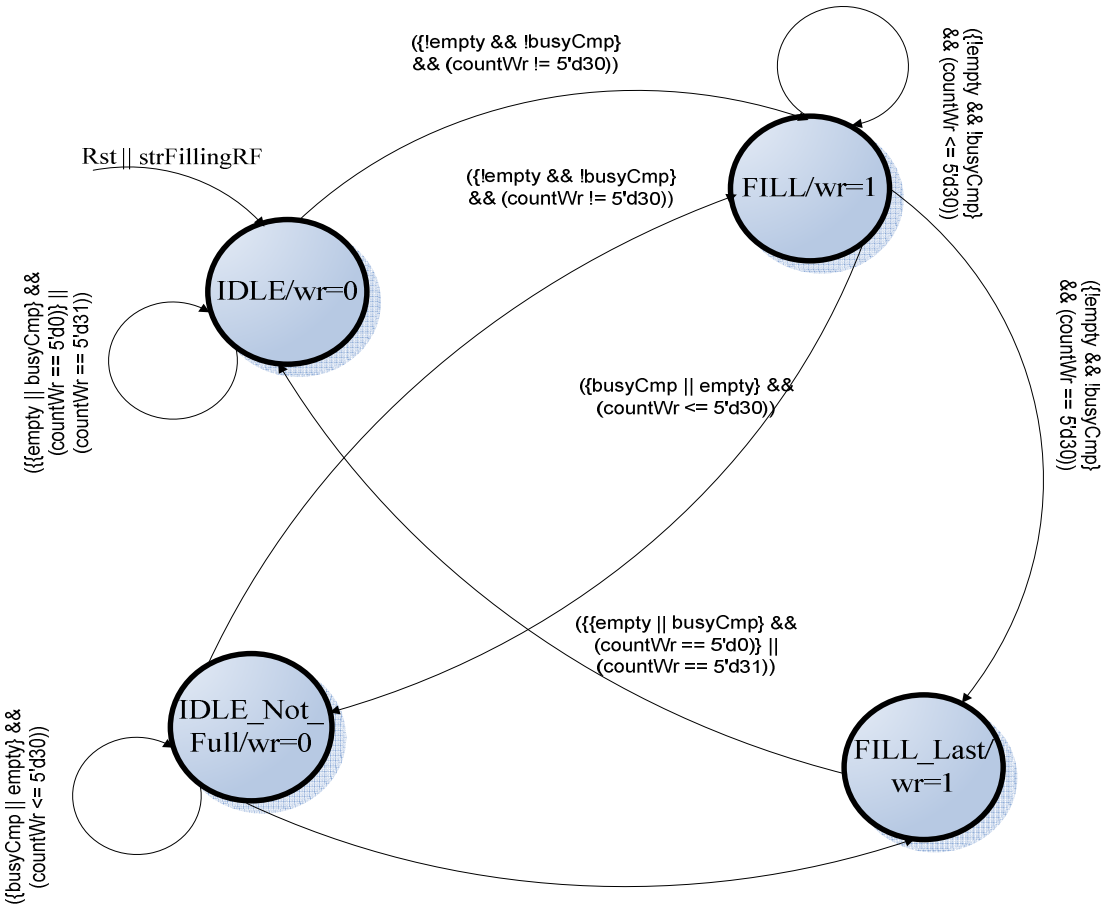


Figure 4.16: FSM_RFIn state diagram

State Diagram Description

This FSM controls filling the RF with the input data from the external world. If the rst or strFillingRF is high the next state will be IDLE and the wr (register file enable write signal) signal will be low. The state will be IDLE as long as empty or busyComp is high. In other words, the system will not be able to start filling the RF if the FIFOs are empty or the system is busy computing. The countWr will be zero while the current state is IDLE. This counter keeps track of the number of inputs that were imported from the external module and stored in the RF. In our case, the external module that our system will be connected to is a module that outputs the data from three FIFOs in parallel. Thus, the counter will be incremented by three at every clock cycle.

If the empty and bsuyComp are low, the next state will be transited from the current state (IDLE) to FILL state and the wr signal will be high. The current state will be FILL as long as the empty and busyComp are low and countWr is less than or equal to 30. If the empty or busyComp signals become high, then the state will be transited to IDLE_Not_Full and wr signal will be low. The next state will be FILL_Last if countWr is 30 and empty and busyComp are low. The wr signal will be high for one cycle and next state will be IDLE.

FSM_Compute module

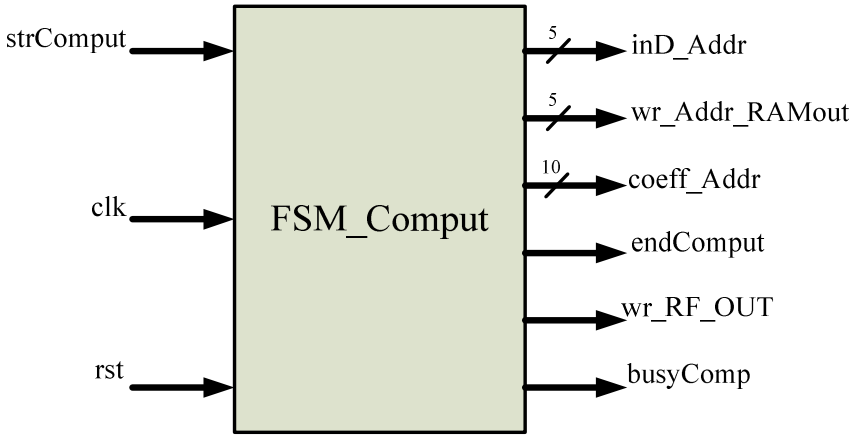


Figure 4.17: FSM_Comput to control computation stage

Module functionality:

This module controls the computation stage. It generates an address to read the stored data vector in RF and the coefficients in ROM. Also, it generates the address location to store the multiplication results in RAM. In addition, there are handshaking signals such as, endComput, busyComp and wr_RF_OUT that are produced in this module to control other modules. This module also produces the addresses to locate the selected coefficient in the ROM and the input data from the RF.

Input ports:

- strComput - this is a handshake signal that is wired with the output InRdy of the module readInRF. This signal triggers the computation module to start computing.

Output ports:

- inD_Addr – 5-bit port to address the RF locations to read the input data vector.
- wr_Addr_RAMout- 5-bit port to indicate the RAM location where the result should be stored
- coeff_Addr- 10-bit port to address the ROM locations to read the coefficients
- endComput- a handshake signal to indicate that computation is finished
- wr_RF_OUT-
- busyComp-the computation is till in progress

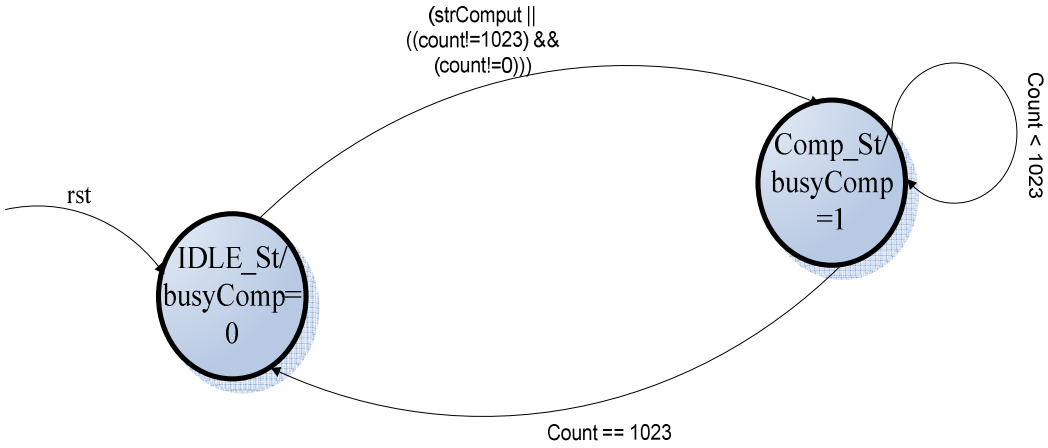


Figure 4.18: FSM_Comput state diagram

State Diagram Description

At rest the current state will be IDLE_St and the control signal busyComp is going to be 0. If the strComput signal is asserted and 0 < count < 1023, then the next state us Comp_St and the busyComp is 1. As long as the counter is less than 1023, there is not transition happens in states.

5 Testing and Results

The targeted device to implement the hadwared desing was a Spartan 3E. Thus, Xilinx tools were used to verify the design functionality. ISE 11.1 and ModelSim were used as an environment to run the functional and time simulations. Matlab was used to verify the dot product and also to generate the matrix coefficients.

5.1 Testing Methodology

As discussed above, the design is divided into three main parts: registering input, computation stage, and read out results. Every module in these stages was tested individually. Before proceeding in testing the entire design, every stage's functionality was verified. In the registering input stage, the modules were tested: readInRF, FSM_RFIn and ROM_Coeff. ROM_Coeff is module that was created using IP Core generator. A special attention was paid to test the FSM that controls the RF. The module was verified by filling a RF with known data and after reading the data from the RF locations. In the computation stage, since the FSM_Comput is responsible for managing the computation stage, it was tested exhaustively. It was verified that the FSM generates the reading inputs addresses and the writing address to the RAM location. The read out stage was tested to verify if the right address was generated to read from the right RAM location in computation stage.

The entire system was tested by verifying the small functionally design which is 4x4 matrix multiplication. First, a set of 16 random 16-bit numbers were generated in Matlab to form the 4x4 matrix. Random 32-bit 4x1 vectors were also generated to form the synthetic input data. The simulated hardware processed the data, and the resulting dot

products were stored in a text file. This multiplication procedure was performed analytically and the data is sorted in a text file. After the end of computation, the two files are compared to verify the results accuracy. The test bench generated 1000 input vectors and multiplied by the matrix coefficient and the results were compared. Using this test mythology, it was verified the small scale design accuracy.

5.2 Device Specifications

The current design targeted the FPGA device Xilinx Spartan 3E XC3S500E. The Spartan-3E family of FPGAs is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The following table describes the targeted device features [7, 8].

Table 5.1: Summary of Spartan 3E XC3S500E Features

Feature	Size	Feature	Size
System Gates	500K	Equivalent Logic Cells	10,476
Distributed RAM bit	73K	DCMs	4
Block RAM bits	360K	Maximum User I/O	232
Dedicated Multipliers	20	Maximum Differential I/O Pairs	92

Table 5.2: Configurable Logic Array in 3E XC3S500E Features (One CLB=Four Slices)

Feature	Size
Rows	46
Columns	34
Total CLBs	1,164
Total Slices	4,656

5.3 Simulation and Synthesis Results

Table 5.3: Matrix Multiplication Design Summary

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	1,185	9,312	12%
Number of 4 input LUTs	3,454	9,312	37%
Number of occupied Slices	1,874	4,656	40%
Number of Slices containing only related logic	1,874	1,874	100%
Number of Slices containing unrelated logic	0	1,874	0%
Total Number of 4 input LUTs	3,560	9,312	38%
Number used as logic	3,452		
Number used as a route-thru	106		
Number used as Shift registers	2		
Number of bonded IOBs	228	232	98%
IOB Flip Flops	128		
Number of RAMB16s	4	20	20%
Number of BUFGMUXs	1	24	4%
Number of MULT18X18SIOs	2	20	10%
Average Fanout of Non-Clock Nets	4.38		

Table 5.4: Timing Summary

Data path delay	35 ns (Maximum Frequency 28 MHz)
Slack	65 ns (15 MHz)

6 Conclusion

Matrix-vector multiplication is a part case of matrix multiplication which is a powerful tool in different Digital Signal Processing applications. Using this procedure with a large set of data can be a very intensive computing. Thus, implementing it on hardware is an optimal because of the parallelism property that distinguishes hardware from software. In project, a mathematical model to calculate the toroidal modes was developed and tested in Matlab. Part of this model was implemented on hardware.

The matrix-vector multiplication was designed and implemented on FPGA. The design contains three stages. First stage is the input stage to fill out the register file and ROM with input vector and the matrix coefficients. After the input data is ready a flag is sent to the second stage, computation stage, to start computing. The dot product is stored in a RAM until starts the next set of input data. The timing simulation shows that the design has met the specifications.

REFERENCES

[1]. Russell Tessier and Wayne Burleson. Reconfigurable Computing for Digital Signal Processing: A Survey. Journal of VLSI Signal Processing 28, 7–27, 2001

[2]. Katherine Compton and Scott Hauck. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Survey, 34(2):171–210, 2002.

[3]. Ian Kuon, Russell Tessier and Jonathan Rose. FPGA Architecture: Survey and Challenges. Foundations and Trends in Electronice Design Automation. Vol.2, No. 2 (2007) 135-253

[4]. Darren Craig. Magnetic Mode Analysis in MST. *University of Wisconsin – Madison, Physics Department, MST group. May, 2005*

[5]. D-TACQ User Guide. D-TACQ Solutions Ltd: http://www.d-tacq.com/

[6]. Dennis Silage, Embedded Design Using Programmable Gates Arrays. Bookstand Publishing, 2008.

[7]. Xilinx. Datasheet Spartan 3E: http://www.xilinx.com/support/documentation/spartan-3e.htm

[8]. Xilinx. Spartan 3E Starter Kit Reference Designs: http://www.xilinx.com/products/boards/DO-SPAR3-DK/reference_designs.htm

[9]. K.C. Bickerstaff, E.E. Swartzlander, Jr., and M.J. Schulte, "Analysis of Column Compression Multipliers", Proceedings of the 15th IEEE Symposium on Computer Arithmetic, Vail, Colorado, IEEE Computer Society Press, pp. 33-39, June 2001.

[10]. Wikipedia: http://www.fusion.ciemat.es/fusionwiki/index.php/Toroidal_coordinates

[11]. EE Times: http://www.eetimes.com/design/programmable-logic/4014815/All-about- FPGAs

[12]. C.S. Wallace, “A Suggestion for a Fast Multiplier”, IEE Transactions on Electronic Computers, EC-13, p.14-17, 1964.

[13]. Latha Pillai, Matrix Math, Graphics, and Vidio. Xilinx: XAPP284 (v1.1), October 2001