IMPLEMENTATION OF EFFECTIVE MATRIX MULTIPLICATION ON FPGA

Xiaoxiao Jiang[1], Jun Tao[2]

1 Department of Electrical Engineering, University of Minnesota,
Twin Cities, USA
2 Department of Computer Science, University of Minnesota
Twin Cities, USA
Jiang311@umn.edu, tao@cs.umn.edu

Abstract

Matrix Multiplication is a basic operation that can be used in many applications of DSP. For raw matrix data cannot feed into Simulink Xilinx block directly, thus a new module needs to be designed to complete the matrix multiplication. The original method is straightforward, while consuming considerable hardware resources. In order to save the consumption, we propose a new method to design the matrix multiplication module on Simulink Xilinx platform, which is also implemented on Spartan 3E FPGA (Field Programmable Gate Array). The main idea of the proposal is to reuse the resource and input the data in serial. In this way, the hardware cost can be dramatically decreased; meanwhile decreased but more time for the computation will be needed.

Keywords: Matrix Multiplication; FPGA; Hardware Resource

1 Introduction

As a basic mathematic operation, matrix multiplication, which involves a large number of addition and multiplication blocks with the complexity of $O(n^3)$, is extensively used in varieties fields of engineering[1]. In recent years, Field Programmable Gate Arrays (FPGAs) have become a platform of choice for hardware realization of computation intensive applications [2] and have been improved considerably in speed, density, and functionality, which makes them ideal for system-on-a-programmable-chip designs for a wide range of applications[3]. Traditionally, matrix multiplication operation is often performed by parallel processing systems which distribute computations over several processors to achieve significant speedup gains[4].

In this paper, we will first introduce the original parallel module and then present a new serial method to reduce the hardware resource. Finally we will demonstrate and analyze the hardware cost of each module.

2 Algorithm and module

The product of two matrices can be achieved by multiplying and accumulating the row elements of the first matrix and the column elements of the other. Generally speaking, the computation of a matrix multiplication, C = A×B, is represented as Eq. 1.

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj} \tag{1}$$

Where A =[a_{ij}] B =[b_{ij}], C = [c_{ij}][4].

A number base is obtained on one row in matrix A and one column in matrix B. So the traditional method is be simply designed as Figure 1[4] and the model on Simulink is shown in Figure 2.

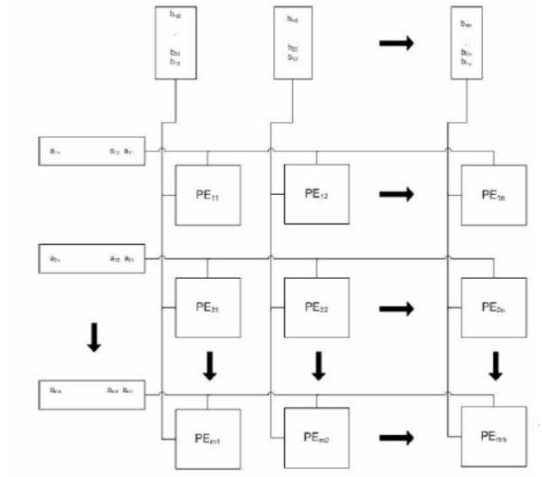

Figure 1 Module of parallel matrix multiplication

978-1-61284-159-5/11/$26.00 ©2011IEEE

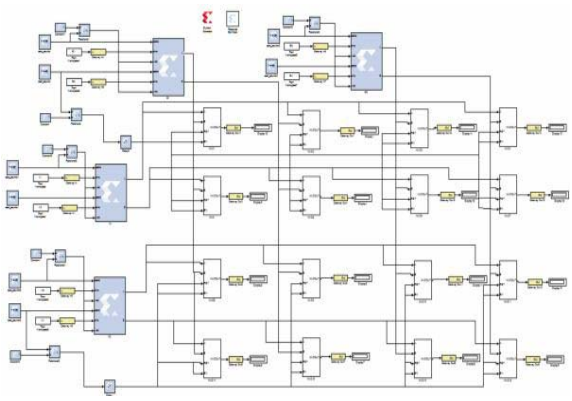

Figure 2 Module of original matrix multiplication (on Simulink)

Taking two 4 by 4 matrices as example, the Dual-RAM can be used to store the rows of matrix A and the columns of matrix B. Since every two rows or two columns need one duel-RAM and every row and columns is input in parallel, we totally need (N+M)/2 RAMs, where N is the number of rows in matrix A and M is the number of columns in matrix B. Besides, N×M MACs are needed for the computation. So for the two 4 by 4 matrices multiplication, we need 4 RAMs and 16 MACs, which are too expensive for the implementation. The structure of the MAC block is designed as Figure 3, which involves one multiplier and one accumulator. The down-sampling block and the enable delay ensure that each element output could be obtained through every four MAC operations (multiply and add).

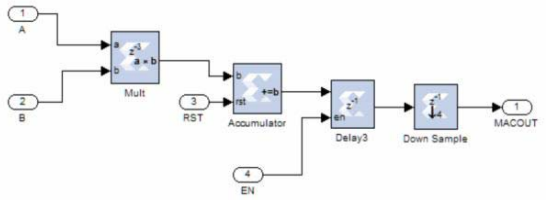

Figure 3 Structure of the MAC block

This method is straightforward and the result could be obtained within less time since each output is computed in parallel. However, considerable hardware resources are consumed. In some complicated algorithms, such as extended Kalman filter, where several operations of matrix multiplication are needed, it is impossible for the implementations to be achieved due to the limited hardware resources. Therefore, an alternate approach should be designed to realize the matrix multiplication with less hardware and power consumptions.

In order to decrease the consumption of the hardware resources, with the main idea of reusing the resources, we designed a new module to implement the matrix multiplication. This new method, which is shown in Figure 4, needs less hardware resources and power than the original one.

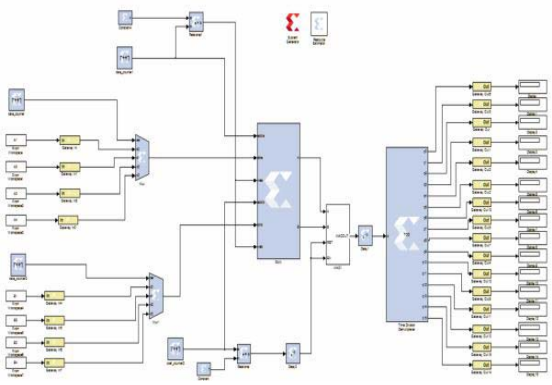

Figure 4 New module of matrix multiplication

In this design, one Dual-RAM is used to store the two matrices. Since the Dual-RAM is separated into two parts, the first part is used to save the matrix A and the second part is to save the matrix B. Two control logics are used to control the input. The input rows and columns are selected through using the counter and MUX blocks. Taking two 4 by 4 matrices as example, we input the matrix as row or column and assume the sample period of the input data is Ts. Every 4×Ts, a row or column can be obtained.

For the first control logic, the sample period of the counter is 16×Ts, which means that every 16×Ts we change one row to another. The counter of the second control logic changes the number at 4×Ts. Since we input the data serial into the RAM, the hardware resource can be dramatically decreased. Only one RAM and one MAC are needed regardless of how large the matrix is. However, more time is required to complete the computation which is the price of the small hardware cost.

3 Hardware resources

In this section, we will demonstrate the hardware resource and power used by different modules. First, we can see the hardware and power are needed in Table 1 based on the original method. In order to reduce the IOBs and complete the hardware implementation, we output the result as 4 lines instead of 16 numbers.

Table 1 Hardware cost of original module

Number of Flip Flops:	5,218	56%
Number of 4 input LUTs:	6,054	65%
Number of occupied Slices:	3,539	76%
Total Number of 4 input LUTs:	6,074	65%
Number of bonded IOBs:	225	96%
Number of RAMB16s:	4	20%
Number of BUFGMUXs:	1	4%
Total Power:	0.432W	
Peak Memory Usage:	314 MB	

In the Table 2, it is the hardware cost and power requirement of the new method and the output is also changed into 4 lines.

Table 2 Hardware cost of new module1

Number of Flip Flops:	365	3%
Number of 4 input LUTs:	452	4%
Number of occupied Slices:	293	6%
Total Number of 4 input LUTs:	458	4%
Number of bonded IOBs:	165	71%
Number of RAMB16s:	1	5%
Number of BUFGMUXs:	1	4%
Total Power:	0.102W	
Peak Memory Usage:	172 MB	

It is obviously that the hardware resources and the power have been decreased significantly with the application of the new methods. For the original module, the occupied numbers of Slices, LUTs and FFs are all more than 50% which means that such design cannot support large size matrix multiplication as well as more than two matrix multiplications. However, with the new modules, the cost of Slices, LUTs and FFs only occupied less than 10%, which is easy to be implemented. On the other side, since we reuse the resource, we need to compute every element of the matrix serial but not parallel. Thus, more time will be spending on the computation. Considering the two 4 by 4 matrix multiplications and assuming we can get all the 16 elements in Ts for the original method, we will need about 16×Ts to obtain the final result for the new method.

In addition, we can choose whether or not to use the embedded multiplications which are provide in the Spartan 3E with 20 embedded multipliers (more embedded multipliers are provided in Vertax-5). Due to limited number of such multipliers, we can also choose not use the embedded multipliers. In this way, the implementation will consume other hardware resources, such as slices, LUTs or FFs to complete the operation.

4 Conclusions

Matrix multiplication can be applied extensively in many areas. Original parallel method of the implementation on Simulink Xilinx Block consumes considerable hardware resources, which makes it hard to be realized. We provided a new method to improve the module in order to save the resources. The basic idea is to reuse the RAMs and MACs. The control logic is needed to select the input line or column. In this way, less hardware is needed but more time is taken to complete the computation. Other methods can be used to optimize the matrix multiplication, such as pipeline technology which has been described in paper[5] and [6].

References:

[1] Bravo, I.; Jimenez, P.; Mazo M.; Lazaro, J.L.; de las Heras, J.J.; Gardel, A. “Different proposals to matrix multiplication based on FPGAs”, In Proceedings of the IEEE International Symposium on Industrial Electronics, December 2007; pp. 1709-1714.

[2] Syed M. Qasim, Ahmed A. Telba and Abdulhameed Y. AlMazroo, “FPGA Design and Implementation of Matrix Multiplier Architectures for Image and Signal Processing Applications”, in IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.2, February 2010.

[3] M. Ceschia, M. Bellato, A. Paccagnella, and A. Kaminski, “Ionbeam testing of ALTERA APEX FPGAs”, in Proceedings of IEEE Radiation Effects Data Workshop, pp. 45–50, Phoenix, Ariz, USA, July 2002.

[4] Syed M. Qasim, Shuja A. Abbasi, “Hardware Realization of Matrix Multiplication using Field Programmable Gate Array”, MASAUM Journal of Computing Vol.1 No.1 August 2009

[5] Abbas Bigdeli, Morteza Biglari-Abhari, Zoran Salcic, and Yat Tin Lai, “A New Pipelined Systolic Array-Based Architecture for Matrix Inversion in FPGAs with Kalman Filter Case Study”, in EURASIP Journal on Applied Signal Processing, Vol 2006, pp. 1–12

[6] Jia Di, J.S.Yuan, “Power-aware pipelined multiplier design based on 2-dimensional pipeline gating”, in Great Lakes Symposium on VLSI, 2003.