國 立 中 正 大 學

資訊工程研究所碩士論文

開發與實現可應用於巨量資料分析之
FPGA 硬體加速平台

Development and Implementation of an
FPGA-Based Hardware Accelerator for
Big Data Analysis

研究生 ： 劉俊凱
指導教授 ： 鍾菁哲 博士

中華民國 一零三 年 八 月

摘要

不論在圖片或影像的處理中往往伴隨著資料的運算與儲存，而當這些影像與圖片量大到某個程度使得運算量過大且難以管理這些資料時，我們將這樣的問題稱之為”巨量資料 (Big data) ”。

而當一般的個人電腦 (PC) 遇到 Big data 時往往面臨著執行時間過久或是使用過多的儲存空間等問題。在圖片或影像的處理的程式中，無非就是許多矩陣的運算，若我們將一些重複度很高的矩陣運算，例如:矩陣相乘...等等，切割至現場可編輯邏輯閘陣列 (FPGA) 上運算，這樣將可以省下許多的運算時間及暫存空間。

而將這些資料由 PC 端傳送到現場可編輯邏輯閘陣列 FPGA 端上儲存，然後等待 FPGA 運算完後，再將其結果再傳回 PC 這樣的一個環境便是本論文的重點之一，而我們將這環境稱之為現場可編輯邏輯閘陣列硬體加速平台(FPGA-based hardware accelerator) 或是現場可編輯邏輯閘陣列協處理器平台(Co-processor FPGA platform)。

放置在 FPGA-based hardware accelerator 上的應用便是本論文的另外一個重點，由於許多影像處理演算法會使用到矩陣相乘，例如: Bilateral filter 與 weighted least squares ，這兩者都是利用矩陣乘法來強化邊線輪廓外，還有著各式各樣的影像處理演算法都使用到，所以本論文利用硬體可以多套運算器的特性，設計一個能夠解決各種尺寸的浮點矩陣相乘電路，來分擔 PC 上執行大量矩陣乘法的負擔。

關鍵詞：巨量資料、現場可編輯邏輯閘陣列硬體加速平台、現場可編輯邏輯閘陣列協處理器平台、矩陣相乘在現場可編輯邏輯閘陣列。

Abstract

Dealing the image or video are often accompanied by data computing and data storage. When the number of dealing image or video is very large makes computing excessive and difficult to manage such condition, we call this problem is “Big data analysis”.

When the personal computer (PC) met big data problems, are often faced the execution time too long and use too much storage space problems. The processing program of image or video is usually include a lot of matrix operations. If we send matrix operations of the high repeat degree such as matrix multiplication to field programmable logic gate array (FPGA), it will save a lot of time and storage space.

We send those matrix data from PC to the DDR memory or FPGA evaluation board, and waits FPGA computing is done. Then the result is sent from FPGA back to PC. That environment is one of the priorities of this thesis. We call the environment is “FPGA-based hardware accelerator or FPGA Co-processor platform”.

The application which is running on the FPGA-based hardware accelerator is another priorities of this thesis. Since many image processing algorithms used to matrix multiplication to solve problem such as” Bilateral filter” and “weighted least squares”, those algorithm are used matrix multiplication to enhance the “Edge-Preserving”, so this thesis uses the feature of multiple sets of hardware to design a circuit of matrix multiplication of no size limitation. This thesis will use the FPGA-based hardware accelerator and application of matrix multiplication to accelerate and overcome the bottleneck of PC.

Index Terms —Big data, FPGA-based hardware accelerator, Co-processor FPGA platform, Matrix multiplication on FPGA.

致　　謝

首先，我要誠摯地謝謝我的指導教授—鍾菁哲博士—這兩年來的指導。從一開始什麼都不懂的暑期訓練，到現在研究有了小成果這一切都要感謝老師不斷地用心指導。在研究的路途中常常碰壁但老師卻從未放棄過我們，總在關鍵時刻輔助我們度過難關，希望以後到職場上我也能力回饋一點給我的老師給我的研究室。

第二，我要感謝我的父母。父母從小管教甚嚴雖然非常在意我們的課業但是卻不會限制我們想念的科系，只要是正當的事，就會讓我們依自己的興趣不斷努力向前，使得我有現在的一點點小成就。

第三，我要感謝我的朋友宜臻、維維與阿信在我研究有問題時毫不吝嗇的幫助我，沒有他們我的研究可能會花得更多時間。希望在未來繁忙的日子裡我們都不會忘記彼此，也祝福那些朋友不論是在工作或是課業上都可以順順利利。

最後 IC 設計這個領域是我選擇的也是我所喜歡的方向，我絕對不會半途而廢，我會堅持到底。

劉俊凱

中華民國一零三年八月

寫於國立中正大學資工所

Content

Lis of Figure

List of Table

Chapter 1 Introduction

1.1 Introduction to Big Data

Big Data is the term for a collection of data set too complex and large that it make difficult to manage, analyze and process using the traditional database system [1, 2]. Big Data include activity logs, business transaction, images, and surveillance videos that can reach massive proportions over time [1, 2]. In some statistics, those data generated exceed 2.5 quintillion bytes everyday [1].

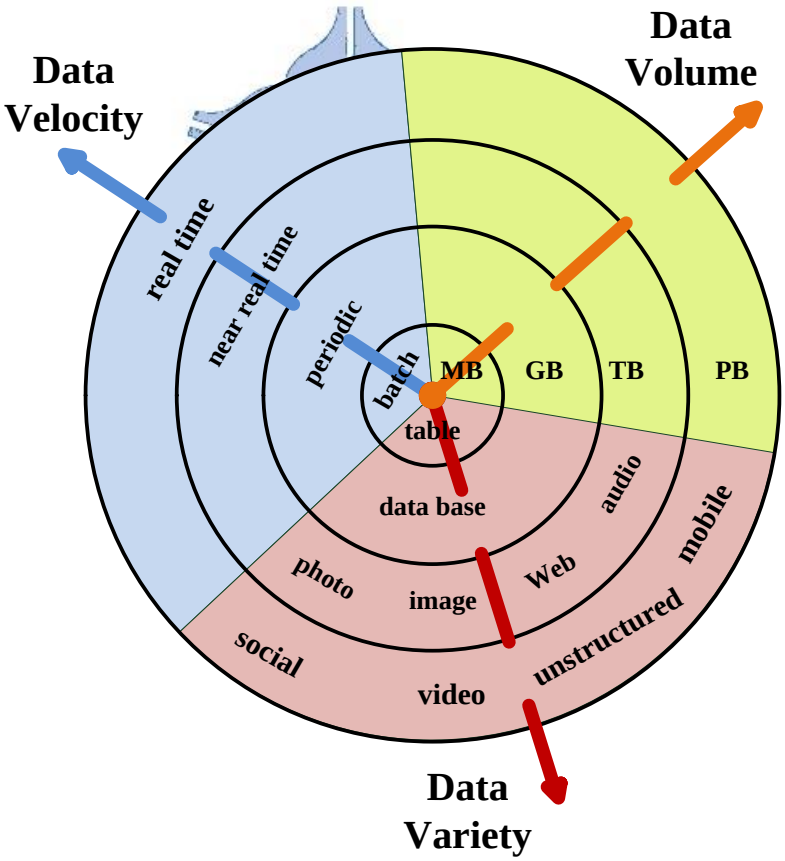


Fig. 1.1 3Vs Big Data models [3]

Fig. 1.1 shows the 3Vs Big Data models [3], it has three characteristics: volume,

variety and velocity.

Volume indicates the number of byte of data sets, we can see the exponential growth from inner channel to outer channel in Fig. 1.1. Image and photo are very easy to have terabyte level, video even reach the petabytes level. Table 1.1 shows the volume size of various data sets.

Table 1.1 Data set volume size [2]

Value	Abbreviation	Name
1000^1	KB	Kilobyte
1000^2	MB	Megabyte
1000^3	GB	Gigabyte
1000^4	TB	Terabyte
1000^5	PB	Petabyte
1000^6	EB	Exabyte

Velocity represents that how fast the data sets are generated. We can see velocity of data sets generation from batch of inner channel to real time of outer channel. According the reference [2], there are over than 328 million Google searches every day, 2 terabyte of photos uploaded to community websites every day, 14 million hours of video watched on YouTube every day. The velocity of data generated has gone beyond our imagination.

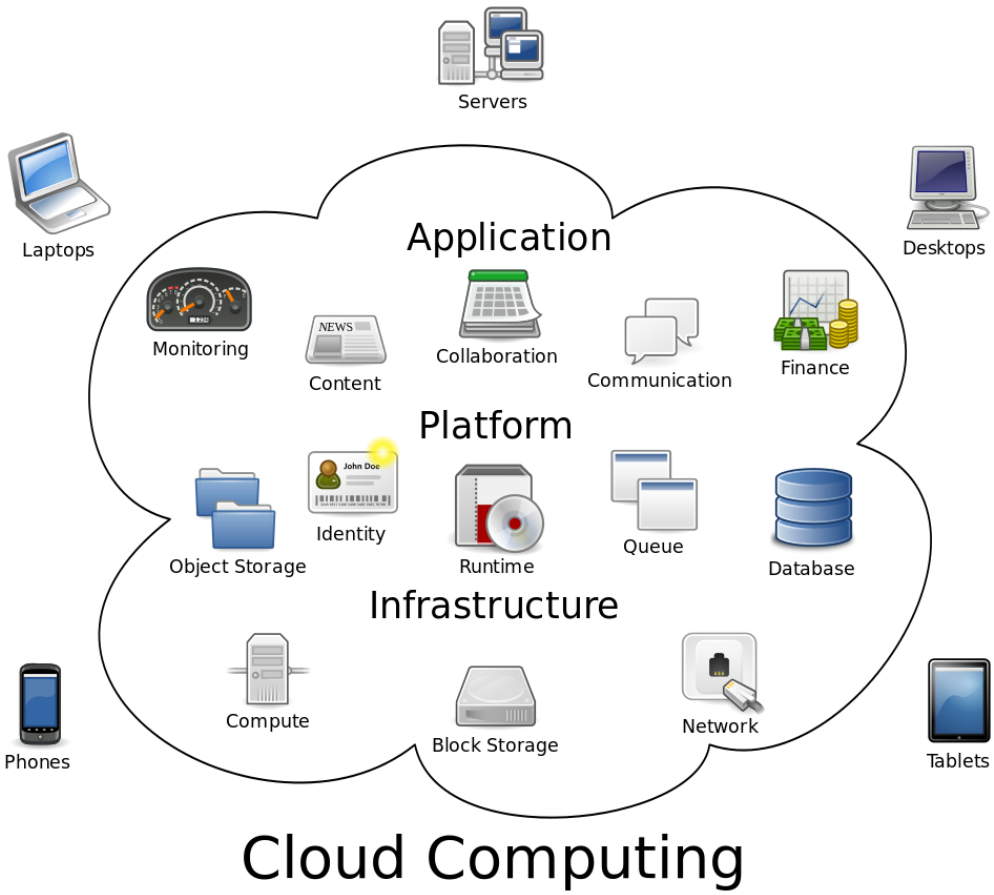
Variety represents that the data stored in multiple formats. There are several different formats in various applications. Some data can be stored in a text file, such as database and Excel, but most of data are not in the traditional formats, such as image, photo, video, audio and pdf. Those data must be organized, that make it meaningful. If the data all in the same format, it will be easy to do, but it is impossible. Hence, organizing the many different formats is one of the challenges with the Big Data.

Today, Big Data system has become an enterprises of technologies supplier, such as IBM, Yahoo and Facebook [4]. There are many small enterprises, who also want to use the Big Data techniques, but they don't have enough capital to do, Cloud-based Big Data processing is suggested by [4]. It has feature of elastic services framework, lets user in a pay-as-you-go manner [4]. Those small enterprises do not have to spend a lot of money to upgrade the hardware, they can spend those money in core of design and research. On the other hand, the user who want to use the Big Data techniques can readily obtain data and application what they want by environment of Cloud computing supplier [5]. Users also can accomplish the same work in different operating system.

Cloud computing can be express a group of physical machine servers, it provides services to users [6]. Services of Cloud computing is a platform of computing and management of data in physical machines servers. It usually presents a virtual network when user sends a service request. Virtual network is a set of virtual nodes and virtual links, it can be expressed a group virtual machine that mapping on the physical machine server. Fig. 1.2 shows the schematic view of Cloud computing.

In the above, brief introduction to the problem of the Big Data and Cloud computing to handle the Big Data problem. Although Cloud computing provides a powerful computing and management platform, but it has some problem. Problem 1:

Since user data must be placed on the Cloud computing servers, so users must take some risks that data maybe stolen. Problem 2: Not all users can afford the cost of Cloud computing services. Problem 3: If all of Big Data problem are using Cloud computing, that is too waste. In fact, above problems can occur in everyday life, for example we want to access the monitor videos and make it clearer by image processing. Assuming the amount of videos has Big Data level, it faced the execution time too long and use too much storage space problem for PC or server, and we can't put processing of videos immediately into Cloud computing serves to solve. There are many similar case in the life.



http://commons.wikimedia.org/wiki/File:Cloud_computing.svg?uselang=gan-hant#filelinks

Fig. 1.2 Cloud computing

There are some application for data analysis such as traditional database system and typical data mining. Traditional database system is not powerful for Big Data analysis. Due to database system is not storage independent, database system take too much storage space to construct data. In addition, database system uses a lot of loading and indexing during the processing data, database system take a lot of time to query data. There are many disadvantages of database system for Big Data. Typical data mining is not efficient for Big Data analysis. Typical data mining algorithms require all data that is loaded into the main memory, it maybe cause the memory full stage and take a lot of time to load data.

In Big Data analysis application, there are many applications such as Big Data mining and MapReduce. In reference [7] - [10], Big Data mining is not the same as data mining, big data mining has mechanism of information exchange and information integration. This mechanism will make sure that all information sources work together to achieve a global optimization goal. In reference [11] - [13], MapReduce is an effective tool for Big Data analysis, because MapReduce has unique features which include simplicity and communicative manners of its programming model. MapReduce has mainly two functions map( ) and reduce( ).Function of map( ) performs filtering and sorting, and function of reduce() that performs a summary operation.

Big Data is a significant problem in various application. One of Big Data challenge is Big Data analysis. Traditional database system and typical data mining are not enough to analyze Big Data. In recent, MapReduce replace the traditional database system and Big data mining replace the typical data mining, they are two kind of powerful analysis tool for Big Data.

According above case, we want proposed a new approach to solve Big Data problems. In this thesis, first we against the speed of data processing with Big Data problems to research a hardware accelerator.

1.2 Hardware Accelerator

In a general purpose processor, it usually can execute arithmetic unit at once, so it's sequential execution. Because processor is executed one by one, it takes some time to wait for previous execution end [14]. If a program has a lot of instructions, then the waiting time is considerable. Suppose we want to improve processor performance, we have to speed up the clock frequency, but speed up the clock frequency of the processor that is not cost-effective, and it has bottleneck on increasing the clock frequency.

Hardware accelerator can be used to implement by FPGA or application-specific integrated circuit (ASIC) [15]. Fig. 1.3 shows the Cyclone III FPGA development board. Fig. 1.4 shows the ASIC chip.

http://www.altera.com/products/devkits/altera/kit-cyc3.html

Fig. 1.3 Cyclone III FPGA Development Board

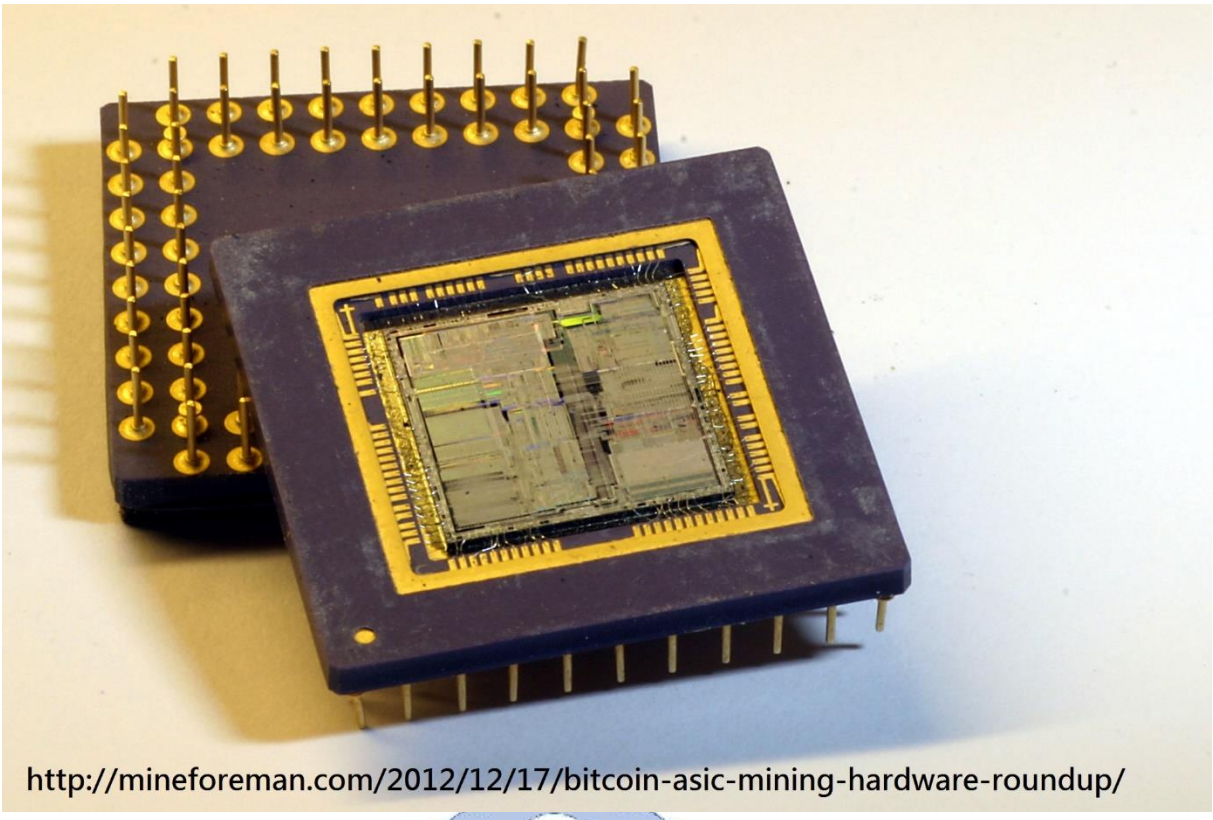


Fig. 1.4 ASIC chip

There are many techniques that can improve execution performance, hardware accelerator is one of them. The difference between software and hardware is concurrency. Due to we can design multi sets of arithmetic units in hardware, it does not take a long time to wait for previous execution end. The number of arithmetic units is limited by FPGA capacity or ASIC chip area.

Hardware accelerator is usually designed for computationally repeated or computationally intensive software code [16]. The application range of hardware accelerator can vary from a small functional unit to a large functional block, such as motion estimation in H.264. Now, hardware accelerator has been used widely in floating-point accelerator or graphics accelerator [17], etc. "Hardware accelerator" is an older term, and nowadays we call video or graphic card. Fig. 1.5 shows the graphic card [5].

http://www.bit-tech.net/hardware/graphics/2010/07/01/gigabyte-hd-5870-soc-graphics-card-review/1

Fig. 1.5 Graphic card

How to choose the development platform of hardware accelerator? Since the development of the accelerator is in the beginning, first we choose FPGA to develop our design, and finally we design ASIC until the development of design is almost done. In this thesis, our purpose is focus on the acceleration of data computing which is divided Big Data problems from PC into multi hardware accelerators, so that communication interface between PC and FPGA is very important. According above description, the first emphasis about choosing the development platform of hardware accelerator is FPGA board with networking. Why we don't use the Advanced

Microcontroller Bus Architecture (AMBA) FPGA, compiling the program into an ARM CPU on a Zynq evaluation board, then uses the AMBA to connect the FPGA and CPU. However, if we use the AMBA FPGA, then it is not possible to assign job to many FPGAs. The second problem, the clock frequency of an ARM CPU integrated in a FPGA is too slow as compared to Intel CPU used by PCs. The second emphasis about choosing the development platform of hardware accelerator is FPGA board with memory. The hardware accelerator in order to solve Big Data problem, so it must has memory space to store those huge data.

1.3 Matrix Multiplication on FPGA-Based Hardware Accelerator

1.3.1 Matrix Multiplication complexity

According to the above introduction of the Big Data, one of big data challenges are images or videos processing. Image or video processing is usually include many matrix operations. In various image processing algorithms, there many algorithms use the matrix multiplication to solve image processing problem, such as “Bilateral Filter,” ”Weighted Least Squares” and “Motion Estimation”. Two algorithms as we mentioned before are used matrix multiplication to enhance the “Edge-Preserving”. The third algorithms as we mentioned before is used matrix multiplication to find the vector of moving object in image. A matrix multiplication is repeated multiplications and additions, so we believe that matrix multiplication with a high order of parallelism. Fig. 1.3.1 shows the relationship chart of matrix order and number of multiplication and additions in matrix multiplication. As shown in Fig. 1.6, we can see that when the matrix size is twice bigger than the original matrix size, the number of multiplications and additions of matrix will be eight times than the number of multiplications and additions of original matrix, this growth rate is very fast. There are “x” matrixes of “n” order are multiplied continuously, we can deduce that have $(n)^3 * (x-1)$ multiplications and $(n)^3 * (x-1)$ additions. Whether “n” and “x” are fixed values to divide into four types of time complexity. Table 1.3.1.1 shows the time complexity of

matrix multiplication.

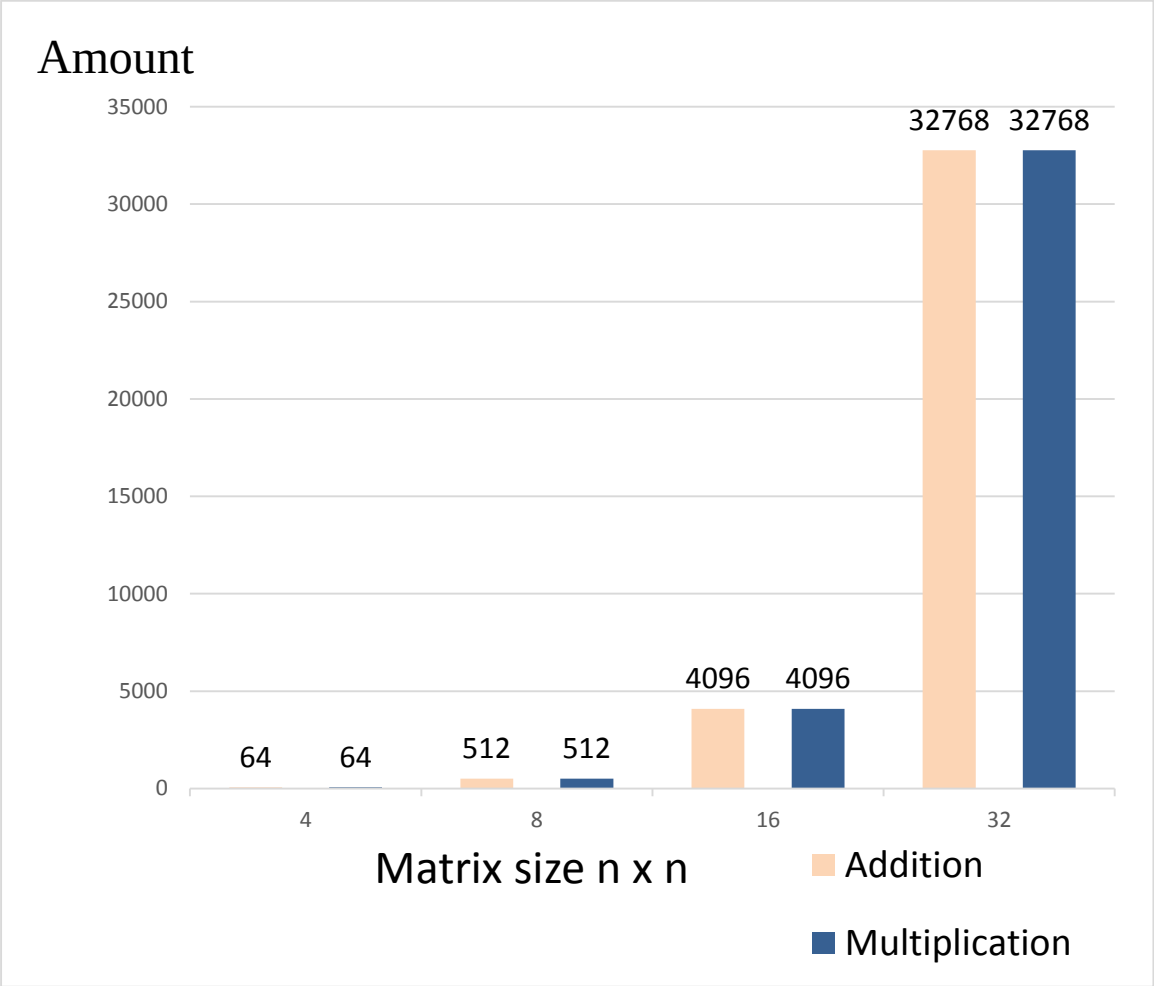


Fig. 1.6 Relationship chart of matrix size and number of multiplication and additions in matrix multiplication

Table 1.2 Time complexity of matrix multiplication

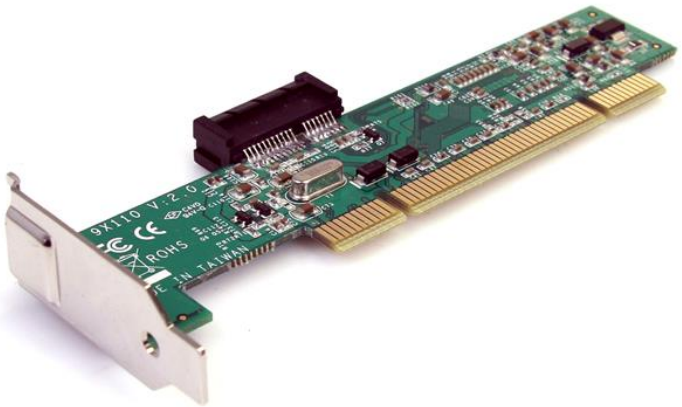
Matrix order n	Matrix number x	Time complexity
Fixed	Fixed	$O(1)$
Fixed	Non-Fixed	$O(x)$
Non-Fixed	Fixed	$O(n^3)$
Non-Fixed	Non-Fixed	$O(n^3 * x)$

When “n” or “x” is too large, there are huge operation of multiplications and additions. The huge number of multiplications and additions is definitely a considerable problem for PC. If we can divide the multiplications and additions into FPGA that compute multiplications and additions in parallel, then we can save a lot of time in matrix operation.

1.3.2 Related Work

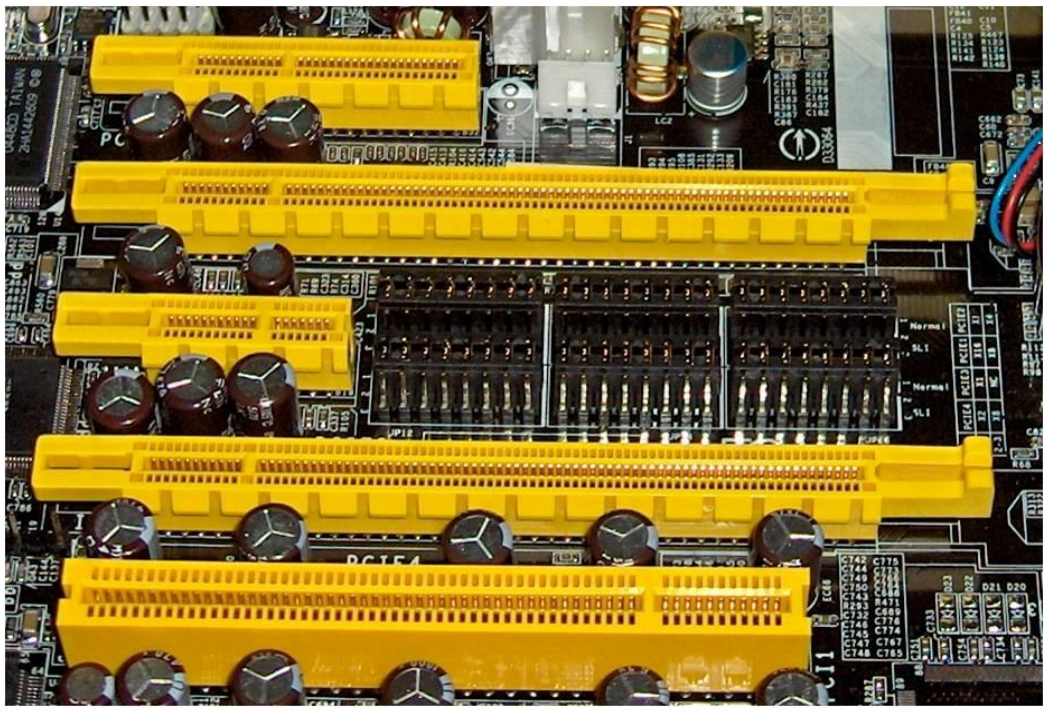
In the recent, there are several research that handle the problem of performing matrix multiplication in FPGA [18] - [20].

In the reference [18], it is use the peripheral component interconnect (PCI)-express 2.0 8 x endpoint to connect the PC and FPGA, its bandwidth is 4GB/s in two-way. Using the PCI as a communication interface between FPGA and PC is a great choice, because the transmission rate is higher than Ethernet. Although the communication interface of PCI has a fast transfer rate, but it has a disadvantage that PC has limited PCI slots, so the number of FPGA connected to PC is limited. Although communication interface of Ethernet has a slower transmission rate than PCI, it can connect number of FPGA is unlimited through Ethernet switch. Fig. 1.7 shows the PCI connector. Fig. 1.8 shows the PCI-express slot.



http://www.startech.com/Cards-Adapters/Slot-Extension/PCI-to-PCI-Express-Adapter-Card~PCI1PEX1

Fig. 1.7 PCI connector



http://gittagraciaevelyndiva.blogspot.tw/2014/01/kelompok-kapiten-patimura-12-ipa-2.html

Fig. 1.8 PCI-express slot

The reference [18] propose two architectures of floating-point matrix multiplication, it consists of a linear list of multiplier – adder processing elements (PEs). Fig. 1.9 shows the architecture "a" and "b" of PE. Let X and Y be two matrixes, that X with dimensions p × q and Y with dimensions q × r. The n is number of PE. When p, q, r ≥ n, matrix X consists of i × j and matrix Y consists of j × k blocks, where i = $\lceil p/n \rceil$, j = $\lceil q/n \rceil$ and k = $\lceil r/n \rceil$. (They pad zero on the under "n" part, like Formula 1.3.2.1.) The result matrix R with dimensions p × r, consists i × k blocks. The architecture "a" of PE is use the algorithm 1 to operation, and architecture "b" of PE is use the algorithm 2 to operation.

Algorithm 1: The PC consecutively sends the blocks of input matrixes X and Y which the order of sending is correspond with result block. For example, the result block $R_{11} = X_{11} * Y_{11} + X_{12} * Y_{21} + \cdots + X_{1j} * Y_{j1}$. It's totals is sending (2 × i ×

j × k) of input block. Since the algorithm 1 has a problem of sending many repeated matrix X blocks, author proposed the algorithm 2.

Algorithm 2: Let a row of result matrix R be a result unit. The PC consecutively sends the blocks of matrix X and Y which order of sending is correspond with result unit. Formula 1.1 shows the first row of result unit. In formula 1.2, it can be simplified into formula 1.3 it can only sent once of the same block of X, then according the order of formula 1.4 sent the matrix blocks of Y continuously, like $X_{11}Y_{11}Y_{12}\cdots Y_{1k},\quad X_{12}Y_{21}\cdots Y_{2k},\quad \cdots\cdots,\quad X_{1j}Y_{j1}Y_{j2}\cdots Y_{jk}$. When accelerator is completed a part of result unit, such as $X_{11}Y_{11}Y_{12}\cdots Y_{1k}$, it will be sent to the PC to store and waiting the other parts for adding. We can repeat above steps until the final part of result unit finished that is done of matrix multiplication. Totally, it sends (((1 + k) × j)× i) of input block.

$$
\begin{array}{l}
\overbrace{}^{n}\quad\cdots\quad\overbrace{}^{n} \\
n=\left\{\begin{bmatrix} x_{11} & & \\ & x_{22} & \\ & & \ddots \end{bmatrix}\ \cdots\ \begin{bmatrix} 0 & & \\ & \ddots & \\ & & 0 \end{bmatrix}\right. \\
\vdots \\
n=\left\{\begin{bmatrix} & & \\ & \ddots & \\ & & x_{pq} \end{bmatrix}\ \cdots\ \begin{bmatrix} & & \\ & \ddots & \\ & & 0 \end{bmatrix}\right. \\
n=\left\{\begin{bmatrix} 0 & & \\ & \ddots & \\ & & 0 \end{bmatrix}\ \cdots\ \begin{bmatrix} 0 & & \\ & \ddots & \\ & & 0 \end{bmatrix}\right.
\end{array}
$$

Formula 1.1 Padding 0 on under n parts

$$Result\ unit:$$

$$\begin{bmatrix} R_{11}, & R_{12}, & \cdots, R_{1k} \end{bmatrix} = \begin{bmatrix} X_{11} * Y_{11}, & X_{11} * Y_{12}, \cdots X_{11} * Y_{1k} \\ + & + & + \\ X_{12} * Y_{21}, & X_{12} * Y_{22}, \cdots X_{12} * Y_{2k} \\ + & + & + \\ \vdots & \vdots & \vdots \\ + & + & + \\ X_{1j} * Y_{j1}, & X_{1j} * Y_{j2}, \cdots X_{1j} * Y_{jk} \end{bmatrix}$$

Formula 1.2 Result unit

$$\begin{aligned} & X_{11} * \left[Y_{11}, Y_{12}, Y_{13}, \cdots, Y_{1k}\right] \\ + & X_{12} * \left[Y_{21}, Y_{22}, Y_{23}, \cdots, Y_{2k}\right] \\ & \vdots \\ + & X_{1j} * \left[Y_{j1}, Y_{j2}, Y_{j3}, \cdots, Y_{jk}\right] \end{aligned}$$

Formula 1.3 Result unit simplify

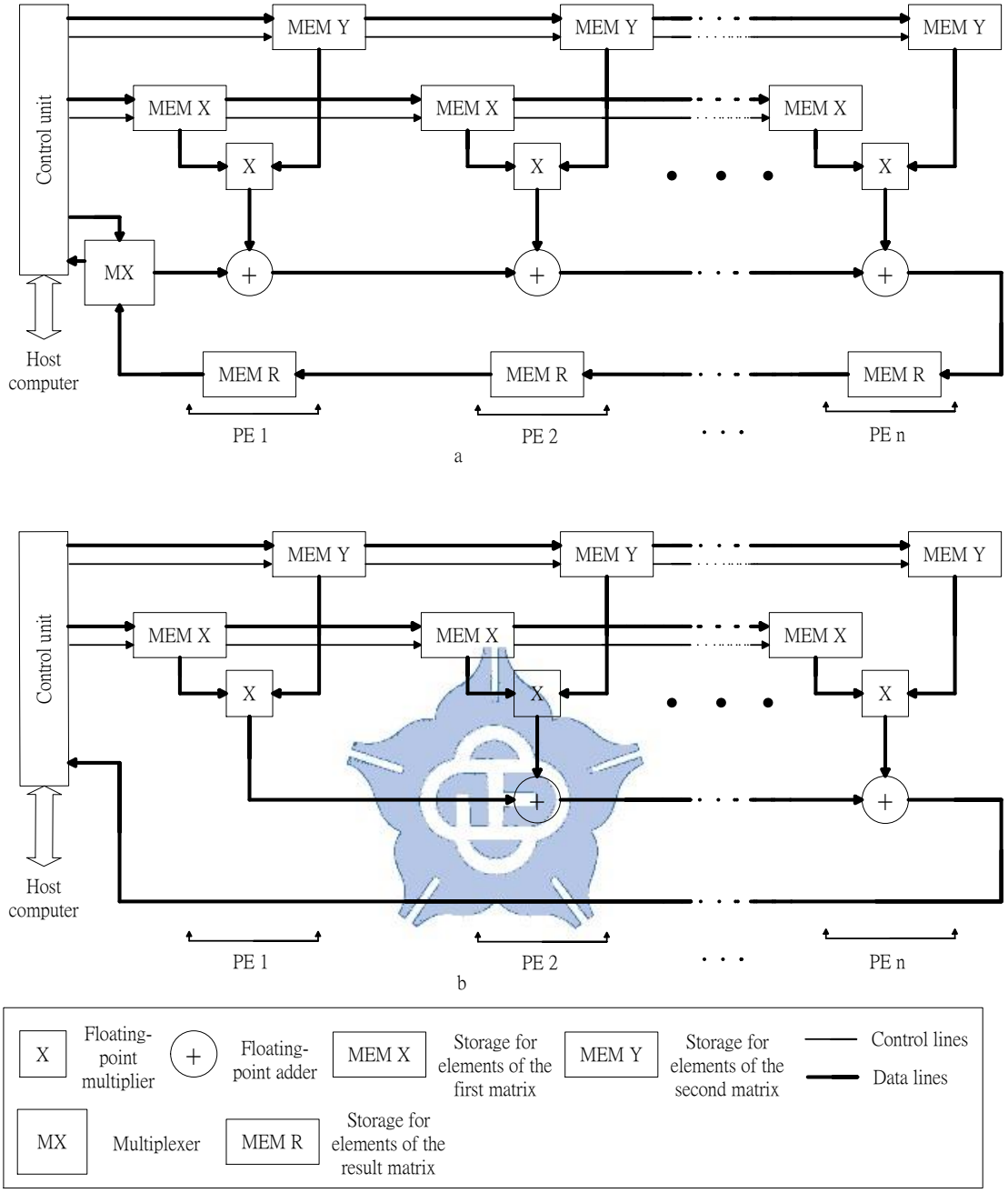
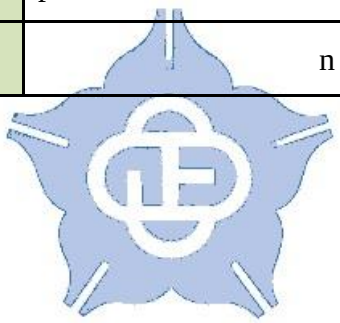


Fig. 1.9 Two architecture “a” and “b” of PE [18]

According to the above description, we make comparison with algorithm 1 and algorithm 2, as following table 1.3

Table 1.3 Comparison with algorithm 1 and algorithm 2

Algorithm 1 with Architecture of a of PE	
Advantage	The algorithm is simple.
Disadvantage	Sending too many repeated block of input matrixes.
No. of pass delay of adder	n stages
Algorithm 2 with Architecture of b of PE	
Advantage	Sending lesser repeated block of input matrixes than algorithm 1.
Disadvantage	It needs the PC to help to temporarily store the some parts of result matrix.
No. of pass delay of adder	n -1 stages



1.4 Motivation

The problems of Big Data has become importance, only improves clock frequency of the computer is not enough, it still takes a lot of time to execution. So one PC link multi general-purpose FPGA-based accelerator over the Ethernet to solve the Big Data analysis problem of long execution time that is our vision. This vision is very broad, so we first study the single-precision floating-point matrix multiplication in one FPGA-based accelerator to solve the problem of long execution time of numerous floating-point matrix multiplications.

In floating-point matrix multiplication, we survey the reference [18]. It has two kinds of algorithms and architectures, the both of algorithms and architectures have some disadvantages. We want to find out an method that to reduce the number of sending input matrix blocks and this method does not need computer that help to temporarily store the some parts of result matrix. We also want to research an architecture of PE that to reduce the gate delay and it must meet our algorithm.

In Chapter 2, we will introduce the architecture of our FPGA-based hardware accelerator and describe the operation flow. In Chapter 3, we will introduce the architecture of our floating-point matrix multiplication circuit and describe the operation flow. In Chapter 4, we will appear the execution time comparison chart which is perform a matrix multiplication and one PC with FPGA accelerator and one PC without FPGA accelerator. Finally, discuss the future works and make a conclusion in Chapter 5.

Chapter 2 Architecture of FPGA-Based Hardware Accelerator

2.1 Architecture Overview

Fig. 2.1 shows the architecture overview of our vision that is one PC link multi FPGA-based accelerator over the Ethernet.

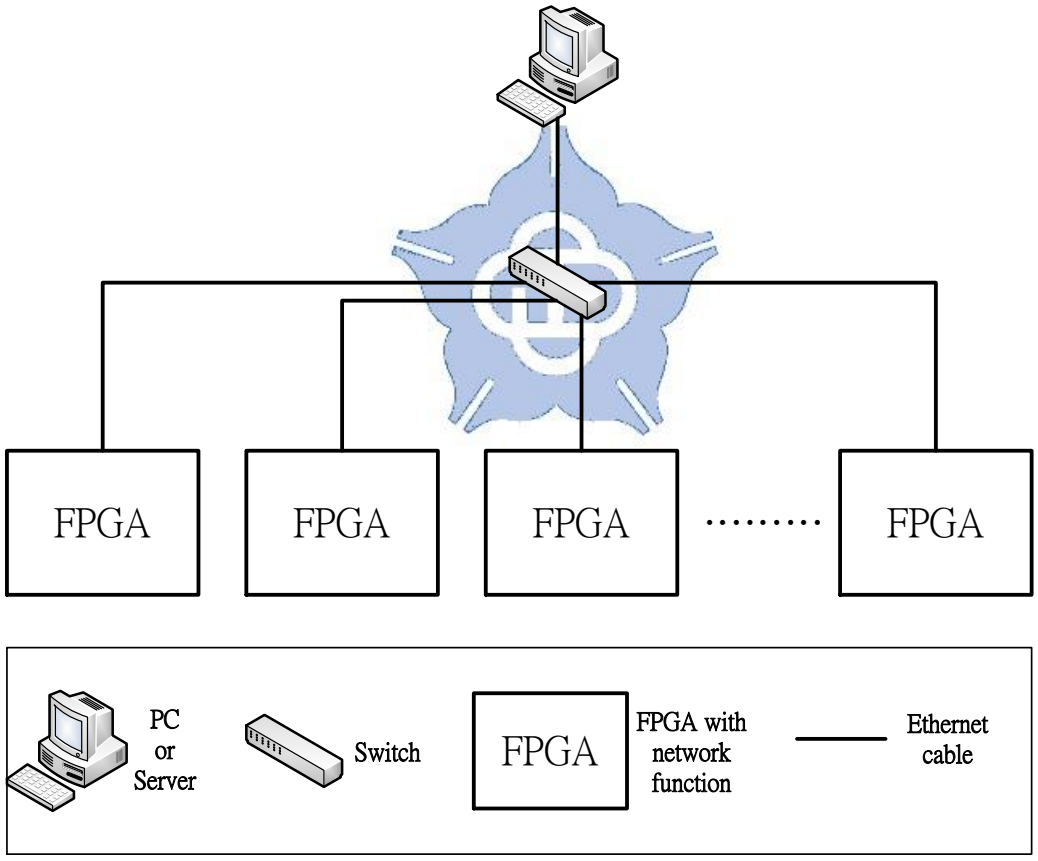


Fig. 2.1 Architecture overview of multi FPGA-based accelerator

In Fig. 2.1 we can see a PC or a server that connects the multi FPGAs which with network function. We use this architecture to speed up the computer which is going to solve the problem of Big Data analysis

2.2 FPGA Development Platform

Our accelerator is implemented using Xilinx Virtex-7 VC707 board (VX485T FPGA). Fig. 2.2 shows the Xilinx Virtex-7 VC707 board. Table 2.2.1 describe the main VC707 board component that we use.

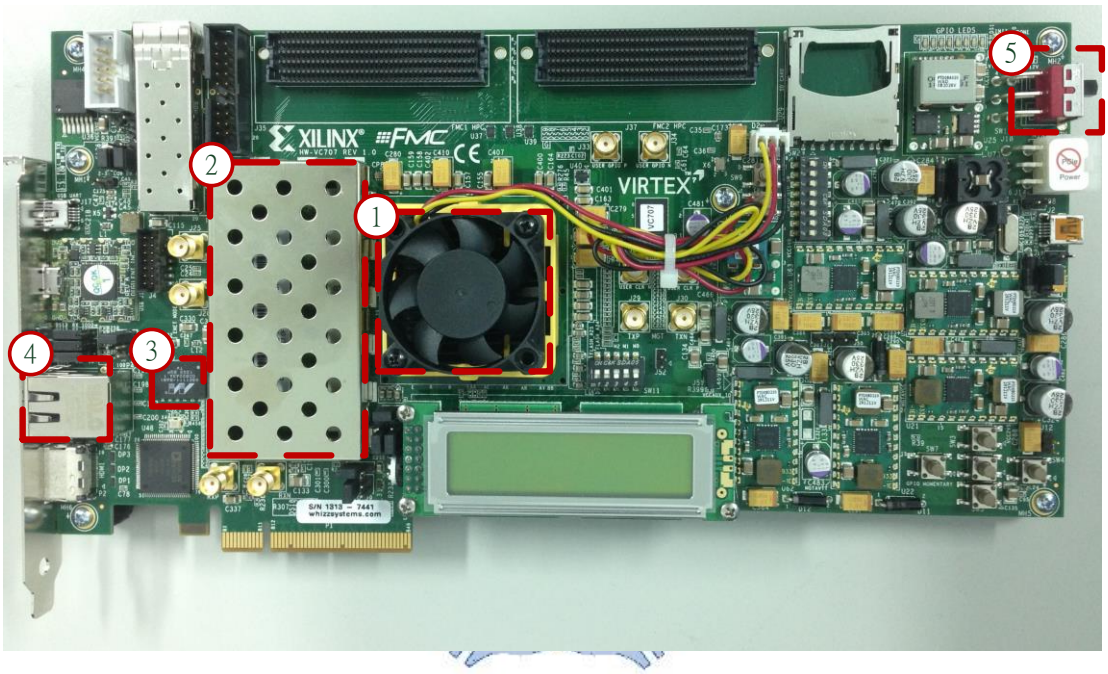


Fig. 2.2 Xilinx Virtex-7 VC707 Board

Table 2.1 VC707 Board Main Component Descriptions

Callout	Component Description
1	Virtex-7 FPGA with cooling fan
2	DDR3 SODIMM memory (1GB)
3	10/100/1000 Mb /s Ethernet PHY
4	Ethernet port
5	Power on/off switch

We mainly use two differential clock sources on VC707 board, namely system clock and GTX transceiver clock. System clock is a 200MHz differential signal pair named SYSCLK_P and SYSCLK_N, it is a reference clock used to provide a high-speed clock for designed top module. GTX transceiver clock is a 125MHz differential signal pair named SGMIICLK_Q0_P and SGMIICLK_Q0_N. It is a reference clock of high-quality and low-jitter for Ethernet IP core (SGMII GTX Transceiver). The description about clock distribution will introduce in next section.

2.3 Architecture of FPGA-Based Hardware Accelerator

Fig. 2.3 shows the architecture of FPGA-based hardware accelerator and clock distribution.

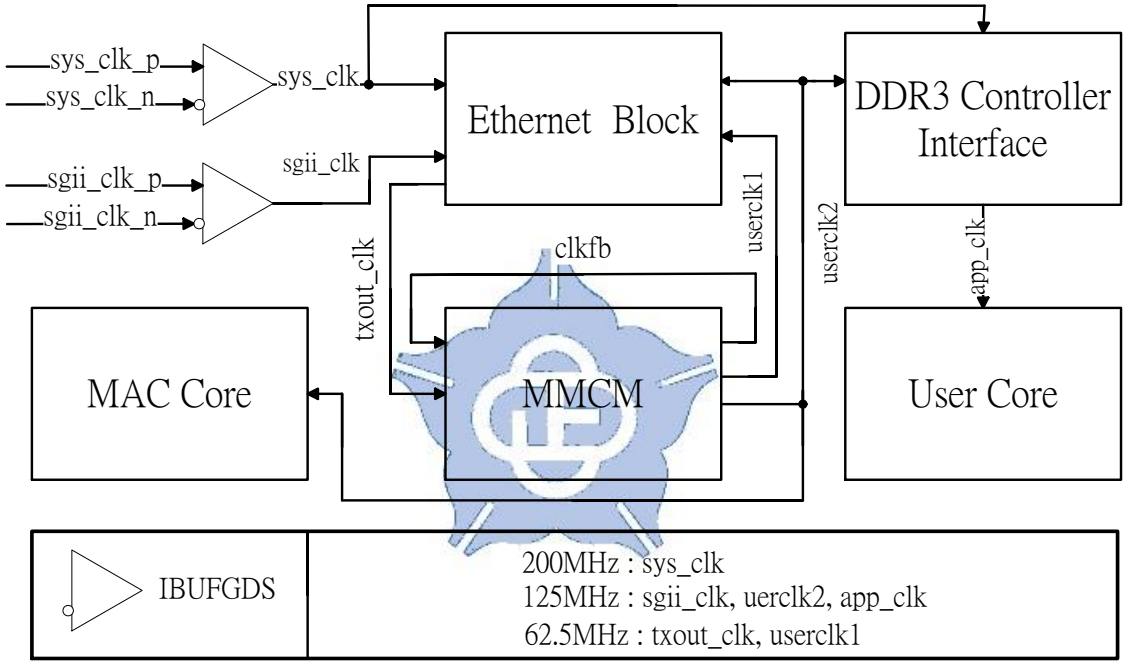


Fig. 2.3 Architecture of FPGA-based hardware accelerator and clock distribution

In Fig. 2.3, we can see the Ethernet block needs two reference clocks, namely sys_clk and sgii_clk, that are 200MHz and 125MHz, respectively. Ethernet block also generates a clock of 62.5MHz named txout_clk that is used as a reference clock for Mixed-Mode Clock Manager (MMCM). MMCM is a Phase-locked loop (PLL) component. After MMCM align phase of txout_clk, it multiplied the clkfb by 10

times. MMCM divided clkfb by 10 times and 5 times, then generate two clock of 62.5MHz and 125MHz respectively, namely userclk1 and a userclk2. Userclk1 is used as a transmission clock for Ethernet block. Userclk2 is used as a reception clock, memory interface clock and Media Access Control (MAC) clock for Ethernet block, DDR3 controller interface and MAC core respectively. DDR3 controller interface generates a clock of 125MHz named app_clk that is used as an operation clock for user_core.

Fig. 2.4 shows the mainly data and control signal flow of FPGA-based hardware accelerator.

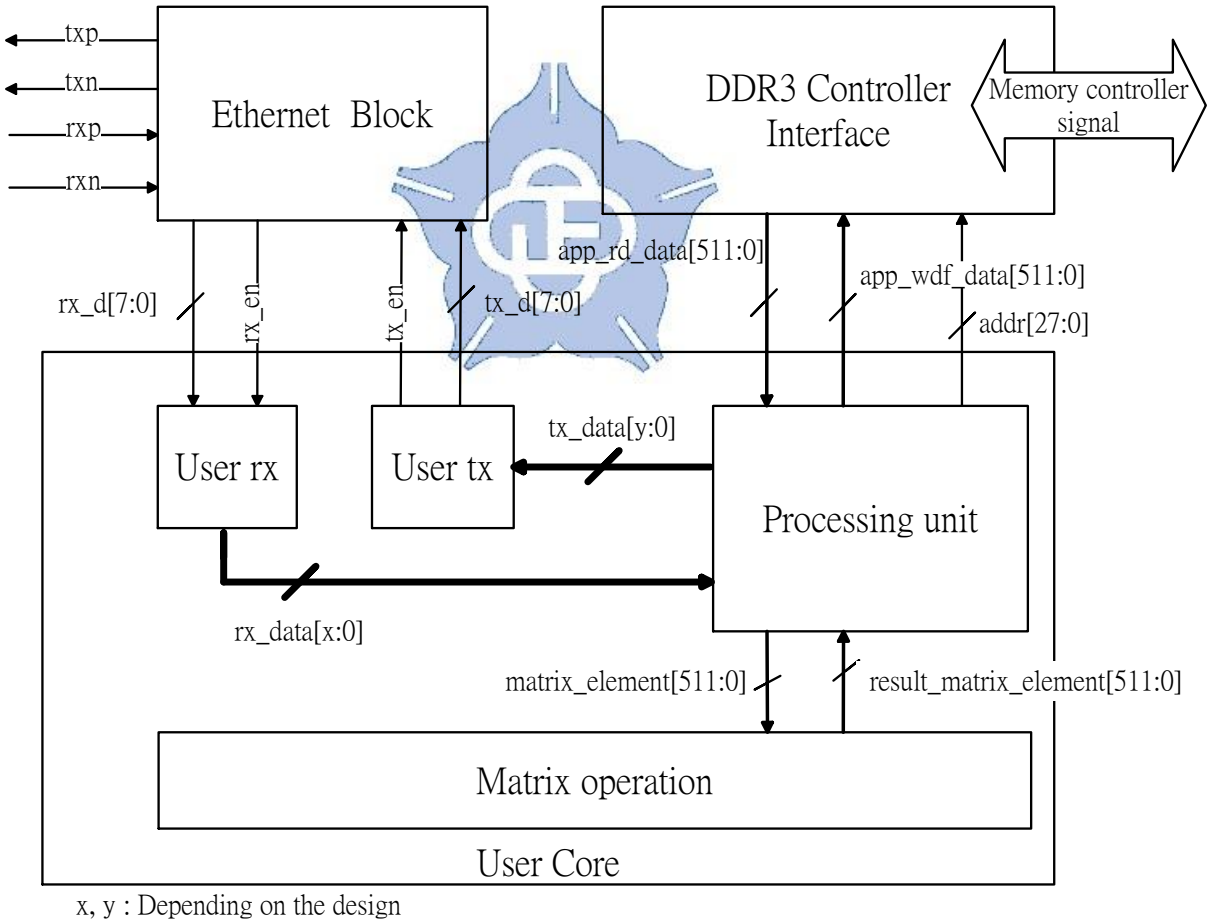


Fig. 2.4 Major data and control signal flow of FPGA-based hardware accelerator

In Fig. 2.4, the Ethernet block is an IP core, it is used to receive a pair of rxp and rxn differential signals and process the two signals into a one byte rx_d bus. The rx_d [7:0] is the received packet. Ethernet block is used to process tx_d [7:0] that is transmission packet into a pair of txp and txn differential signals and sends the two signals into Ethernet PHY.

The DDR3 Controller Interface is an IP core, too. It is used to send app_wdf_data and addr[27:0] to the memory. The app_wdf_data bus is the data that write to the memory. The addr[27:0] is the address for writing or reading the memory. The addr[27:0] is also used to catch an app_rd_data bus from memory controller.

User Core includes the User rx, User tx, Processing unit and matrix operation modules. The User rx module combines rx_d into a received packet, then send the rx_data to processing unit. In section 2.5.1, we will describe the packet processing in User rx module.

The User tx module is used to build up tx_d bus into a transmission packet, then transmit the transmission packet into Ethernet block. In section 2.5.2, we will describe the packet processing of User tx module.

The Processing unit module controls the state machine and manages the data flow of all sub modules in the User Core module. In section 2.4, we will use the control flow to describe the state machine of the Processing unit module.

The Matrix operation module is used to compute matrix multiplication with parallel floating point arithmetic unit that help the host PC to quickly complete matrix multiplication. In section 3.3, we will describe the algorithm and architecture of matrix multiplication.

2.4 User Core Control Flow

Fig. 2.5 shows control flow of User Core. It actually is a state machine in User core, we want to use the simple way of control flow to describe the operation of state machine.

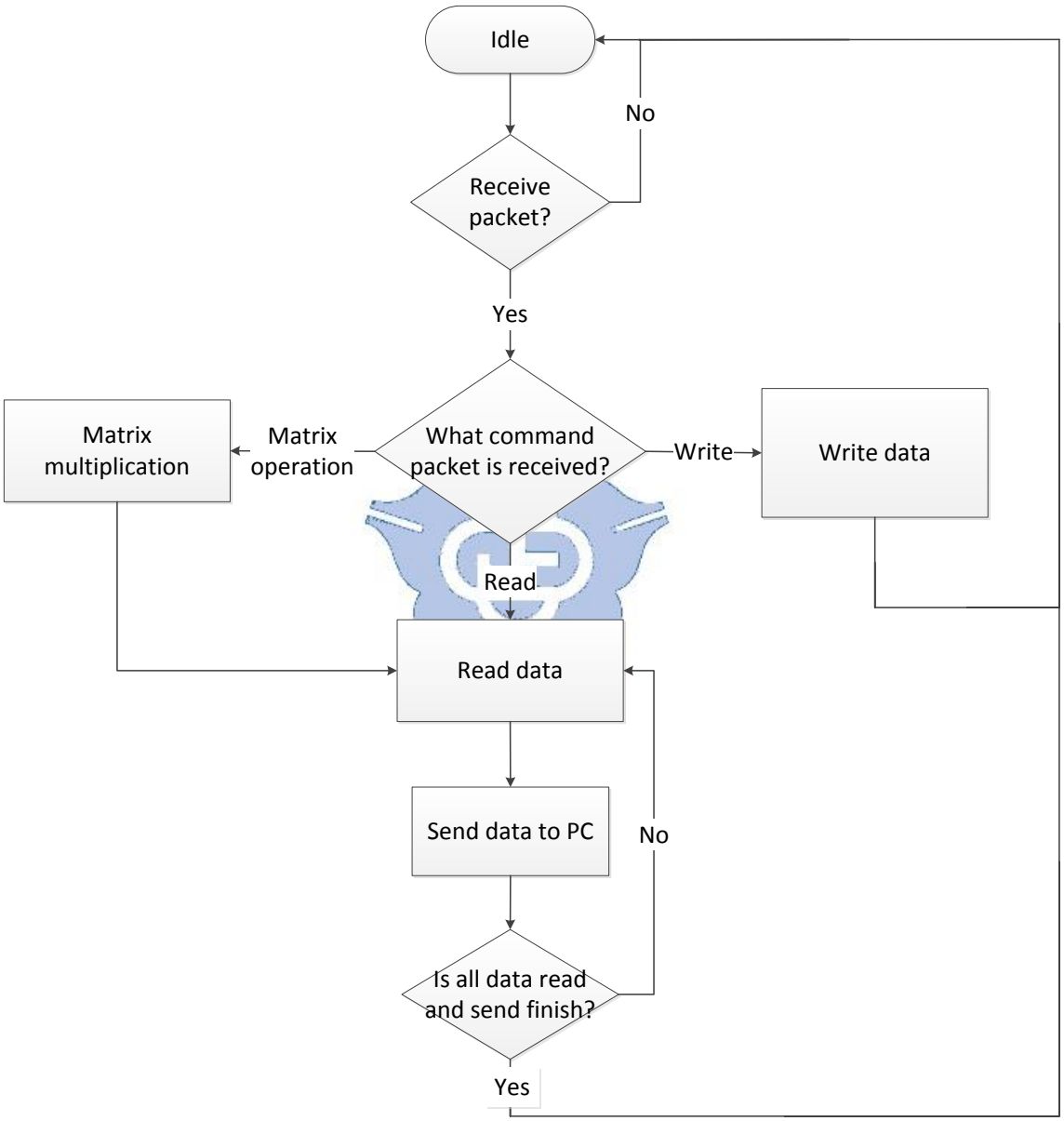


Fig. 2.5 User Core control flow

In Fig. 2.5, we can see the beginning procedure of control flow is idle, then waiting the packets come from PC. After we receive a packet, Processing unit module will check its command field. The command field actually is a data field in packet. In section 3.2,we will introduce the packet format. The command field includes three kind of command, that are write, read and matrix operation. If we receive a write command, control flow will go to procedure of writing data. When the control flow enters the procedure of writing data, the Processing unit module will perform a procedure of writing data. After writing data, procedure of control flow back to idle. In section 2.6.1, we will describe the procedure of writing data. If we receive a read command, control flow will go to procedure of reading data. When the control flow enters the procedure of reading data, the Processing unit module will perform a procedure of reading data, After reading data, procedure of control flow will go to procedure of send data to PC. When the control flow enters the procedure of send data to PC, the Processing unit module will perform a procedure of sending data. After sending the data, Processing unit module will check the condition that if all data has been read and sent. If all data is not been read and sent completely then control flow go back to the procedure of read data. If all data has been read and send the control flow go back to the procedure of idle. In section 2.6.2, we will describe the produce of reading data. If we receive a matrix multiplication command, control flow will go to procedure of matrix multiplication. When the control flow enters the procedure of matrix multiplication, the Processing unit module will compute matrix multiplication. After matrix multiplication, procedure of control flow will go to read data.

2.5 MAC control

In fact, the MAC core is a free IP core, so its function is simple. The function of packet reception and packet transmission are performed in User rx and User tx module respectively.

2.5.1 Packet Reception

Fig. 2.6 shows the timing diagram of packet reception. In Fig. 2.6, if rx_en signals is active-hihg then rx_d is valid.

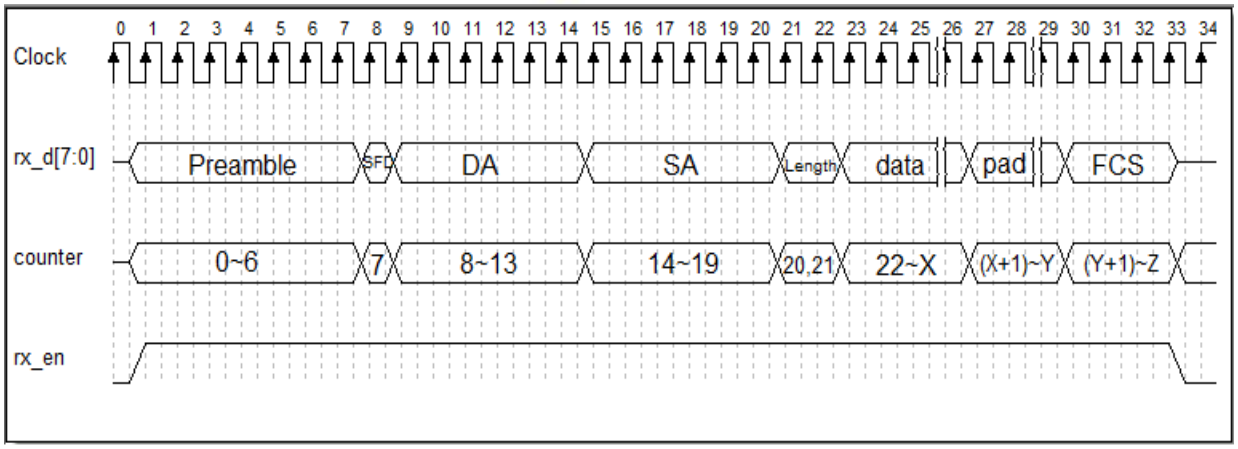


Fig. 2.6 Packet reception timing diagram

When rx_en signals is high, we can use a counter to count the clock cycles and use the value of counter to distinguish what is field of the packet. Table 2.5.1.1 describes the field of packet. After receiving the DA and SA, User rx module will check whether the SA equal to PC MAC address and check whether the DA equal to FPGA MAC address. If the both of DA and SA are equal to the FPGA address and PC address, respectively. Then the data is for us. After Uer rx module receives all data, it will sent

data to Processing unit module.

Table 2.2 Packet field description

Field name	Counter value	Length	Description
Preamble	0~6	7 bytes	Preamble contains a 0x55 pattern, it indicates forefront of a packet. In usually, the preamble is not used.
SFD	7	1 bytes	SFD field contains a 0xD5 pattern, it marks the start of the frame. In usually, the SFD is not used.
DA	8~13	6 bytes	DA contains destination MAC address.
SA	14~19	6 bytes	SA contains source MAC address
Length	20~21	2 bytes	Length of data and pad field
Data	22~X	0-1500 bytes	The field is always provided in the packet data for transmission and is always retained in the receive packet data.
Pad	(X+1)~Y	0-46 bytes	The field is used to ensure that the frame length is at least 64bytes (the preamble and SFD field are not includes for this calculation).
FCS	(Y+1)~Z	4 bytes	The value of the FCS field is calculated from the destination address, source address, length, data and pad fields using a 32-bit cyclic redundancy check (CRC). In our design, Ethernet environment is simple, so we don’t use CRC to check packet.

2.5.2 Packet Transmission

Fig. 2.7 shows the timing diagram of one packet transmission. User tx module is similar to Use rx module, they are different operation direction. After User tx module receives the tx_data from the Processing unit module, it uses a counter to count the clock cycle from 0 to value of packet length. User tx module use the value of counter to distinguish what field is sent to tx_d and pull up tx_en until the counter value equals to the length value.

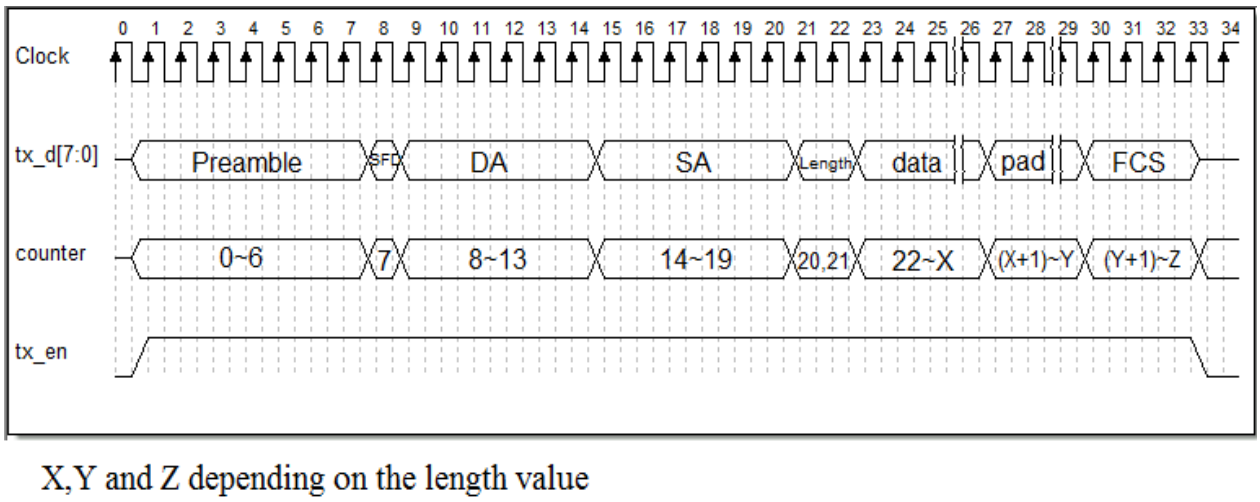


Fig. 2.7 Packet transmission timing diagram

2.6 Memory control

Table 2.3 shows the major signals of memory controller interface of user side.

Table 2.3 major signals of DDR3 memory interface of user side.

Signal	Direction	Description
app_cmd[2:0]	Output	This output selects the command for current request. Read = 001, Write = 000
app_addr[27:0]	Output	This output indicates the address for the current request.
app_en	Output	This is a request signal. The user must apply the desired value to app_addr, app_cmd.
app_rdy	Input	This input indicates the memory interface is ready for accept request of user side.
app_wdf_data[511:0]	Output	The bus provides the data currently begin written to external memory.
app_wdf_end	Output	This output indicates that the data on the app_wdf_data in the current cycle is the last data for the current request.
app_wdf_wren	Output	This input indicates that the data on the app_wdf_data is valid.
app_wdf_rdy	Input	This input indicates that write data FIFO is ready to receive data. Write data accepted when both app_wdf_rdy and app_wdf_wren are asserted.
app_rd_data[511:0]	Input	This output contains the data reading from the external memory.
app_rd_data_valid	Input	This input indicates that the data on the app_rd_data is valid.

2.6.1 Memory Write

Fig. 2.8 shows timing diagram of memory write. In Fig. 2.8, if we want to write a data, we must follow the rules of writeing data. Writing data into memory can be divided into two parts that are writing data and writing commands. First, we assign data to the app_wdf_data of 512 bits and assert the app_wdf_wren and app_wdf_end signals until app_wdf_rdy signal is high. Second, we assign a write command and an address to app_cmd and app_addr, respectively. Then we assert the app_en signal until app_rdy signal is high.

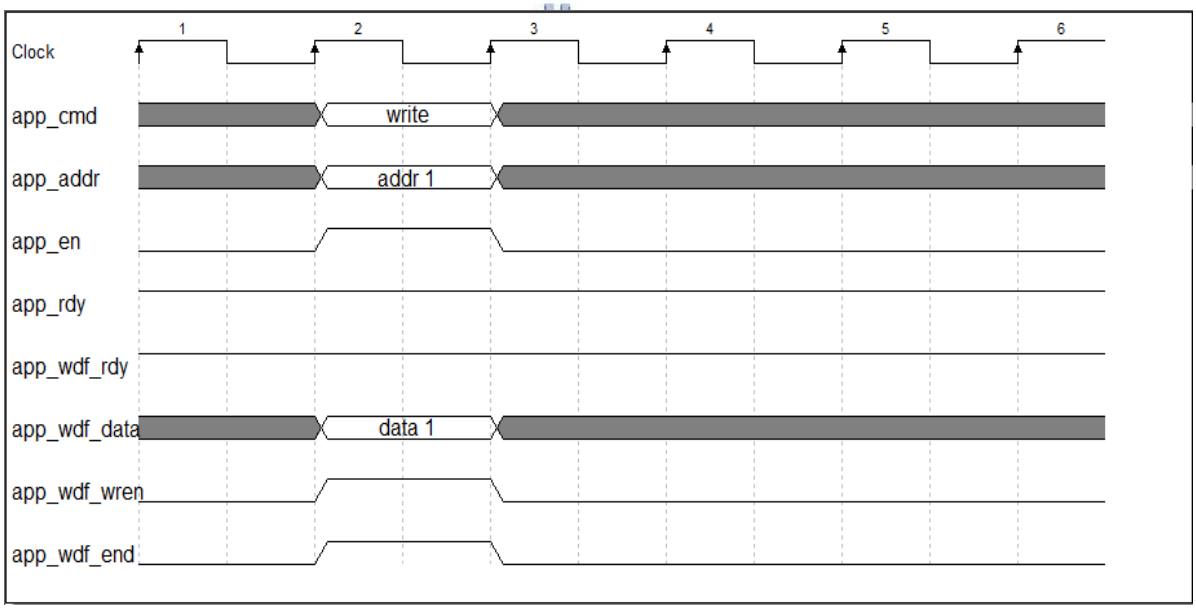


Fig 2.8 Memory write timing diagram

If we want to use memory bandwidth efficiently, we can write data in burst mode. Fig. 2.9 shows timing diagram of burst mode operation with depth 8 times data write. Writing data into memory can be divided into two parts that are writibg data and writing

command, too. First, if app_wdf_rdy signal is always high, and then we continuously assign eight data to the app_wdf_data of 512 bits and continuously assert the app_wdf_wren signal. Then we assert the app_wdf_end in last clock cycle of writing data. In writing data procedure, if app_wdf_rdy signal gives low, we must stop writing data and assert the app_wdf_wren signal until app_wdf_rdy signal is high. Second, if app_rdy signal is high and, and then we continuously assign write commands and 8 address to the app_cmd and app_addr, respectively and continuously assert the app_en signal. In writing command procedure, if app_rdy signal gives low then we must stop writing address and assert app_en signal unitl app_rdy signal is high.

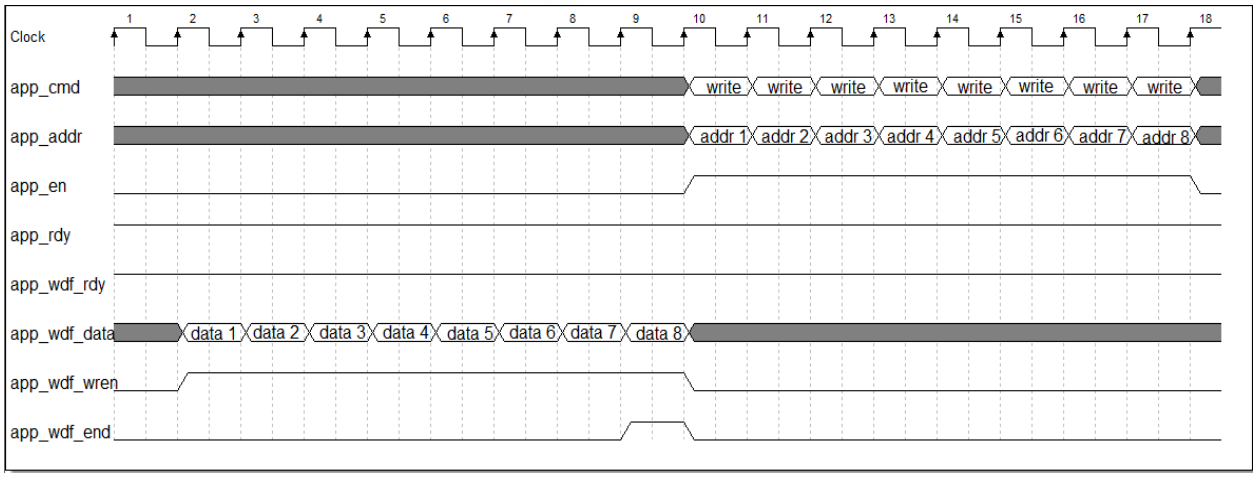


Fig. 2.9 Memory burst mode timing diagram of 8 times data write

2.6.2 Memory Read

Fig. 2.10 shows timing diagram of memory read. In Fig. 2.10, if we want to read a datum, we must follow the rules of reading data. Reading data from memory can be divided into two parts that is read command and read data. First, we assign a command and an address the app_cmd and app_addr, respectively and assert the app_en signal until app_rdy signal is high. Second, if app_rd_data_valid is asserted then store the data from app_rd_data.

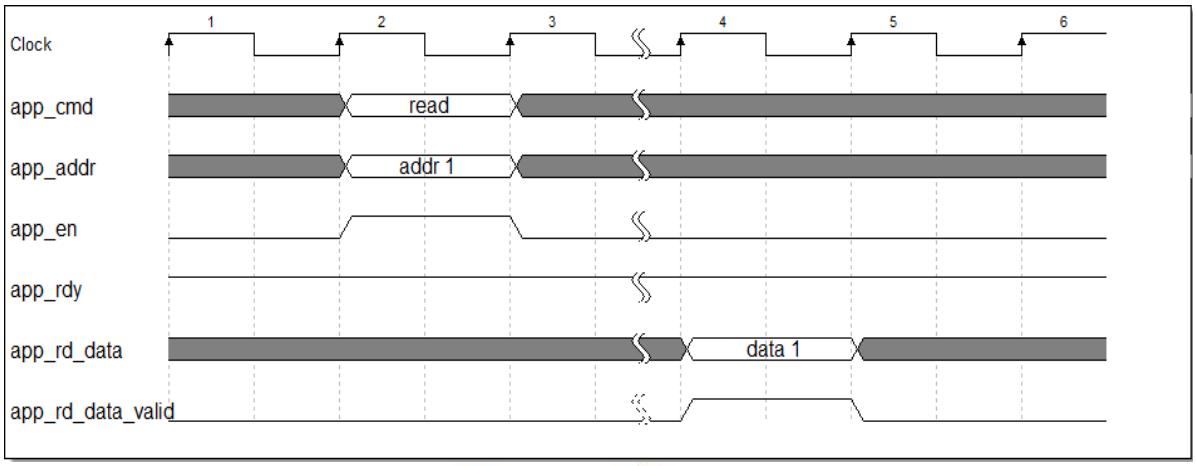


Fig. 2.10 memory read timing diagram.

If we want to read data efficiently, we can read data in burst mode. Fig. 2.11 shows timing diagram of memory burst read mode with depth 8. Reading data from memory can be divided into two parts that are read command and read data. First, if app_rdy signal is high, we can continuously assign read command and 8 addresses to app_cmd and app_addr, respectively. Then we continuously assert the app_en signal, In read command procedure, if app_rdy signal is low, we must stop the read operation and assert the app_en signal until app_rdy signal is high. Second, if app_rd_data_valid is

asserted then store the data from app_rd_data.

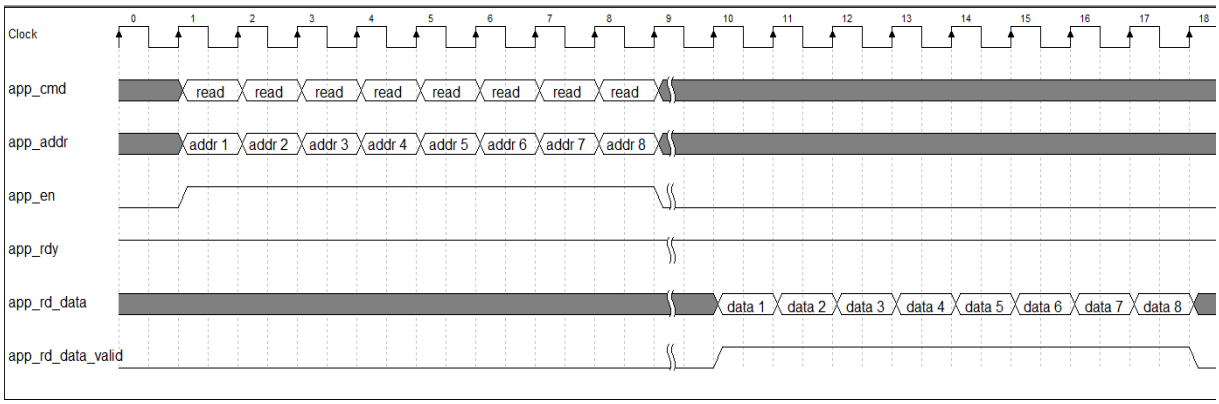


Fig. 2.11 Memory burst mode timing diagram of 8 times data read

Chapter 3 Floating-point Matrix Multiplication on FPGA-Based Hardware Accelerator

3.1 Matrix storage sequence in FPGA board DDR3 Memory

In Formula 3.1 we can see that two 8 by 8 matrix A and matrix B multiplication. That is matrix A * matrix B = matrix C. We first observe the $[c_{1\,1}, \ldots c_{1\,4}]$ in Formula 3.2 In Formula 3.2 we can find a rule about matrix multiplication. If we divide matrix A into several 1 by 4 matrix blocks X and divide matrix B into several 4 by 1 matrix Y like formula 3.5. Then formula 3.2 can be rewritten as formula 3.3. We can simplify Formula 3.3 into formula 3.4. In formula 3.4, we can see a group of regular sequence like $X_{1\,1}, Y_{1\,1}, Y_{1\,2}, Y_{1\,3}, Y_{1\,4}$ and $X_{1\,2}, Y_{2\,1}, Y_{2\,2}, Y_{2\,3}, Y_{2\,4}$, If we use this sequence to store the data and computation, then we can simply multiply two matrixes of various size by hardware. Before we use this sequence to operation, we must use order of formula 3.5 to write matrix A and matrix B element into memory.

$$\begin{bmatrix} a_{1\,1}, a_{1\,2}, a_{1\,3}, a_{1\,4}, a_{1\,5}, a_{1\,6}, a_{1\,7}, a_{1\,8} \\ a_{2\,1}, a_{2\,2}, a_{2\,3}, a_{2\,4}, a_{2\,5}, a_{2\,6}, a_{2\,7}, a_{2\,8} \\ a_{3\,1}, a_{3\,2}, a_{3\,3}, a_{3\,4}, a_{3\,5}, a_{3\,6}, a_{3\,7}, a_{3\,8} \\ a_{4\,1}, a_{4\,2}, a_{4\,3}, a_{4\,4}, a_{4\,5}, a_{4\,6}, a_{4\,7}, a_{4\,8} \\ a_{5\,1}, a_{5\,2}, a_{5\,3}, a_{5\,4}, a_{5\,5}, a_{5\,6}, a_{5\,7}, a_{5\,8} \\ a_{6\,1}, a_{6\,2}, a_{6\,3}, a_{6\,4}, a_{6\,5}, a_{6\,6}, a_{6\,7}, a_{6\,8} \\ a_{7\,1}, a_{7\,2}, a_{7\,3}, a_{7\,4}, a_{7\,5}, a_{7\,6}, a_{7\,7}, a_{7\,8} \\ a_{8\,1}, a_{8\,2}, a_{8\,3}, a_{8\,4}, a_{8\,5}, a_{8\,6}, a_{8\,7}, a_{8\,8} \end{bmatrix} * \begin{bmatrix} b_{1\,1}, b_{1\,2}, b_{1\,3}, b_{1\,4}, b_{1\,5}, b_{1\,6}, b_{1\,7}, b_{1\,8} \\ b_{2\,1}, b_{2\,2}, b_{2\,3}, b_{2\,4}, b_{2\,5}, b_{2\,6}, b_{2\,7}, b_{2\,8} \\ b_{3\,1}, b_{3\,2}, b_{3\,3}, b_{3\,4}, b_{3\,5}, b_{3\,6}, b_{3\,7}, b_{3\,8} \\ b_{4\,1}, b_{4\,2}, b_{4\,3}, b_{4\,4}, b_{4\,5}, b_{4\,6}, b_{4\,7}, b_{4\,8} \\ b_{5\,1}, b_{5\,2}, b_{5\,3}, b_{5\,4}, b_{5\,5}, b_{5\,6}, b_{5\,7}, b_{5\,8} \\ b_{6\,1}, b_{6\,2}, b_{6\,3}, b_{6\,4}, b_{6\,5}, b_{6\,6}, b_{6\,7}, b_{6\,8} \\ b_{7\,1}, b_{7\,2}, b_{7\,3}, b_{7\,4}, b_{7\,5}, b_{7\,6}, b_{7\,7}, b_{7\,8} \\ b_{8\,1}, b_{8\,2}, b_{8\,3}, b_{8\,4}, b_{8\,5}, b_{8\,6}, b_{8\,7}, b_{8\,8} \end{bmatrix} = \begin{bmatrix} c_{1\,1}, c_{1\,2}, c_{1\,3}, c_{1\,4}, c_{1\,5}, c_{1\,6}, c_{1\,7}, c_{1\,8} \\ c_{2\,1}, c_{2\,2}, c_{2\,3}, c_{2\,4}, c_{2\,5}, c_{2\,6}, c_{2\,7}, c_{2\,8} \\ c_{3\,1}, c_{3\,2}, c_{3\,3}, c_{3\,4}, c_{3\,5}, c_{3\,6}, c_{3\,7}, c_{3\,8} \\ c_{4\,1}, c_{4\,2}, c_{4\,3}, c_{4\,4}, c_{4\,5}, c_{4\,6}, c_{4\,7}, c_{4\,8} \\ c_{5\,1}, c_{5\,2}, c_{5\,3}, c_{5\,4}, c_{5\,5}, c_{5\,6}, c_{5\,7}, c_{5\,8} \\ c_{6\,1}, c_{6\,2}, c_{6\,3}, c_{6\,4}, c_{6\,5}, c_{6\,6}, c_{6\,7}, c_{6\,8} \\ c_{7\,1}, c_{7\,2}, c_{7\,3}, c_{7\,4}, c_{7\,5}, c_{7\,6}, c_{7\,7}, c_{7\,8} \\ c_{8\,1}, c_{8\,2}, c_{8\,3}, c_{8\,4}, c_{8\,5}, c_{8\,6}, c_{8\,7}, c_{8\,8} \end{bmatrix}$$

Formula 3.1 two 8 x 8 matrix multiplication

$$\begin{aligned} &\left[c_{1\,1}, c_{1\,2}, c_{1\,3}, c_{1\,4}\right] = \\ &\left[\left[a_{1\,1}, a_{1\,2}, a_{1\,3}, a_{1\,4}\right] * \begin{bmatrix} b_{1\,1} \\ b_{2\,1} \\ b_{3\,1} \\ b_{4\,1} \end{bmatrix}, \left[a_{1\,1}, a_{1\,2}, a_{1\,3}, a_{1\,4}\right] * \begin{bmatrix} b_{1\,2} \\ b_{2\,2} \\ b_{3\,2} \\ b_{4\,2} \end{bmatrix}, \left[a_{1\,1}, a_{1\,2}, a_{1\,3}, a_{1\,4}\right] * \begin{bmatrix} b_{1\,3} \\ b_{2\,3} \\ b_{3\,3} \\ b_{4\,3} \end{bmatrix}, \left[a_{1\,1}, a_{1\,2}, a_{1\,3}, a_{1\,4}\right] * \begin{bmatrix} b_{1\,4} \\ b_{2\,4} \\ b_{3\,4} \\ b_{4\,4} \end{bmatrix} \right] + \\ &\left[\left[a_{1\,5}, a_{1\,6}, a_{1\,7}, a_{1\,8}\right] * \begin{bmatrix} b_{5\,1} \\ b_{6\,1} \\ b_{7\,1} \\ b_{8\,1} \end{bmatrix}, \left[a_{1\,5}, a_{1\,6}, a_{1\,7}, a_{1\,8}\right] * \begin{bmatrix} b_{5\,2} \\ b_{6\,2} \\ b_{7\,2} \\ b_{8\,2} \end{bmatrix}, \left[a_{1\,5}, a_{1\,6}, a_{1\,7}, a_{1\,8}\right] * \begin{bmatrix} b_{5\,3} \\ b_{6\,3} \\ b_{7\,3} \\ b_{8\,3} \end{bmatrix}, \left[a_{1\,5}, a_{1\,6}, a_{1\,7}, a_{1\,8}\right] * \begin{bmatrix} b_{5\,4} \\ b_{6\,4} \\ b_{7\,4} \\ b_{8\,4} \end{bmatrix} \right] \end{aligned}$$

Formula 3.2 c1 1~c1 4 of matrix C

$$\left[c_{1\,1}, c_{1\,2}, c_{1\,3}, c_{1\,4}\right] = \left[\left[X_{1\,1}\right] * \left[Y_{1\,1}\right], \left[X_{1\,1}\right] * \left[Y_{1\,2}\right], \left[X_{1\,1}\right] * \left[Y_{1\,3}\right], \left[X_{1\,1}\right] * \left[Y_{1\,4}\right]\right] +$$

$$\left[\left[X_{1\,2}\right] * \left[Y_{2\,1}\right], \left[X_{1\,2}\right] * \left[Y_{2\,2}\right], \left[X_{1\,2}\right] * \left[Y_{2\,3}\right], \left[X_{1\,2}\right] * \left[Y_{2\,4}\right]\right]$$

Formula 3.3 use matrix block X and Y to indicate c1 1~c1 4 of matrix C

$$\left[c_{1\,1}, c_{1\,2}, c_{1\,3}, c_{1\,4}\right] = \left[X_{1\ 1}\right] * \left[\left[Y_{1\ 1}\right], \left[Y_{1\ 2}\right], \left[Y_{1\ 3}\right], \left[Y_{1\ 4}\right]\right] + \left[X_{1\ 2}\right] * \left[\left[Y_{2\ 1}\right], \left[Y_{2\ 2}\right], \left[Y_{2\ 3}\right], \left[Y_{2\ 4}\right]\right]$$

Formula 3.4 simplify the formula 3.3

$$matrixA = \begin{bmatrix} X_{1\ 1}, X_{1\ 2} \\ X_{2\ 1}, X_{2\ 2} \\ X_{3\ 1}, X_{3\ 2} \\ X_{4\ 1}, X_{4\ 2} \\ X_{5\ 1}, X_{5\ 2} \\ X_{6\ 1}, X_{6\ 2} \\ X_{7\ 1}, X_{7\ 2} \\ X_{8\ 1}, X_{8\ 2} \end{bmatrix}$$

$$matrixB = \begin{bmatrix} Y_{1\ 1}, Y_{1\ 2}, Y_{1\ 3}, Y_{1\ 4}, Y_{1\ 5}, Y_{1\ 6}, Y_{1\ 7}, Y_{1\ 8} \\ Y_{2\ 1}, Y_{2\ 2}, Y_{2\ 3}, Y_{2\ 4}, Y_{2\ 5}, Y_{2\ 6}, Y_{2\ 7}, Y_{2\ 8} \end{bmatrix}$$

$$matrixC = \begin{bmatrix} R_{1\ 1}, R_{1\ 2} \\ R_{2\ 1}, R_{2\ 2} \\ R_{3\ 1}, R_{3\ 2} \\ R_{4\ 1}, R_{4\ 2} \\ R_{5\ 1}, R_{5\ 2} \\ R_{6\ 1}, R_{6\ 2} \\ R_{7\ 1}, R_{7\ 2} \\ R_{8\ 1}, R_{8\ 2} \end{bmatrix}$$

Formula 3.5 order of matrix A , matrix B and matrix C in memory

3.2 Packet Format

In order to communicate between the PC and FPGA, we must define the data fields of a packet. Fig 3.1 shows the data field of one packet that sent from PC to FPGA. Fig 3.2 shows the data fields of one packet that sent from FPGA to PC. Table 3.1 describe the data field of packet.

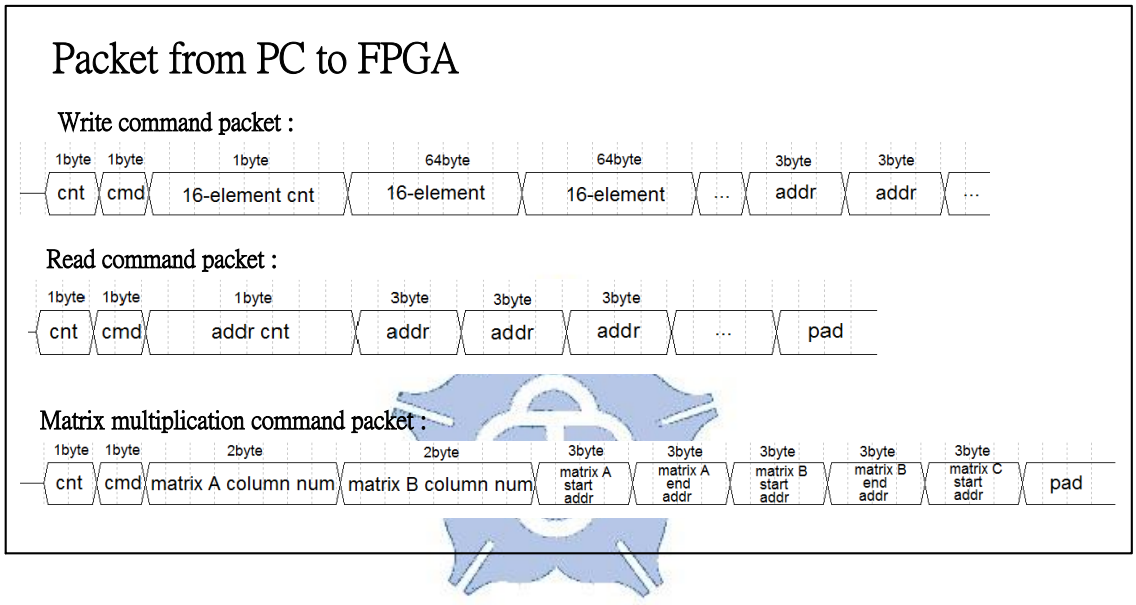


Fig. 3.1 data field of packet from PC to FPGA

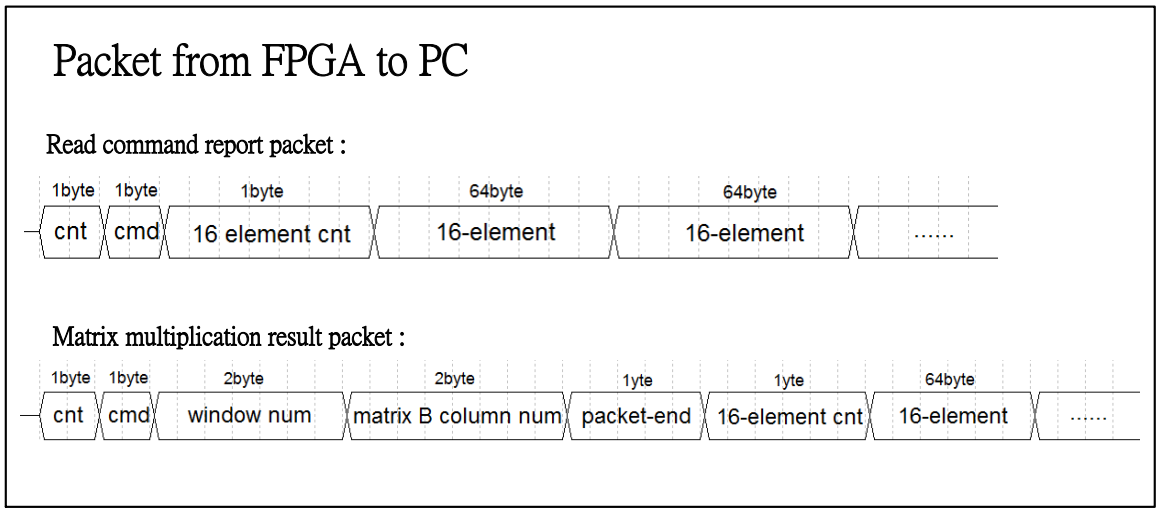


Fig. 3.2 data field of packet from FPGA to PC

Table 3.1 data field description

Field name	Description
cnt	This field indicates the sequence number of packet.
cmd	This field indicates the operation which we want to execute. Write command packet : 0x01 Read command packet : 0x02 Read command report packet : 0x02 Matrix Multiplication command packet : 0x03 Matrix Multiplication result packet : 0x03
16-element cnt	This field indicates the number of 16-element in this packet.
16-element	This field indicates a group of values of 16 4-bytes.
addr	This field indicates the address.
addr cnt	This field indicates the number of address in this packet.
matrix A column num	This field indicates the value of matrix A column.
matrix B column num	This field indicates the value of matrix B column.
matrix A start addr	This field indicates the starting address of matrix A.
matrix A end addr	This field indicates the end address of matrix A.
matrix B start addr	This field indicates the starting address of matrix B.
matrix B end addr	This field indicates the end address of matrix B.
matrix C start addr	This field indicates the starting address of matrix C.

3.3 Architecture of Floating-point Matrix Multiplication Circuit

3.3.1 Architecture overview

Fig. 3.3 shows the architecture overview of floating-point matrix multiplication circuit. In Fig. 3.3, there are one matrix processor master and 4 matrix processors. The matrix processor master reads numerous matrix elements from the DDR memory and sends the elements to various matrix processors for computation. After matrix processors finish computation, matrix processor master sample those results and writes results into DDR memory. Fig. 3.3 shows schematic diagram, there are 16 matrix processors in the floating-point matrix multiplication circuit.

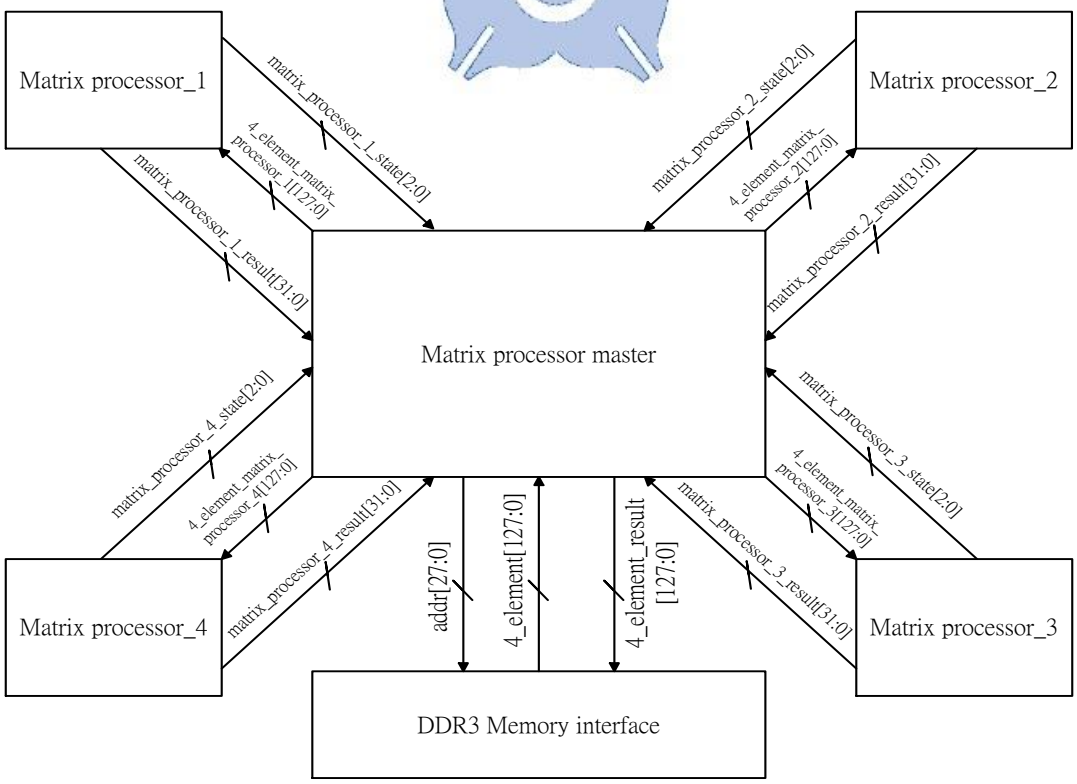


Fig. 3.3 Floating-point matrix multiplication circuit overview

3.3.2 Matrix Processor

Fig. 3.4 shows the architecture of matrix processor.

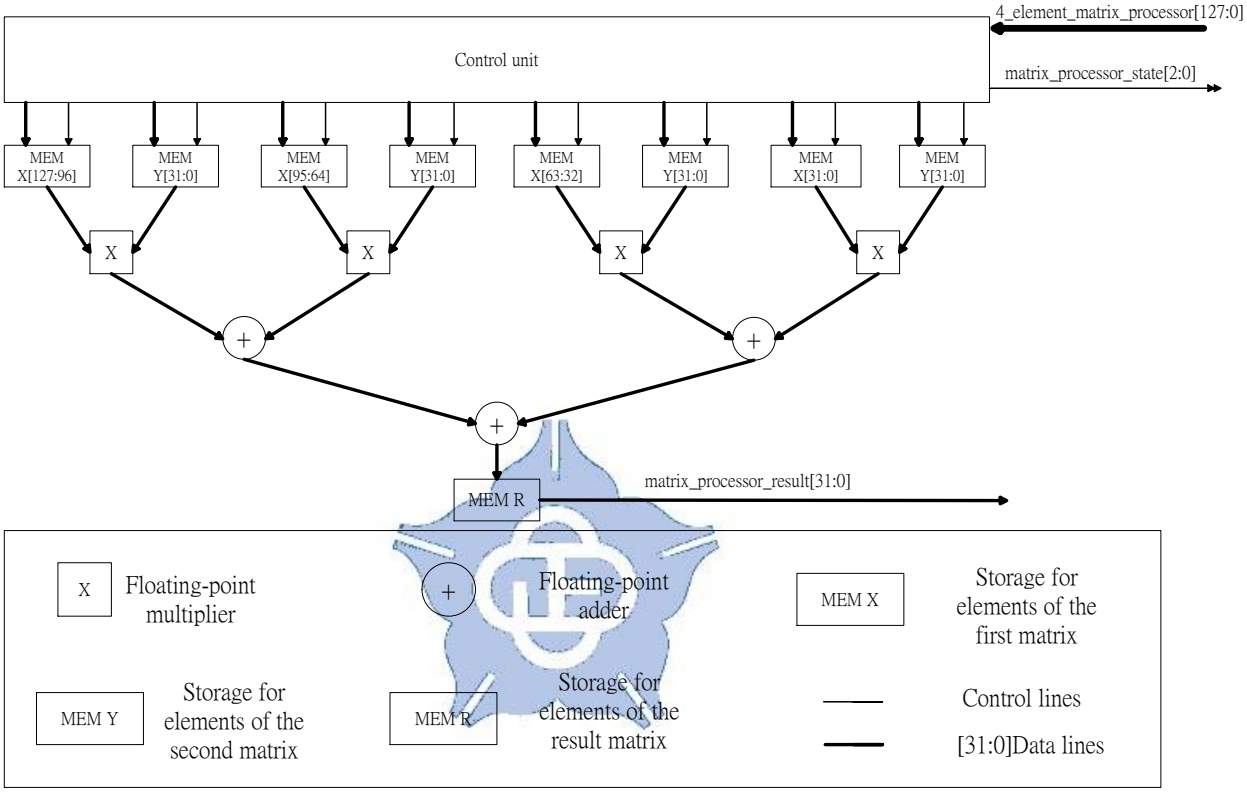


Fig. 3.4 architecture of matrix processor

In Fig. 3.4, we can see an architecture of the matrix processor of full binary tree (4 level) structure. There are 4 PEs in the architecture. First, control unit catch the input data of matrix block 127bits bus, the matrix block X is send to MEM X register and the matrix Y is send to MEM Y register. After the matrix block X and matrix block Y are send, computation starts, then waiting the multiplication results. Figure 3.4 shows the schematic diagram, architecture of matrix processor is a full binary tree (6 level) structure.

In the architecture of matrix processor, the number of pass delay of adder is $\log_2 n$ stages, it is less than n in reference [18], there are 16 matrix processor in matrix processor, so number of pass delay is 8 stages.

3.3.3 Matrix processor master

Fig. 3.5 shows architecture matrix processor master.

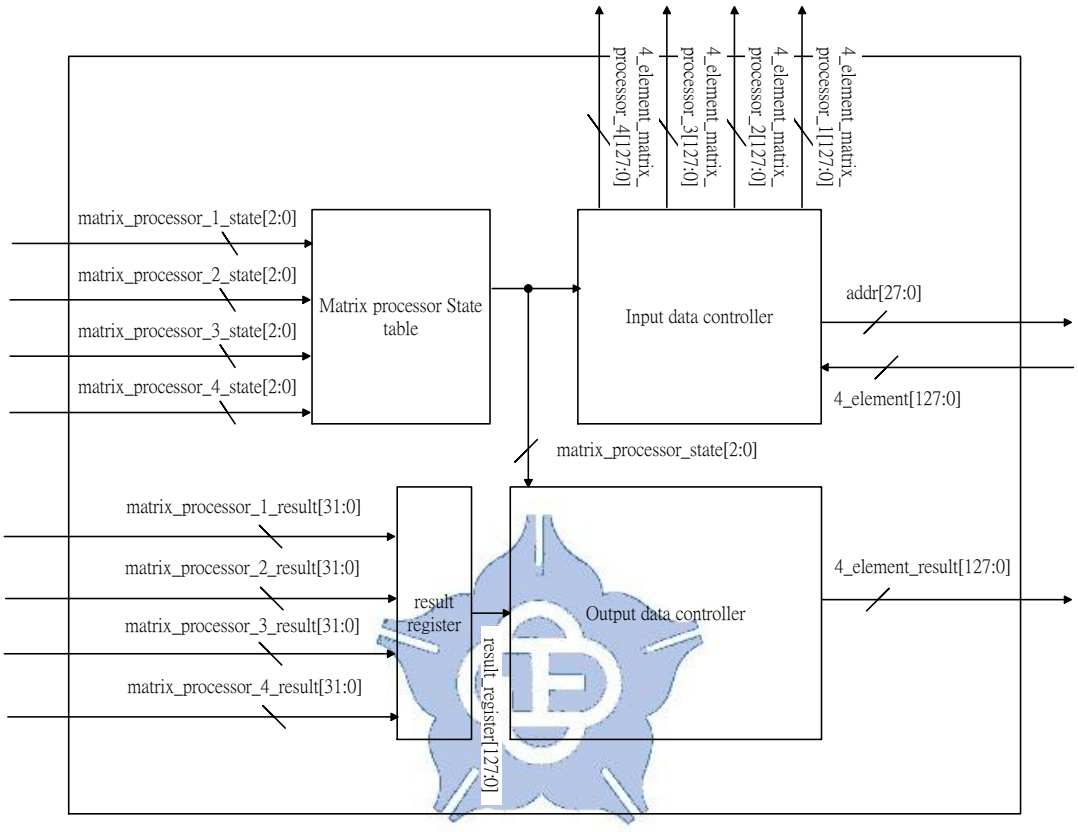


Fig. 3.5 matrix processor master

In Fig. 3.5, we can see the mainly three blocks, Input data controller, Matrix processor state table and Output data controller. First, Input data controller uses information of matrix A and matrix B to read matrix X blocks and matrix Y blocks from memory. After Input data controller receives the matrix blocks, it checks if the matrix processor state is idle. If matrix processor is idle then the Input data controller assigns the matrix blocks to matrix processor. Second, if matrix processor state is calculation finish, then Output data controller will store the result from matrix processor. Third, if result have not been calculated yet then it will be temporarily stored in Result register. Final, if result is complete then result will be written into the memory. Figure 3.5 shows

the schematic diagram, there are 16 element_matrix_process bus, 16 matrix_porcessor_state bus and 16 matrix_processor_result bus.

Chapter 4 Experimental Results

4.1 Ethernet Transfer result

4.1.1 Ethernet Transfer rate

In this section, we transmit the 50 thousand 1500-byte in FPGA-based accelerator and PC to test the transmission rate by using ethernet. Table 4.1 shows the transmission rate in this measurement.

Table 4.1 FPGA-based accelerator and PC transmission rate

Direction	Packet length	Number of Packet	Time	Transmission data rate
PC to FPGA	1500bytes	50,000	11.025s	52.52Mbps
FPGA to PC	1500bytes	50,000	8.698s	66.56Mbps

We also test the transmission data rate with the DDR memory delay includes data from PC to FPGA‘s external memory and data from FPGA external’s memory to PC. Table 4.2 shows the transmission rate in this condition.

Table 4.2 transmission rate with the DDR memory delay

Direction	Packet length	Number of Packet	Time	Transmission data rate
PC to FPGA	1500bytes	50,000	11.52s	52.08Mbps
FPGA to PC	1500bytes	50,000	9.262s	64.78Mbps

The Ethernet transmission data rate will be the bottleneck to the FPGA-based accelerator, for large data computatiion. In the future, if company of Xillinx has provided high transmission rate of Ethernet IP core then it will greatly enhance the

efficiency of our architecture of FPGA-based accelerator.

4.1.2 Ethernet Tx FIFO Issue

Fig. 4.1 shows the transmission data flow. The packet from User Tx module to Ethernet block is 125MHz and packet from Ethernet block run at PHY run at 62.5MHz. If we transmit too many packets, in a short time then it causes transmission FIFO overflow. Due to MAC core is a free IP, it can't to solve the problem of transmission FIFO overflow. We use an easy method to avoid this problem. We obtain the number of waiting clock cycles by experimental result. The number of waiting clock cycles between each packet transmission is equal to $6^{*}(\text{Total packet}/10)$.

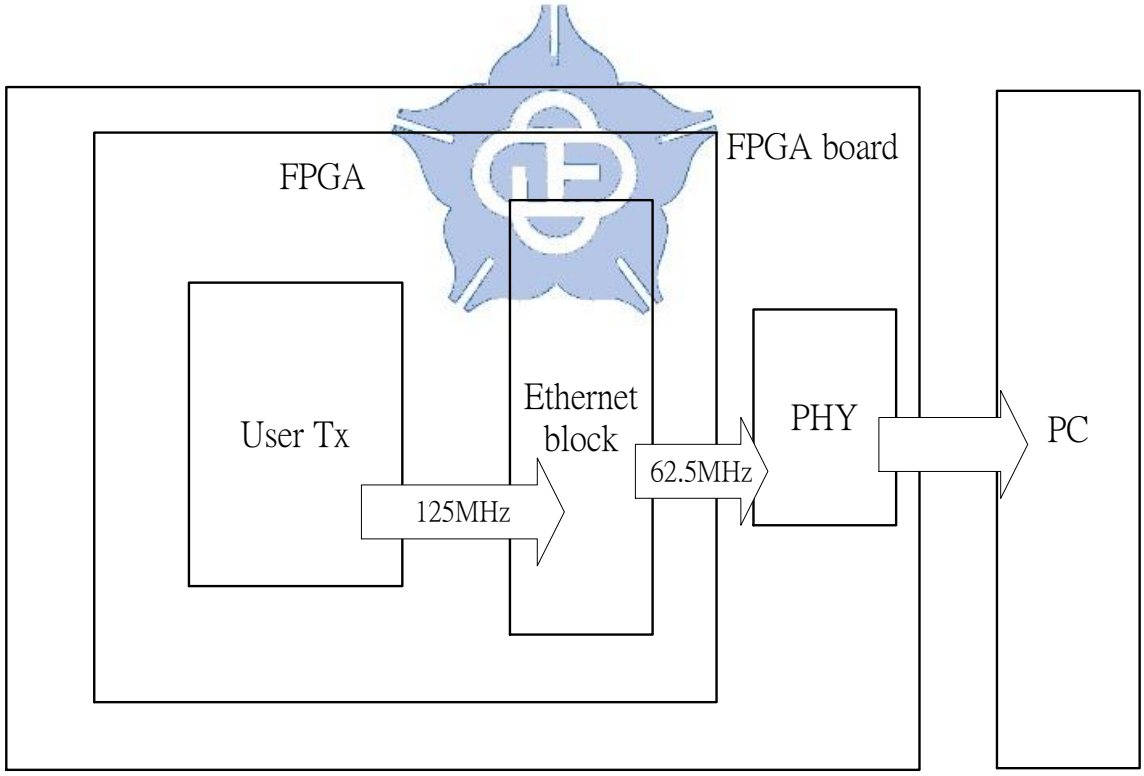


Fig. 4.1 transmission data flow

4.2 Matrix Multiplication Experimental Result

In Fig. 4.2 ~ Fig. 4.10, we use three different computers to compute different matrix sizes matrix multiplication, and then compare the execution time with the proposed FPGA-based accelerator.

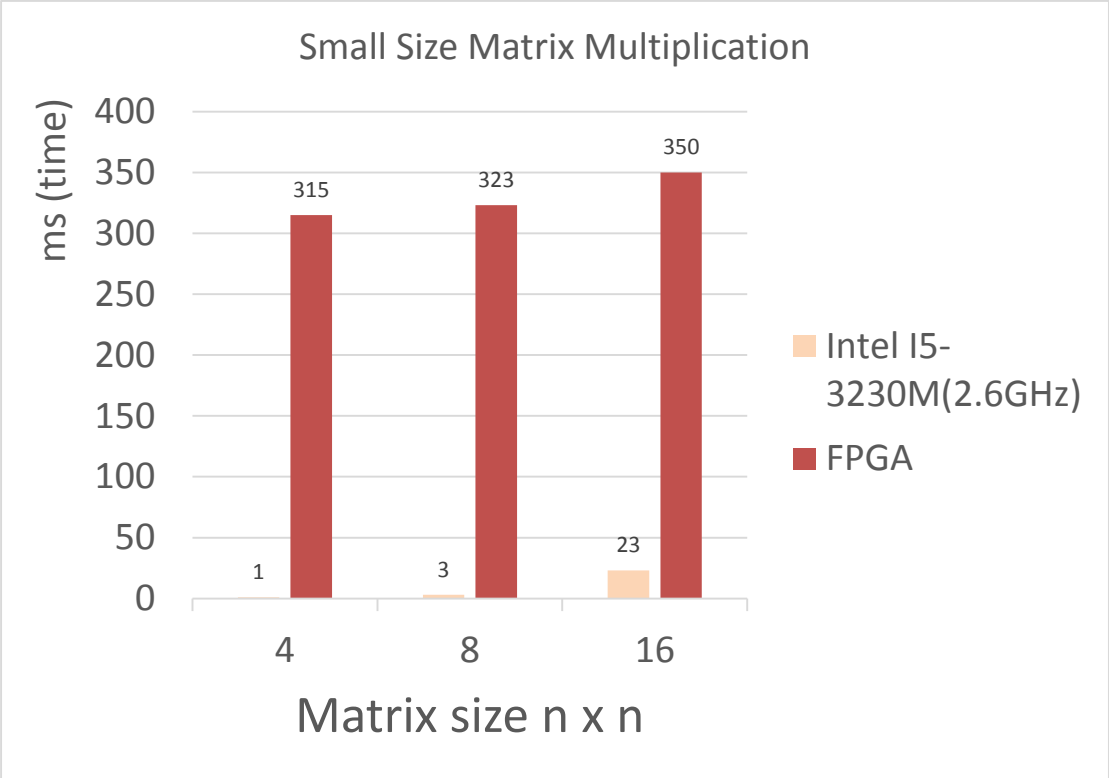


Fig. 4.2 compare Intel I5-3230M (2.60GHz) with FPGA at 4x4, 8x8, 16x16 matrix size

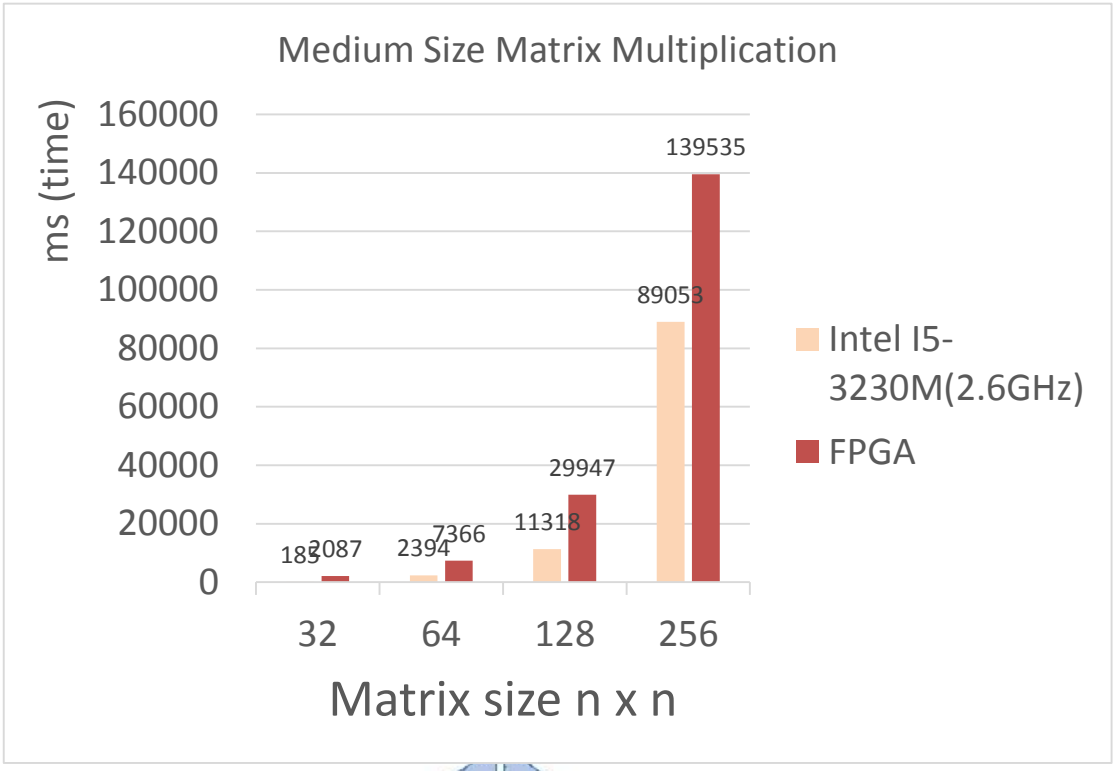


Fig. 4.3 compare Intel I5-3230M (2.60GHz) with FPGA at 32x32, 64x64, 128x128, 256x256 matrix size

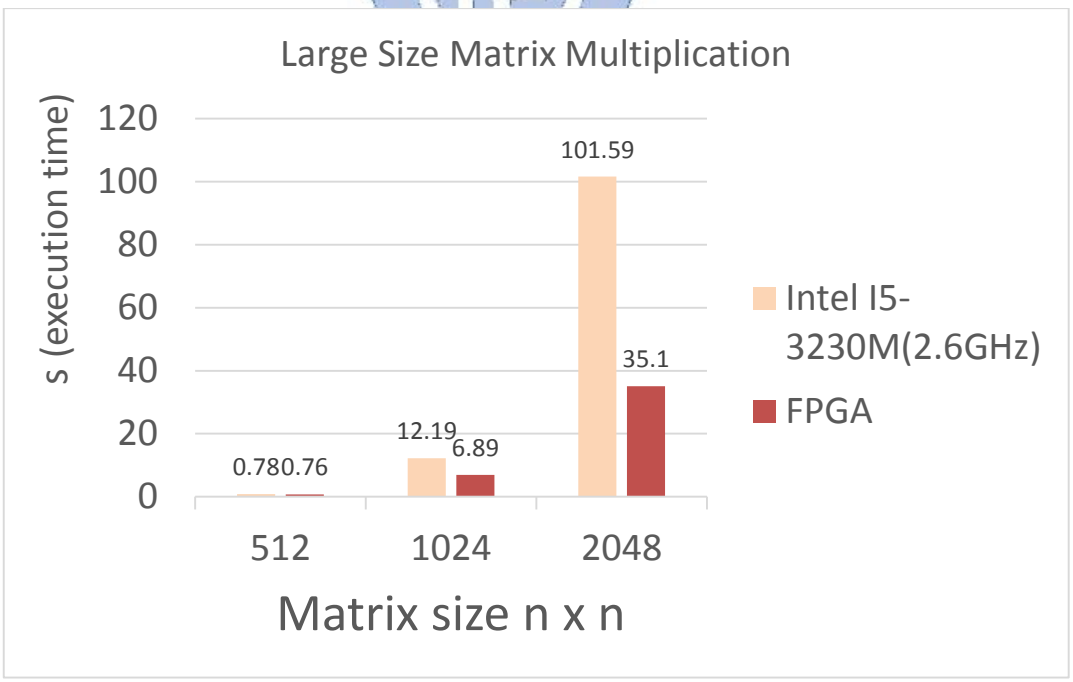


Fig. 4.4 compare Intel I5-3230M (2.60GHz) with FPGA at 512x512, 1024x1024, 2048x2048 matrix size

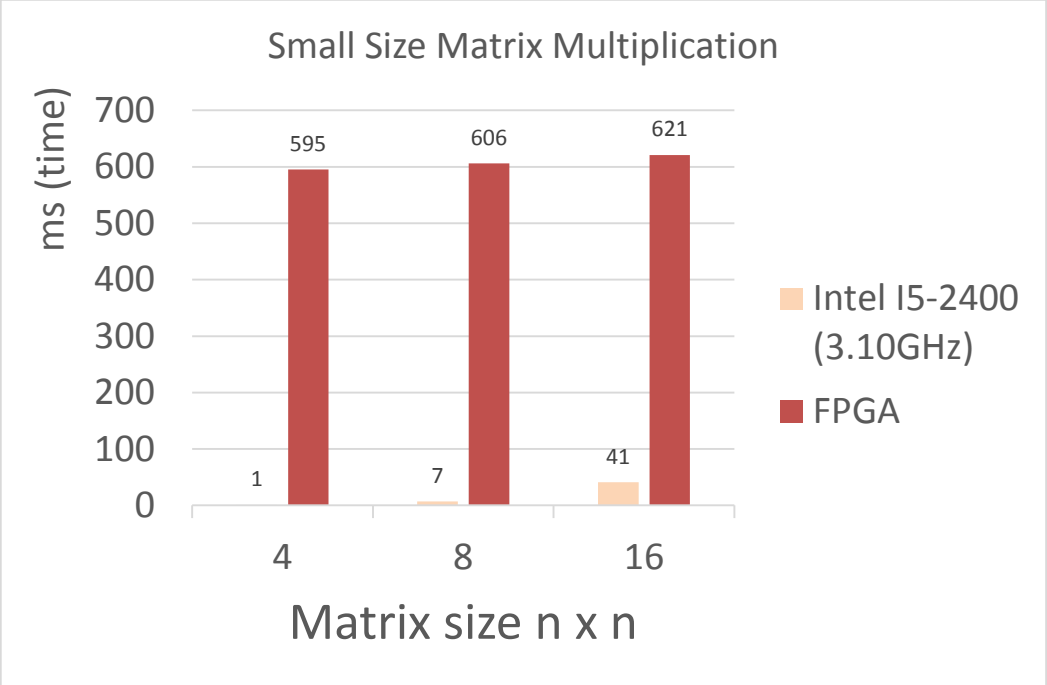


Fig. 4.5 compare Intel I5-2400 (3.10GHz) with FPGA at 4x4, 8x8, 16x16 matrix size

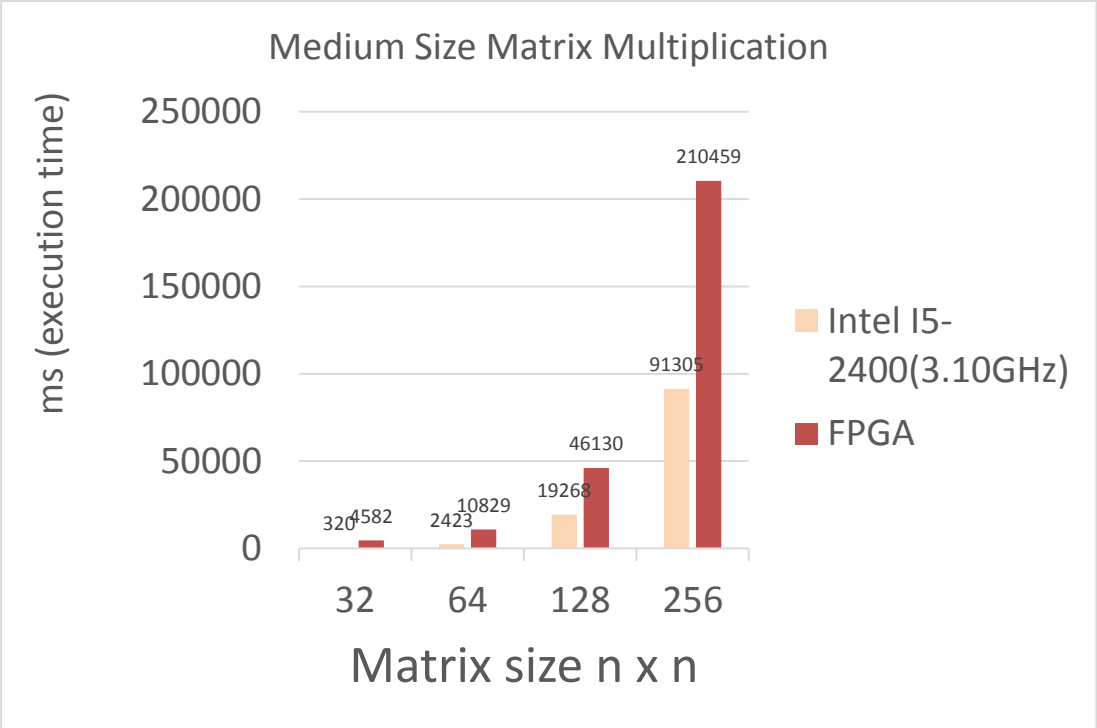


Fig. 4.6 compare Intel I5-2400 (3.10GHz) with FPGA at 32x32, 64x64, 128x128, 256x256 matrix size

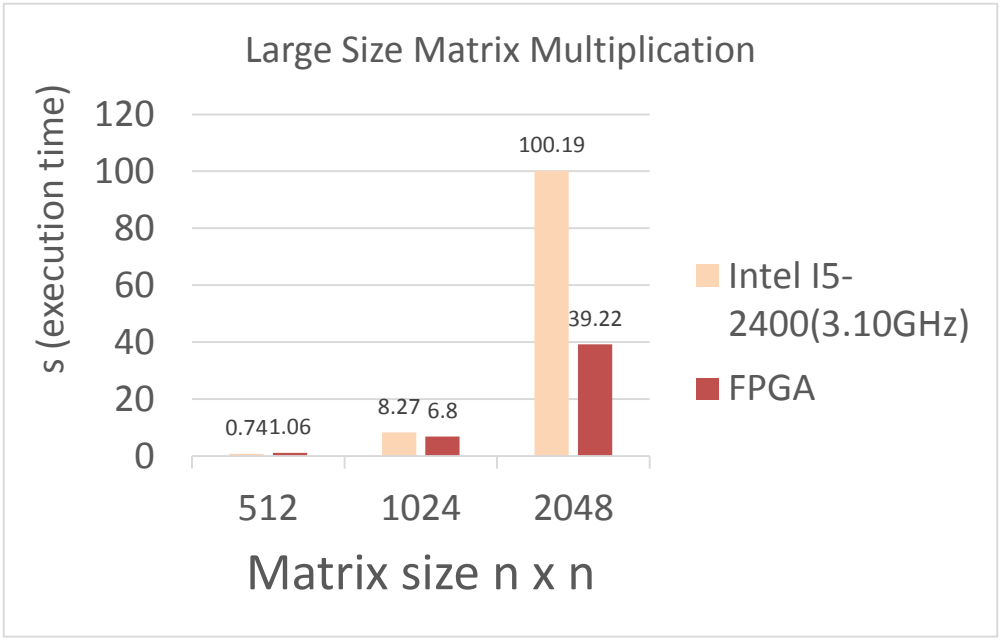


Fig. 4.7 compare Intel I5-2400 (3.10GHz) with FPGA at512x512, 1024x1024, 2048x2048 matrix size

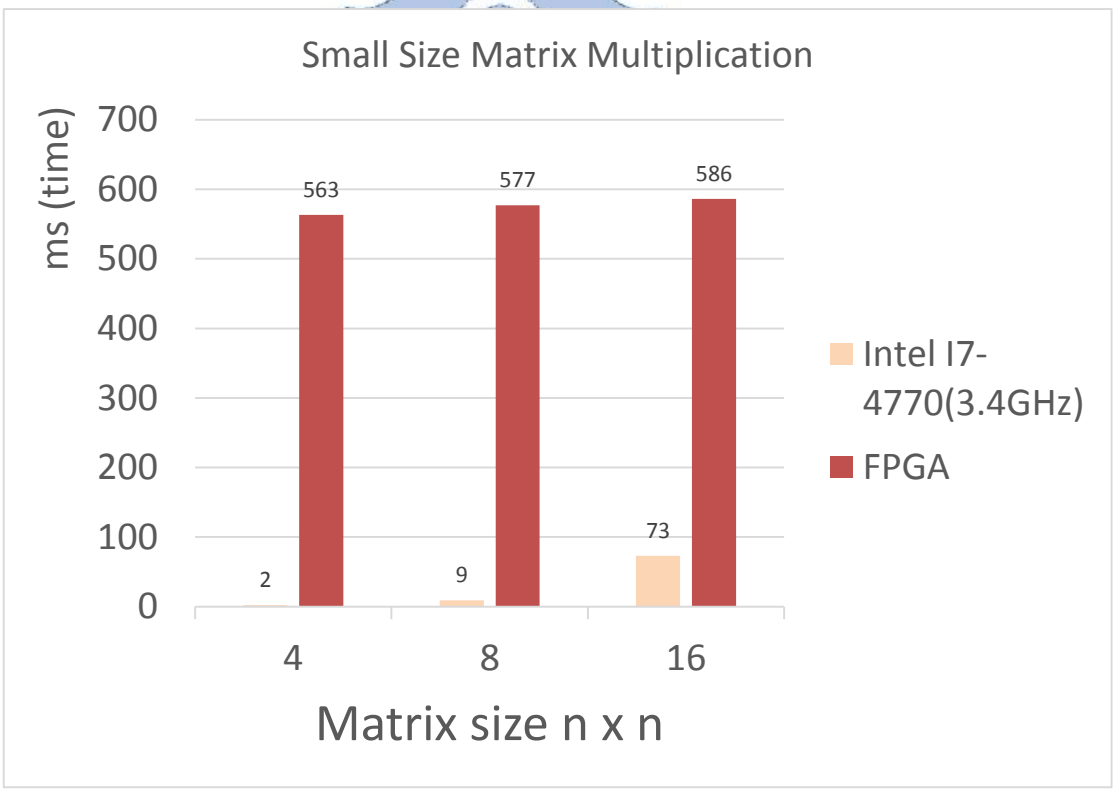


Fig. 4.8 compare Intel I7-4770 (3.40GHz) with FPGA at 4x4, 8x8, 16x16 matrix size

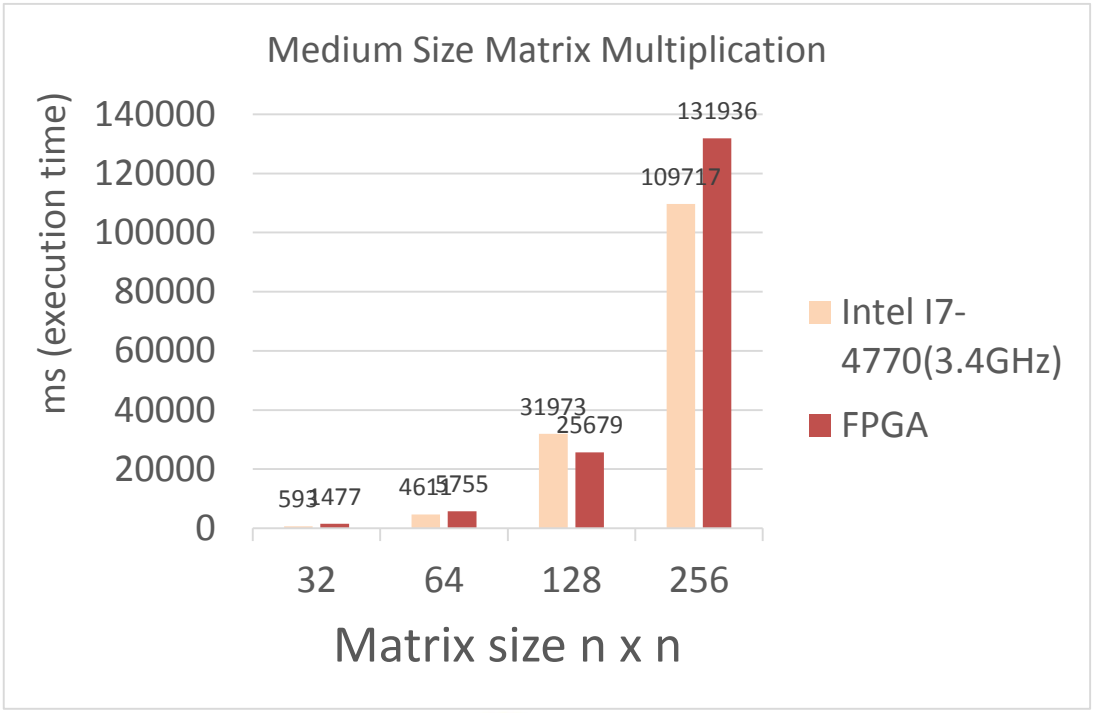


Fig. 4.9 compare Intel I7-4770 (3.40GHz) with FPGA at 32x32, 64x64, 128x128, 256x256 matrix size

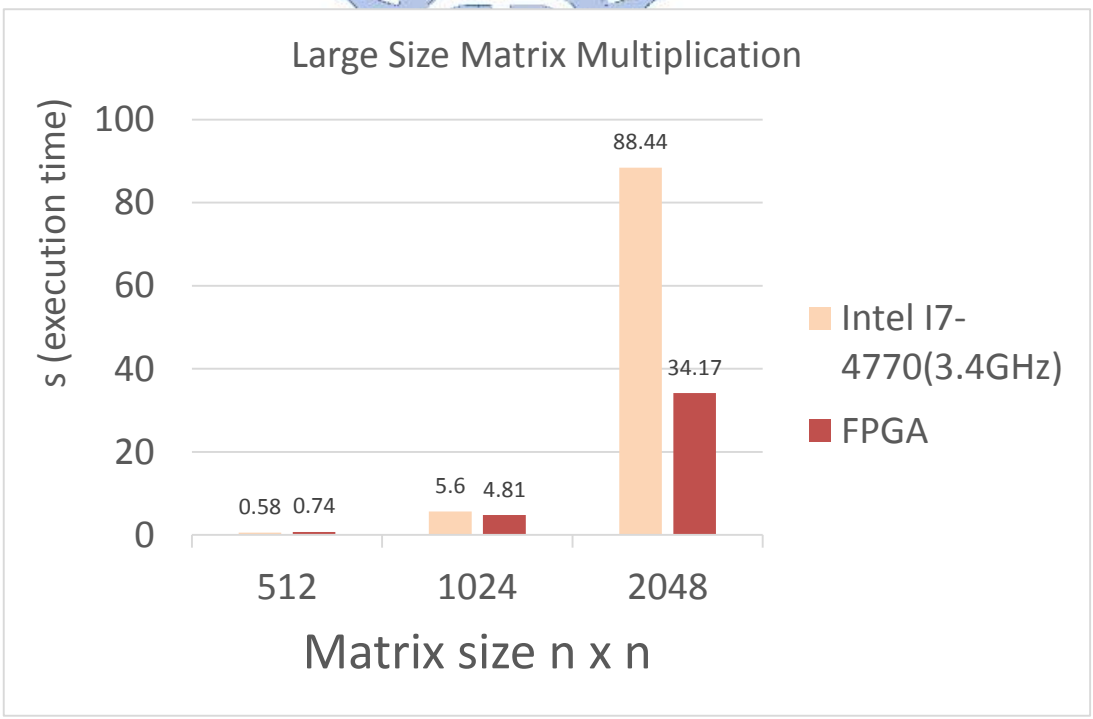


Fig. 4.10 compare Intel I7-4770 (3.40GHz) with FPGA at 512x512, 1024x1024, 2048x2048 matrix size

In Fig. 4.2 ~ Fig. 4.10 we found that if the matrix size is larger than 1024 x1024 then accelerate can speed up the computation time. Due to Ethernet transmission rate is a bottleneck in our hardware accelerator, if the computation time is large enough to cover transmission time then the proposed accelerator can faster than PC. In addition, accelerator is slower than PC in small computation that is acceptable, because we don't use the accelerator in a small size matrix computation.

In Fig. 4.11 ~ Fig. 4.18, we use two different computers to compute different matrix sizes in matrix multiplication of various numbers, and then compare the execution time with the proposed FPGA-based accelerator.

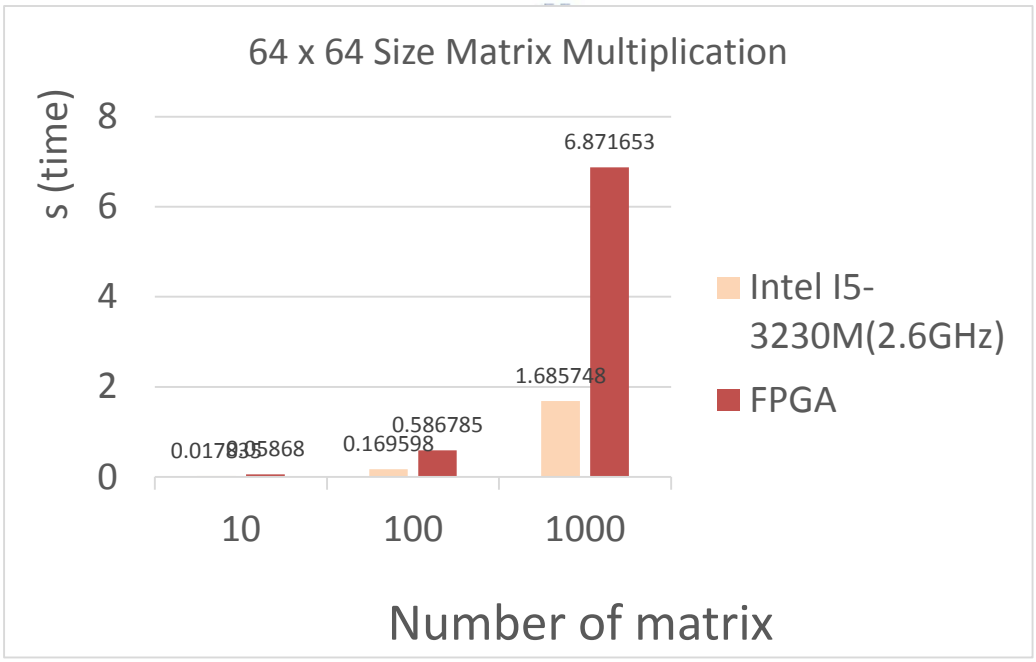


Figure 4.11 is the comparison at 64*64 size matrix of 10 /100 /1000 between Intel I5-3230M (2.6GHz) and FPGA

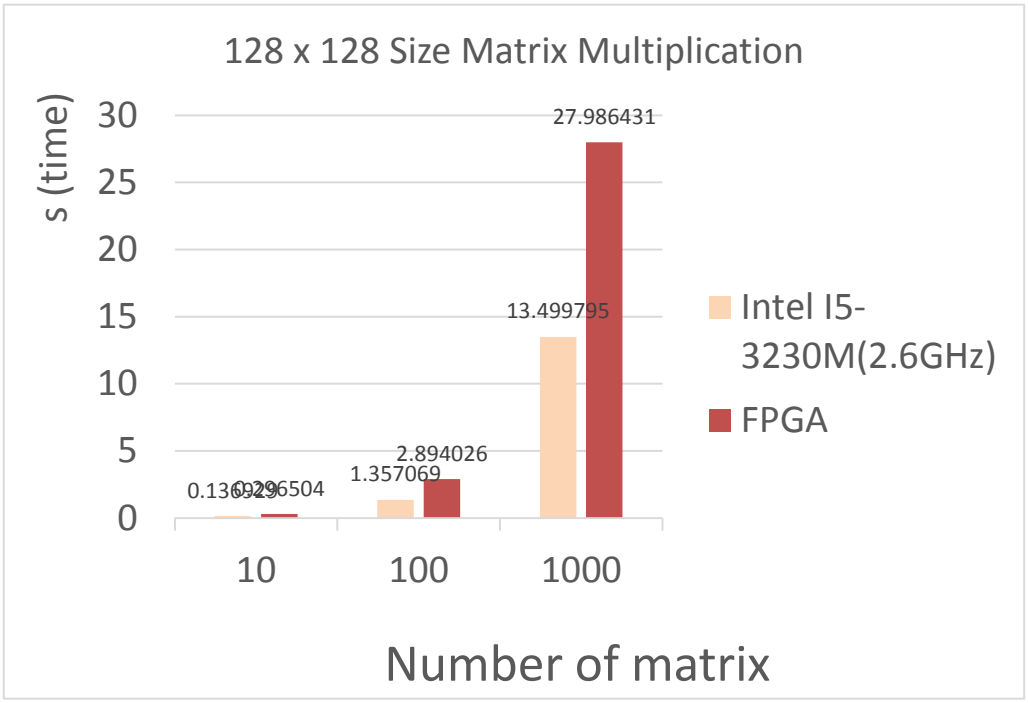


Figure 4.12 is the comparison at 128*128 size matrix of 10 /100 /1000 between Intel I5-3230M (2.6GHz) and FPGA

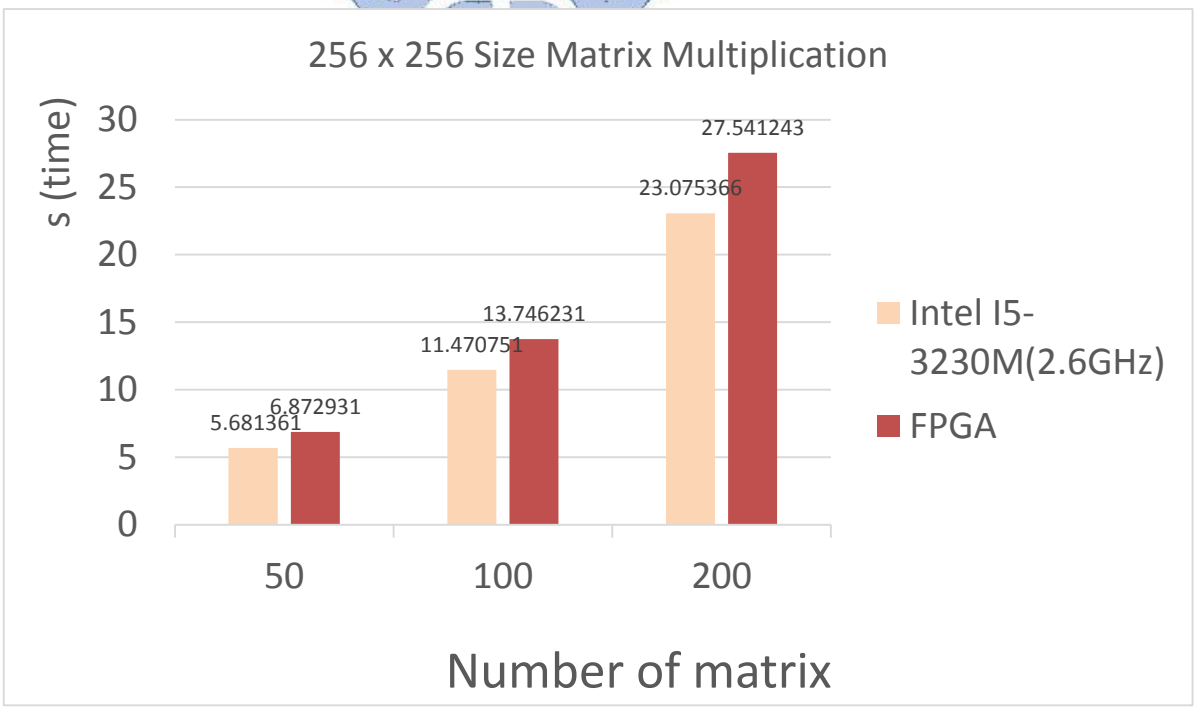


Figure 4.13 is the comparison at 256*256 size matrix of 50 /100 /200 between Intel I5-3230M (2.6GHz) and FPGA

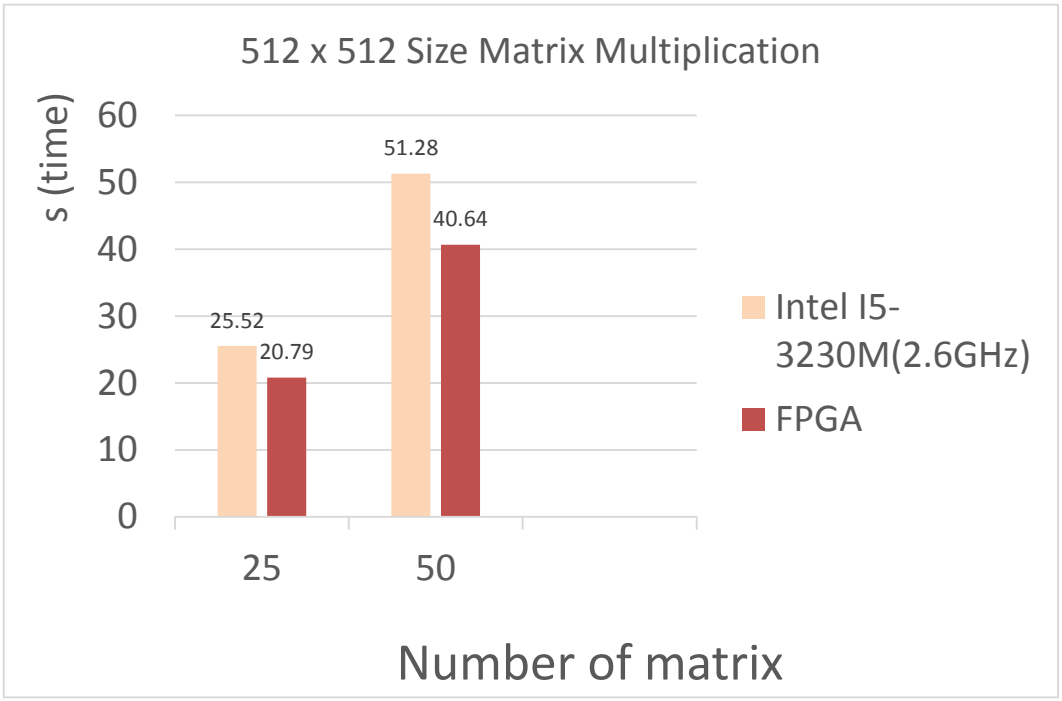


Figure 4.14 is the comparison at 512*512 size matrix of 25 /50 between Intel I5-3230M (2.6GHz) and FPGA

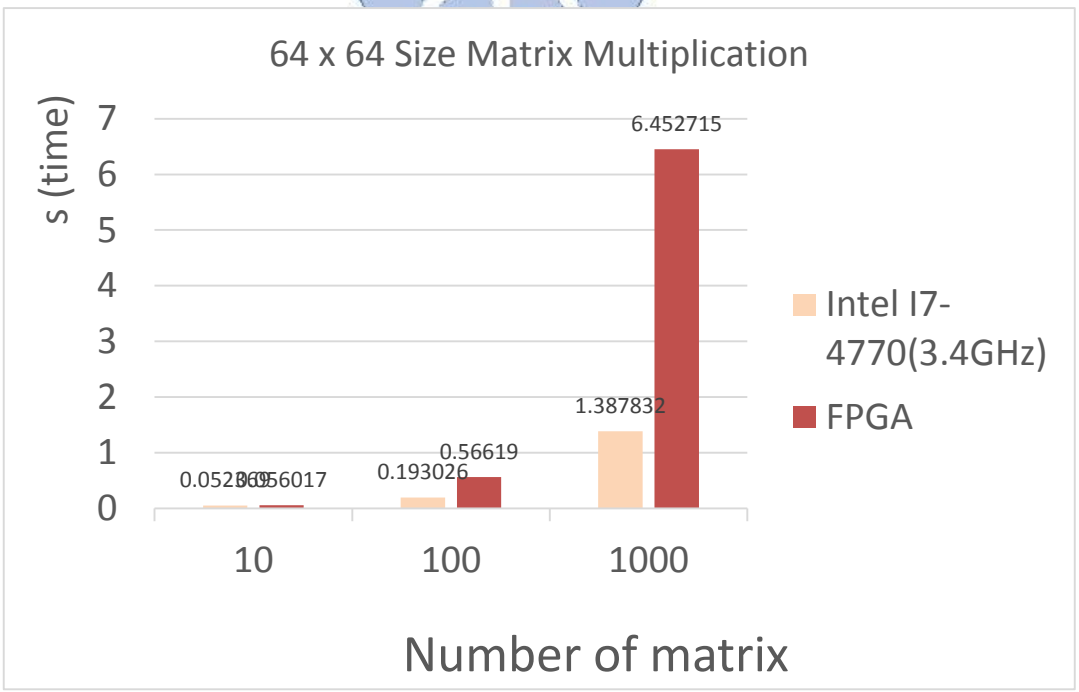


Figure 4.15 is the comparison at 64*64 size matrix of 10 /100/1000 between Intel I7-4770 (3.4GHz) and FPGA

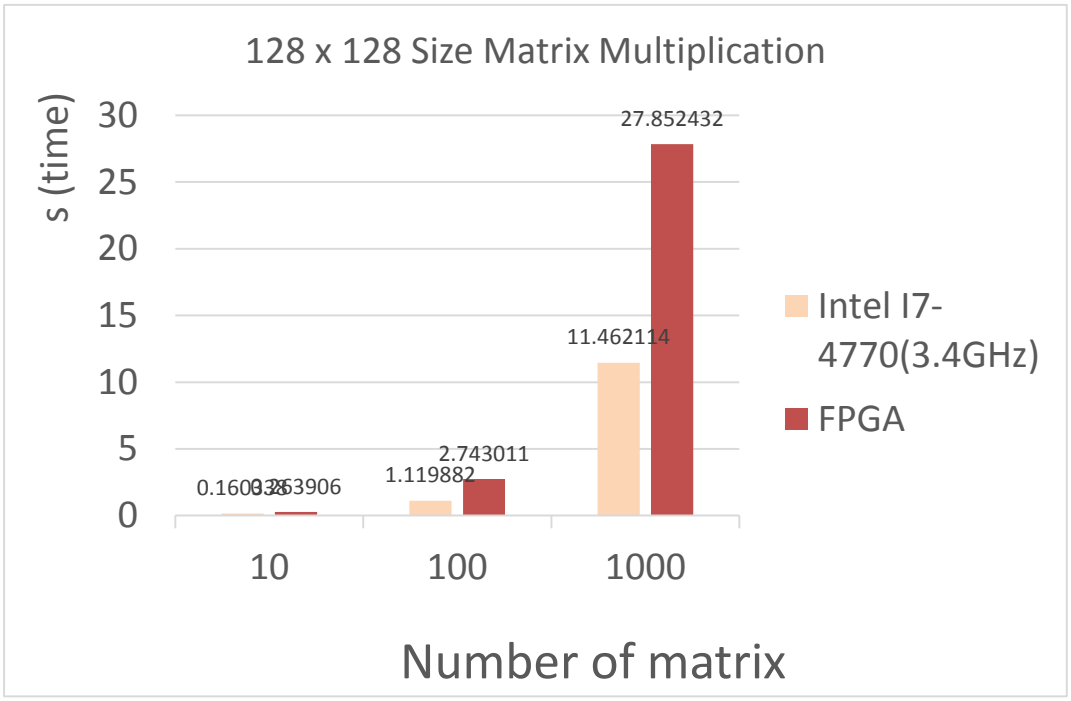


Figure 4.16 is the comparison at 128*128 size matrix of 10 /100/1000 between Intel I7-4770 (3.4GHz) and FPGA

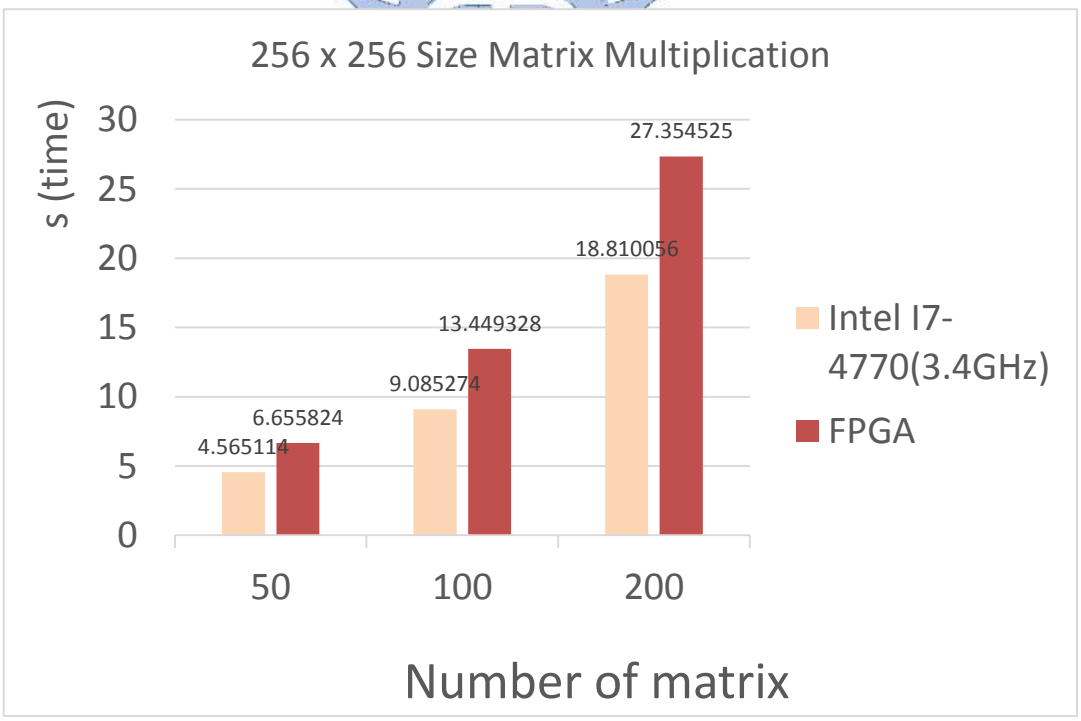


Figure 4.17 is the comparison at 256*256 size matrix of 50 /100/200 between Intel I7-4770 (3.4GHz) and FPGA

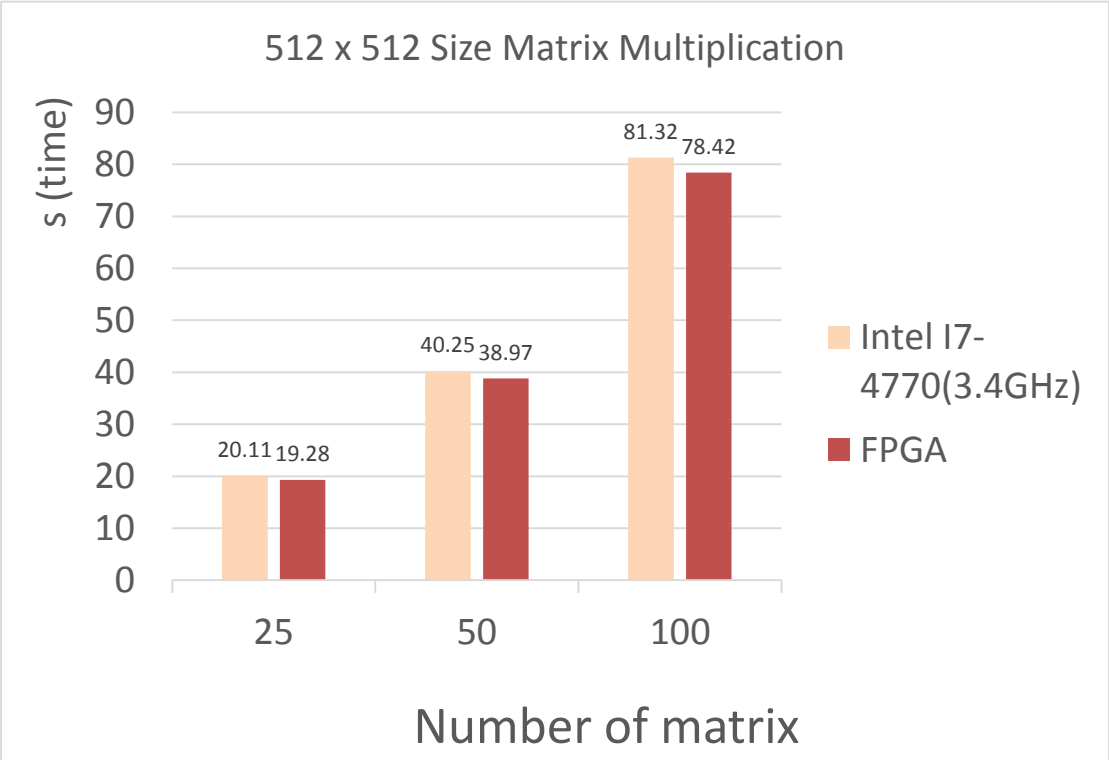


Figure 4.18 is the comparison at 512*512 size matrix of 25 /50/100 between Intel I7-4770 (3.4GHz) and FPGA

In Fig. 4.11 ~ Fig. 4.18，we can find that the larger computations will cause the performance of FPGA equal to or larger than the performance of PC. However, because of the waiting mechanism of Tx FIFO overflow in FPGA, the larger computations cause the longer waiting time such that the performance of FPGA will decreases.

Table 4.3 shows time analysis that the PC and FPGA respective compute 30 matrix those size is 512 x 512, and table 4.4 shows time analysis that the PC and FPGA respective compute 30 matrix those size is 16 x 16.

I7-4770 (3.4GHz)				
Total execution time	24.943997s	Memory read time	18.31459 s	73.42 %
		Memory write time	0.033172 s	0.13 %
		Computing time	6.592300 s	26.46 %
FPGA (125MHz)				
Total execution time	23.973594s	The time that PC send packets to FPGA	9.974400 s	41.62%
		The time that FPGA send packets to PC	0.658349 s	2.74%
		The time of memory read in FPGA board	8.842444 s	36.88%
		The time of memory write in FPGA board	0.028999 s	0.12%
		FPGA computing time	4.469401 s	18.64%

Table 4.3 is the time analysis that the PC and FPGA respectively compute 30 matrix those size is 512 x 512

I7-4770 (3.4GHz)				
Total execution time	0.003181 s	Memory read time	0.002224 s	69.93 %
		Memory write time	0.000139 s	4.38 %
		Computing time	0.000816 s	25.67 %
FPGA (125MHz)				
Total execution time	0.009668 s	The time that PC send packets to FPGA	0.009031 s	93.41 %
		The time that FPGA send packets to PC	0.000300 s	3.10 %
		The time of memory read in FPGA board	0.000285 s	2.94 %
		The time of memory write in FPGA board	0.000028 s	0.28 %
		FPGA computing time	0.000023 s	0.23 %

Table 4.4 is the time analysis that the PC and FPGA respectively compute 30 matrix those size is 16 x 16

In table 4.3 and table 4.4, PC computes the multiplication of big size matrix that takes a lot of time to reads data from memory. If the amount of data of matrix is larger than the capacity of cache in PC, then PC will reads the data from memory that is called cache miss. Reading data from memory that takes too many time that is more than the time of reading data from cache. FPGA computes the matrix multiplication that takes a lot of time to send data between PC and FPGA. According above, we only compare the computing time of PC and computing time of FPGA.

4.3 FPGA Utilization

Table 4.5 shows the slice utilization of the proposed accelerator without matrix multiplication circuit. Table 4.6 shows the slice utilization of the proposed accelerator with matrix multiplication circuit.

Table 4.5 Accelerator Environment without Matrix Multiplication Utilization

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	60,054	607,200	9%
Number of Slice LUTs	52,234	303,600	17%
Number used as logic	50,135	303,600	16%

Table 4.6 Accelerator Environment with Matrix Multiplication Utilization

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	194,312	607,200	32%
Number of Slice LUTs	151,370	303,600	49%
Number used as logic	136,356	303,600	44%

Chapter 5 Conclusion and Future Works

5.1 Conclusion

In recent years, Big Data analysis becomes a serious problem in the data storage, management and analysis. When the PC processes big data problems, they often faces the execution time too long and use too much storage space problem. One of big data challenges are image analyzing and processing. The processing of image or video usually includes a lot of matrix operations. If we send parallel matrix operations such as matrix multiplication to FPGA, it can save a lot of execution.

In this thesis, we implement a FPGA-based hardware accelerator and design a floating-point matrix multiplication in accelerator to help PC to reduce the execution time of computation huge matrix multiplication.

The Ethernet transmission rate is a bottleneck for our accelerator architecture. If the computation is large enough to cover transmission time then accelerator can be faster than PC. In addition, the proposed accelerator is slower than PC in small matrix computation that is acceptable, because we don’t use the accelerator with small matrix computation.

5.2 Future Works

For now, we can only assign the job to one FPGA. If we want to implement a real multi-FPGA-based accelerator, it must write multi-thread program in PC. In this way, PC can parallel assign the job to different FPGA accelerators.

In this thesis, we only implement one function of floating-point matrix multiplication in accelerator, this is not enough to help various computation to enhance the execution time.

According the above, the general purpose multi-FPGA-based accelerator are necessary in the future.

References

[1] Udaigiri Chandrasekhar, Amareswar Reddy and Rohan Rath, ”A Comparative Study of Enterprise and Open Source Big Data Analytical,” *in Proceedings of IEEE Conference on information & Communication Technologies (ICT)*, Apr. 2013, pp. 372-377.

[2] Jinson Zhang and Mao Lin Huang, “5Ws Model for Big Data Analysis and Visualozation,” *in Proceedings of IEEE Conference on Computational Science and Engineering (CSE)*, Dec. 2013, pp. 1021-1028.

[3] Stamford, “Gartner Says Solving ‘Big Data’ Challenge Involeves More Than Just Managing Volume of Data”, posted on June 27, 2011, http://www.Garthner.com/newsroom/id/1731916

[4] Yi YUAN, Haiyang WANG, Dan Wang and Jiangchuan LIU, “On Interfernce-aware Provisioning for Cloud-based Big Data Processing,” *in Proceedings of IEEE/ACM 21st International Symposium of Quality of Service (IWQoS)*, Jun. 2013, pp. 1-6.

[5] Z. Zheng, J.Zhu and M.R. Lyu, ”Service-Generated Big Data and Big Data-as-a-Service: An Overview, ” *in Proceedings of 2013 IEEE Internation Congress on Big Data (BigData Congress)*, Jun. 2013, pp. 403-410.

[6] C. Ji, Y. Li, W. Qiu, U. Awada and K. Li, “Big Data Procesing in Cloud Computing Enviroments, ” *in Proceedings of 2012 12th International Symposium on Pervasive Systems*, Dec. 2012, pp. 13-15.

[7] X. Wu, X. Zhu, G. Wu and W. Ding, “Data Mining with Big Data, ” *IEEE Transactions on Knowledge and Data Engineering,* vol. 26, no. 4, pp. 97-107, Jan. 2014.

[8] Feng Ye, Zhijian Wang, Fachao Zhou, Yapu Wang and Yuanchao Zhou, “Cloud-based Big Data Mining & Analyzing Services Platform integrating R, ” *in Proceedings of 2013 International Conference on Advanced Cloud and Big Data (CBD),* Dec. 2013, pp. 147-151.

[9] Jie Xu, Cem Tekin and Mihaela van der Schaar, “Learning Optimal Classifier Chains for Real-tine Big Data Mining, ” *in Proceedings of 2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton),* Oct. 2013, pp. 512-519.

[10] Shuliang Wang and Hanning Yuan, “Spatial Data Mining in the Context of Big Data, ” *in Proceedings of 2013 International Conference on Parallel and Distributed Systems (ICPADS),* Dec. 2013, pp. 486-491.

[11] Shweta Pandey and Dr.Vrinda Tokekar, “Prominence of MapReduce in BIG DATA processing, ” *in Proceedings of 2014 Fourth International Conference on Communication Systems and Network Technologies (CSNT),* Apr. 2014, pp. 555-560.

[12] Jie Yang and Xiaoping Li, “MapReduce based Method for Big Data Semantic Clustering, ” *in Proceedings of 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC),* Oct. 2013, pp. 2814-2819.

[13] Xiongpai QIN, Huiju Wang, Furong Li, Baoyao Zhou, Yu Cao, Cuiping Li, Hong Chen, Xuan Zhou, Xiaoyong Du and Shan Wang, “Beyound Simple Integration of RDBMS and MapReduce – Paving the Way toward a Unified System for Big

Data Analytics: Vision and Progress, ” *in Proceedings of 2012 Second International Conference on Cloud and Green Computing (CGC),* Nov. 2012, pp.716-725.

[14] I. Yasri, N.H. Hamid and N.B. Zain Ali, “VLSI Based Edge Detection Hardware Accelerator for Real Time Video Segmentation System, ” *in Proceedings of 2012 4th International Conference on Intelligent and Advanced System (ICIAS),* Jun. 2012, pp. 719-724.

[15] Leonidas G. Bleris, Panagiotis D. Vouzis, Mark G. Arnold and Mayuresh V. Kothare, “A Co-Processor FPGA Platform for the Implementaion of Real-Time Model Predictive Control, ” *in Proceedings of 2006 American Control Conference, Jun. 2006.*

[16] F. Pardo, P. L´opez and D. Cabello, “FPGA-based hardware accelerator of the heart equation with applications on infrared thermography, ” *in Proceedings of International Conference on Application-Specific System, Architectures and Processors,* Jul. 2008, pp.179-184.

[17] http://en.wikipedia.org/wiki/Hardware_acceleratoration

[18] Z. Jovanovic and V. Milutionvic, “FPGA accelerator for floating-point matrix multiplication, ” *Institution of Engineering and Technology (IET) Computers & Digital,* vol. 6, no. 4, pp. 249-256, Jul. 2012.

[19] Kiran Kumar Matam and Viktor K. Prasanna, “Energy-Efficient Large-Scale Matrix Multiplication, ” *in Proceedings of 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig),* Dec. 2013, pp. 1-8.

[20] Nirav Dave, Kermin Fleming, Myron King, Michael Pellauer and Muralidaran Vijayaraghavan, “Hardware Acceleration of Matrix Multiplication on a Xillinx FPGA, ” *in Proceedings of 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign, (MEMOCODE),* May. 2007, pp. 97-100.