electronics

Article

An Accelerator Architecture of Changeable-Dimension Matrix Computing Method for SVM

Ruidong Wu, Bing Liu *, Ping Fu, Junbao Li and Shou Feng

School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin 150001, China; 17B901027@stu.hit.edu.cn (R.W.); fuping@hit.edu.cn (P.F.); lijunbao@hit.edu.cn (J.L.); fengshou@hit.edu.cn (S.F.)

* Correspondence: liubing66@hit.edu.cn

Received: 29 December 2018; Accepted: 26 January 2019; Published: 30 January 2019

Abstract: Matrix multiplication is a critical time-consuming processing step in many machine learning applications. Due to the diversity of practical applications, the matrix dimensions are generally not fixed. However, most matrix calculation methods, based on field programmable gate array (FPGA) currently use fixed matrix dimensions, which limit the flexibility of machine learning algorithms in a FPGA. The bottleneck lies in the limited FPGA resources. Therefore, this paper proposes an accelerator architecture for matrix computing method with changeable dimensions. Multi-matrix synchronous calculation concept allows matrix data to be processed continuously, which improves the parallel computing characteristics of FPGA and optimizes the computational efficiency. This paper tests matrix multiplication using support vector machine (SVM) algorithm to verify the performance of proposed architecture on the ZYNQ platform. The experimental results show that, compared to the software processing method, the proposed architecture increases the performance by 21.18 times with 9947 dimensions. The dimension is changeable with a maximum value of 2,097,151, without changing hardware design. This method is also applicable to matrix multiplication processing with other machine learning algorithms.

Keywords: changeable-dimension matrix computing; field programmable gate array (FPGA); support vector machine (SVM); ZYNQ

1. Introduction

Field programmable gate array (FPGA)-based data processing has advantages, such as high parallelism, fast processing speed, customizable configuration, and high flexibility [1]; thus, it is widely used in digital signal processing [2], deep learning [3], data compression [4], signal acquisition [5], and other fields. The support vector machine (SVM) algorithm is often used in data classification and is prevalent in such fields as pedestrian detection, facial recognition, etc. [6–10]. Thus, an accelerated SVM implementation on FPGA has far-reaching significance and has attracted wide attention. One scheme for SVM acceleration was proposed in [11], in which the computer sends data from the main memory to the external memory on a VC707 board through the Peripheral Component Interconnect express (PCIe) interface. Then, the data are cached into the row and column buffer of the SVM acceleration component. The data enter the acceleration component to perform the calculations of the SVM algorithm. This design achieves a 23x speed-up at a 200 MHz clock frequency. An architecture, to combine two different data transmission modes, was proposed in [12]. The architecture stores the support vectors in the FPGA's internal RAM, and the test vectors are transmitted by the computer to the classifier component through the PCI-X controller. The proposed architecture outperforms other proposed FPGA and graphic processor unit approaches by more than seven times; however, for embedded applications, the application scope of the above architecture is limited, due to factors such as power consumption and volume. To accelerate the SVM algorithm for applications in embedded systems, Kyrkou C et al. built a cascade support vector machine on the Xilinx Spartan6 platform, that was intended for real-time target detection [13], where input data are cached in the data register during processing, and support vectors are stored in the dedicated memory area (memory region). The proposed architecture achieved a real-time processing rate of 40 frames per s for the benchmark face detection application. In [14], a parallel hardware architecture was proposed for real-time image classification. based on the scale-invariant feature transform (SIFT), bag of features (BoFs), and SVM algorithms. In this implementation, a First In First Out (FIFO) buffer is used to store 20 sets of input data and support vectors. The proposed architecture exploits different forms of parallelism in these algorithms in order to accelerate execution and achieve real-time performance. A pedestrian detection framework, with less computational complexity and higher computational accuracy, was proposed in [15], using a Nios core built into the system to control the modules. However, the algorithm implementation also requires a large-capacity linear buffer to save the calculation results. In summary, the traditional SVM algorithm is usually trained with input samples on the computer to obtain a corresponding model. Therefore, the dimension of support vectors are fixed in the implementation of the SVM model on the FPGA. Thus, when the support vectors are obtained by the training changes, the classification algorithm implemented on FPGA is required to be re-configured, thereby limiting the flexibility of SVM algorithms on an FPGA.

In other machine learning areas, convolutional neural networks (CNNs) have different classification and recognition accuracies depending on their size and complexity. Implementing a CNN algorithms on FPGAs has attracted widespread attention [16–21]. Peemen et al. implemented a CNN to perform multiple vision tasks on the Virtex 6 FPGA platform, using on-chip BRAM to buffer and reuse input images, weights, and bias, and achieved an 11x acceleration for the calculations [22]. An accelerator for large-scale CNNs was proposed in [23] that places special emphasis on the impact of memory on accelerator design, performance, and energy. A CNN includes structures, such as convolutional layers, pooled layers, and fully connected layers [24–26]. For traditional implementation, the data is generally stored in off-chip memory, it must be cached into on-chip memory and processed at the processing element. The convolutional layer and fully connected layer perform computations that can be regarded as matrix multiplication. However, because the parameters in each layer differ, the matrix dimension is not fixed during the operation [27]. For example, in AlexNet, the kernel sizes include 3×3, 5×5, and 11×11. The traditional architecture usually uses a specific computational structure that cannot fully utilize the on-chip resources of the FPGA and thereby reduces its performance. As the computational complexity increases, this traditional computing architecture increases the consumption of on-chip RAM resources, which limits system flexibility.

This paper proposes an accelerated architecture of changeable-dimension matrix computing method that stores matrix data in an external memory, which solves the matrix size limitation on the on-chip resources. The micro-controller controls the transfer of the matrix data to the processing element (PE). A handshake protocol is implemented to achieve synchronous matrix data transmission, which ensures the correctness of the processing result. The synthesis and implementation of the proposed architecture are completed on the ZYNQ platform. A processing element for matrix multiplication with changeable dimensions is designed, and the resource usage and performance of the system are also evaluated.

This paper makes the following major contributions:

1. An accelerator architecture of the changeable-dimension matrix computing method is proposed.
2. A mathematical model is established for the time consumed by matrix data transmission, and the acceleration processing element is analyzed and optimized based on this mathematical model.
3. A changeable-dimension matrix-accelerated computing architecture is built on the ZYNQ platform to evaluate the proposed architecture's resource usage and compare its performance.

The remainder of this paper is organized as follows: Section 2 describes the system architecture and explains the various modules of proposed architecture. The principles underlying the SVM algorithm are analyzed in the Section 3, and architecture verification based on the SVM algorithm is designed. Section 4 describes the system implementation on the ZYNQ platform and optimizes the architecture of the processing element. Section 5 compares the resource usage and time consumption between original and optimized PE, and the experimental results are analyzed and discussed. Finally, Section 6 summarizes the content of this paper and future work.

2. System Architecture Design

The architecture for changeable-dimension matrix computing is shown in Figure 1. It includes the micro-controller, communication bus, memory interface, off-chip memory, data mover unit, PE and other peripherals. The micro-controller completes functions, such as reading and writing off-chip memory, configuring the data mover unit, and retrieving the processing status. The micro-controller in the FPGA can be either, soft core (e.g., Nios II from Altera Corporation or MicroBlaze from Xilinx Corporation) or hard core (e.g., the ARM Cortex-A9). The communication bus handles communications between modules and can be an Avalon Bus (Altera), AXI Bus (Xilinx), Wishbone Bus (Silicore), or others. Off-chip memory (SRAM, SDRAM, DDR2 SDRAM, or DDR3 SDRAM) is used to store the matrix data to be processed. The data mover forwards data to the PE to complete data processing; other peripheral interfaces (PIO, UART, etc.) are used to assist the system.

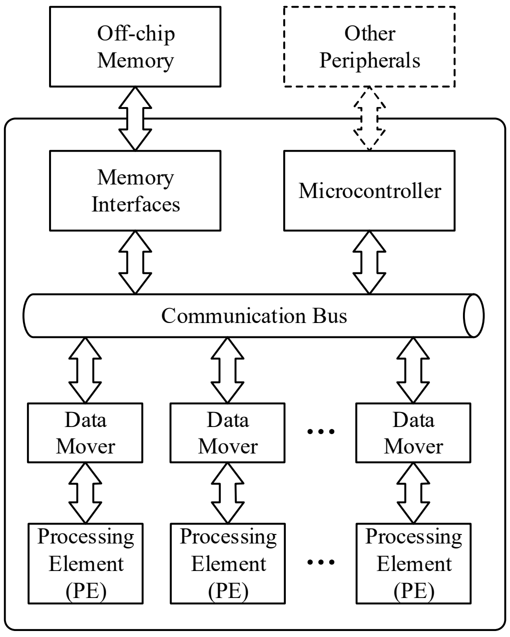

Figure 1. System Block Diagram.

System workflow is shown as follows: Firstly, micro-controller parses input matrix data from off-chip memory, and results are stored in off-chip memory through the communication bus. The data mover is configured in terms of the address and length of matrix data. The matrix data are sent to the PE synchronously by data mover through a communication bus. During matrix processing, the micro-controller only configures the data mover at the first stage and does not participate in the actual processing, which reduces the usage of the micro-controller. Due to the FPGA's parallel nature, multiple data movers can be simultaneously started, further improving the computational efficiency.

As shown in Figure 1, multiple data movers transform matrix data through the communication bus, which may be occupied with data from other movers. In such cases, the data mover stops the reading process until the matrix data has been retransmitted over the communication bus. But the communication bus used in FPGAs has a synchronization mechanism. Here, this paper takes Xilinx's Float Point Unit (FPU) multiply with the AXI4 Stream bus interface as an example. Assume that the inputs of the multiply include row matrix $a = [a_1\ a_2\ \cdots a_m]$ and column matrix $b = [b_1\ b_2 \cdots b_M]^T$.

The outputs are the result of multiplying $a * b$. The waveform of multiply operation is shown in Figure 2, where *aclk* is the system clock. Signals about valid and ready are used as the handshake mechanism. When *s_axis_a_tvalid* and *s_axis_a_tready* are both asserted, the state of *s_axis_a_tdata* is valid. Similarly, column matrix *b* transmits the *s_axis_b_tdata* under the condition that *s_axis_b_tvalid* and *s_axis_b_tready* both asserted. Synchronous processing state is divided into the following stages:

1. T0~T1: *s_axis_a_tready* and *s_axis_b_tready* are both asserted; FPU Multiply is ready to receive data; and *m_axis_result_tready* is also asserted, the result of FPU Multiply can be exported;
2. T1~T2: *s_axis_a_tvalid* is asserted, row matrix a_1 enters FPU Multiply;
3. T2~T3: *s_axis_b_tvalid* is asserted, then column matrix b_1 enters FPU Multiply;
4. T3~T4: *m_axis_result_tvalid* is pulled high, processing result a_1 * b_1 is exported. Next processing state is entered.

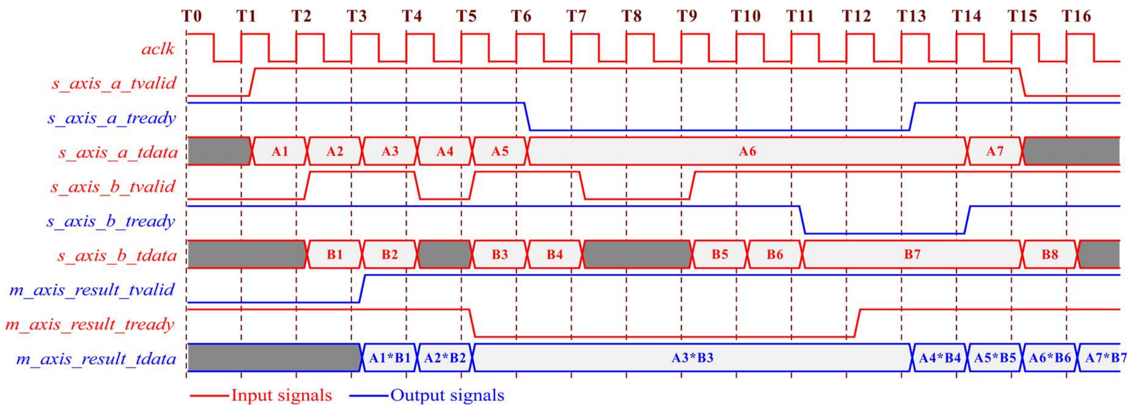

Figure 2. Waveform of the Float Point Unit (FPU) Multiply.

In any clock cycle about Figure 2, if the signals of valid and ready cannot meet the timing requirements (T4~T5), input data and output data are held (T5~T6) until the timing requirement is satisfied for processing (T13~T14). Therefore, the AXI4 Stream protocol can automatically complete the handshake synchronization mechanism to ensure the correct result of FPU Multiply.

In the matrix multiple processing, the total running time T required for processing can be expressed by Equation (1):

$$\begin{cases} T = \left[t_{delay} + \sum\limits_{i=1}^{N} \left(t_{sync\_i} + t_{proc\_i} + t_{latency} \right) \right] \times t_{cycle} \\ M = t_{proc} = \sum\limits_{i=1}^{N} t_{proc\_i} \end{cases} \tag{1}$$

where M is the dimension of the matrix to process, t_{delay} is the clock gap of two sets reaching the FPU Multiply, N is the number of synchronizations of the matrix data during processing, t_{sync_i} is the number of clocks required for the i-th synchronization, t_{proc_i} is the number of clocks processed by the i-th data, t_{proc} is the total number of clocks for matrix processing, which is equal to the matrix dimension M, $t_{latency}$ is the fixed clock latency of FPU Multiply, and t_{cycle} is equal to clock cycle period.

Under the ideal conditions, the input data could reach FPU Multiply at the same time, there is no secondary synchronization during the matrix processing (t_{delay} = 0, N = 1, t_{sync_i} = 0). Therefore, Equation (1) can be expressed as follows:

$$T = \left(t_{proc} + t_{latency} \right) \times t_{cycle} \tag{2}$$

If we use η as the ideal utilization of the FPU Multiply, then the effective utilization ratios η_1 and η_2 corresponding to Equations (1) and (2) can be expressed as follows:

$$\begin{cases} \eta_1 = \dfrac{t_{proc}}{t_{delay}+\sum\limits_{i=1}^{N}\left(t_{sync\_i}+t_{proc\_i}+t_{latency}\right)} \\ \eta_2 = \dfrac{t_{proc}}{t_{proc}+t_{latency}} \end{cases} \tag{3}$$

If the dimension of matrix $M \gg t_{latency}$, the fixed clock latency $t_{latency}$ of FPU Multiply could be negligible (η_2 = 100%). In other words, every clock in FPU Multiply is used for matrix processing. However, due to the time differences at which the input data are transmitted to the FPU Multiply, the data transmission process could be terminated and a second synchronization is required, which causes the decrease in effective utilization. At this point, the effective utilization rate is represented by η_1.

3. Accelerated Architecture Verification Design

To verify the dimensionally accelerated matrix acceleration computing architecture, this paper implements an SVM classification algorithm as an example. SVMs constitute a powerful set of machine learning algorithms that have been widely used in such fields as pedestrian detection, face detection, and so on. The classification decision function for SVM is given in (4):

$$\mathrm{CDF}(z) : \mathrm{sign}\left(\sum_{i=1}^{Nsv} \alpha_i y_i K(z, s_i) + b\right) \tag{4}$$

where N_{sv} is the number of support vectors, α_i is the alpha factor, y_i is the classification label for each sample, z is the input vector, s_i is the i-th support vector obtained from training, $K(z,s_i)$ is the kernel function, and b is the bias. During the prediction, the kernel function occupies a large portion of the computation time [13]. The various kernel functions are as follows:

$$K(z,s) = (z \cdot s) \tag{5}$$

$$K(z,s) = ((z \cdot s) + const)^d, d > 0 \tag{6}$$

$$K(z,s) = \exp\left(-\frac{\|z-s\|^2}{2\sigma^2}\right) \tag{7}$$

The linear kernel function (5) is a dot-product between the input data and support vectors. However, for the nonlinear functions in (6) and (7), the kernels are more complex functions and cannot be processed directly. However, the nonlinear functions (6) and (7) also calculate dot products between the input data and support vectors, which consume considerable computational time. Therefore, improving the matrix computational efficiency is crucial in reducing the computation time of the SVM classification algorithm.

Multi-dimensional matrix multiplication can be decomposed into multiplications of multiple row and column matrices. The multiplication of multi-dimensional matrices can be realized by nested loops. In this paper, a PE architecture for row matrix and column matrix multiplication is designed. For simplicity of algorithm verification, the kernel function in the SVM algorithm uses the linear kernel function. The test sample uses the binary data set provided by SVM Light [28]. The test samples include 600 sets of test vectors, and the trained results contain 9947 sets of support vectors. To facilitate the comparison of algorithmic efficiency, this paper adopts two schemes: software processing and hardware processing. A flowchart of the processing is shown in Figure 3.

Model data and test data are read and parsed from external memory, test vectors and support vectors are stored in the external RAM. The software processing approach uses the micro-controller to complete the matrix multiplication, while the hardware processing method uses PEs to complete the matrix multiplication after the configuring address and dimension of matrix. During the processing of each test vector, this paper uses a timer counter to record time consumption. Finally, the classification accuracy and time consumption of the two methods are also compared.

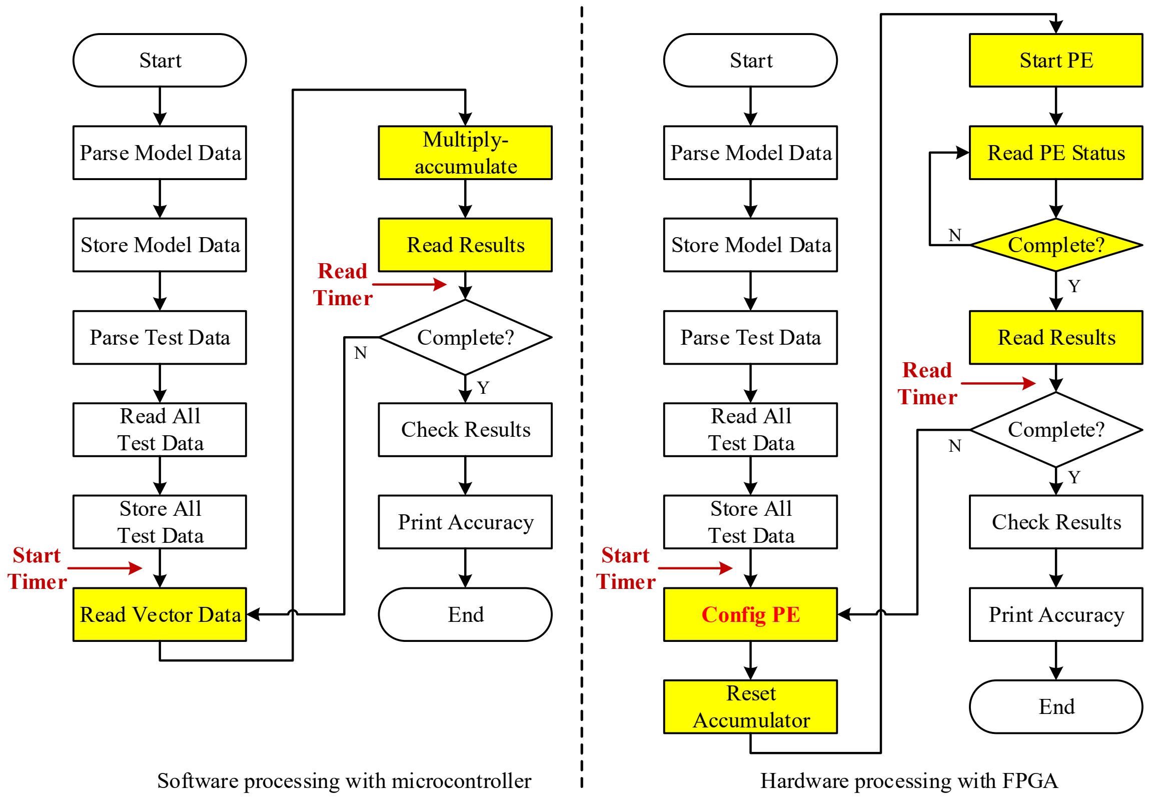

Figure 3. Software flowchart.

4. System Implementation and Optimization

The ZYNQ family is based on the Xilinx System on Chip (SoC) architecture. These products integrate a feature-rich dual-core ARM Cortex-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 CPUs are the heart of the PS and also include on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces. Rich on-chip resources and powerful development tools make it easy to implement complex control functions and accelerate architectural design, while providing room for subsequent functional enhancements. Based on the information provided above, a test platform is built using Xilinx's ZYNQ XC7Z020-CLG484. The changeable-dimension matrix computing architecture is shown in Figure 4, the proposed architecture is divided into two parts: ZYNQ System and off-chip memory. ZYNQ System is divided into the Processing System and Programmable Logic (PL) parts. PS part is used for data parsing and Direct Memory Access (DMA) configuration, and the PL part reads matrix data and performs the processing. Off-chip memory includes SD Card and DDR3 SDRAM, which stores test data and cache data during processing and reduces the consumption of BRAM resources.

The PS part of the ZYNQ System includes two ARM Cortex-A9 CPUs, they are connected to the Central Interconnect and DDR3 Controller through L2 Cache and Controller. An SD Card connected by the SD0 Peripheral in this system is used to store test files. DDR3 Controller allows CPU to access DDR3 SDRAM in the off-chip memory through Memory Interface. PL part implements PE architecture with a data interface for row and column matrix multiplication. As illustrated in Figure 4, there are two ways to exchange data between PS and PL: The General Purpose AXI 32 bit Master Ports (GP) and the High Performance AXI 64 bit Slave Ports (HP). In Figure 4, the arrow direction indicates the master-slave relationship during data transfer. The maximum number of PEs is restricted to 4 owing to the limitation of the bandwidth of DDR3 and the number of HP port. The proposed PE architecture is shown in Figure 5. The Cortex-A9 configures the DMA controller via the AXI4 Lite (32-bit) bus with address and dimension of matrix data. Due to PE contains two sets of data inputs (32-bit), this paper combines two 32-bit sets from the AXI4 Memory Map into a 64-bit AXI4 Memory Map through the AXI Interconnect. The DMA controller reads the data stored in DDR3 through the AXI4 Memory Map (32-bit), and transforms the original data flow into AXI4 Stream (32-bit) to connect FPU Multiply, the multiplication of the matrix data is achieved through this mechanism. DMA controller and FPU can work simultaneously, and when DMA controller reads the next data, FPU performs computing of current data, which forms a pipelining technology. The output of the FPU Multiply is also connected to the FPU Accumulator through AXI4 Stream (32-bit), and the multiplication results is accumulated. A Custom IP core (*WRD_IO32*) is used to reset the FPU state and obtain the final result from FPU Accumulator.

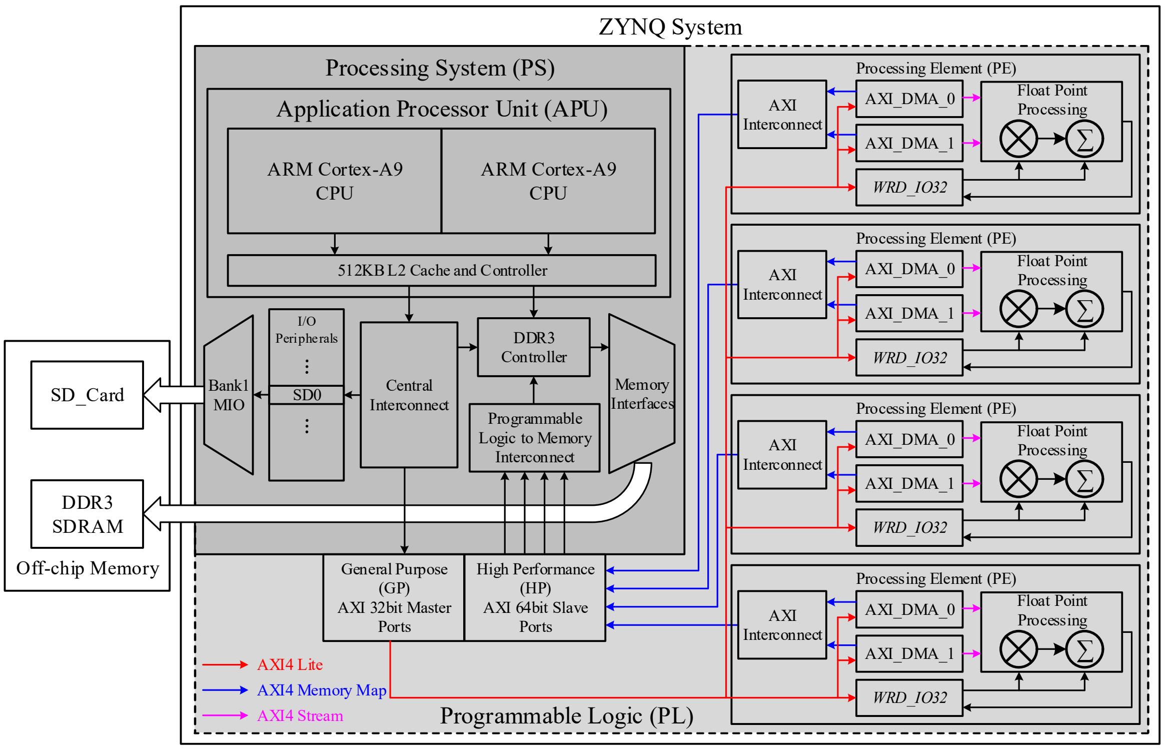

Figure 4. Field programmable gate array (FPGA) architecture of system implementation.

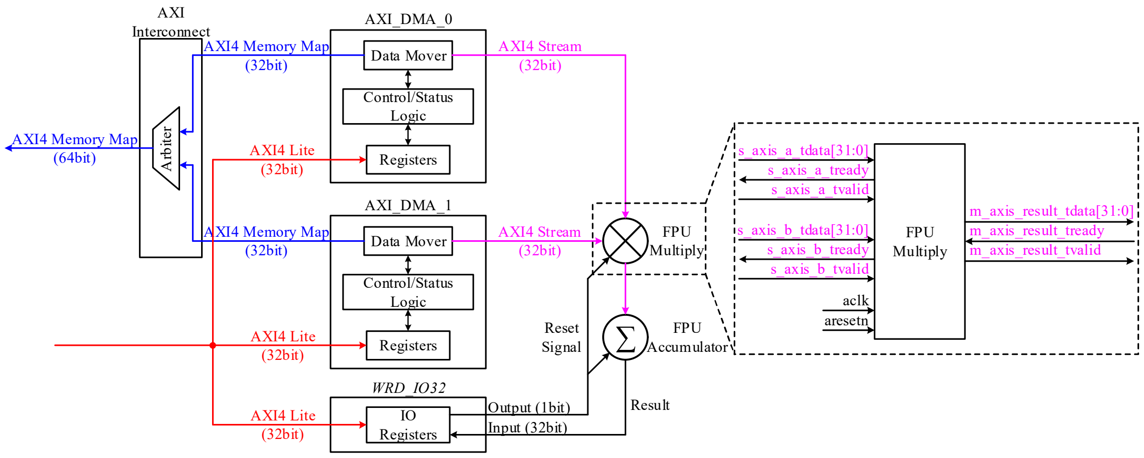

Figure 5. Processing element (PE) architecture without optimization.

In this paper, Vivado2016.4 is used as the integrated development environment to implement the hardware logic design of PL part. The System Integrated Logic Analyzer (System ILA) IP core is used to monitor and collect the status of the internal bus interface. A bitstream file is exported to the Xilinx Software Development Kit (XSDK) to implement the software design of PS part after the hardware design is completed. The DMA controller is monitored by System ILA to obtain the state changes of data bus. The acquisition results of transmission state diagram are shown in Figure 6. *DMA0_Ready* and *DMA0_Valid* are the handshake signals between the FPU Multiply and AXI_DMA_0, and *DMA1_Ready* and *DMA1_Valid* are the handshake signals between the FPU Multiply and AXI_DMA_1. The logical expression is as follows:

DMA0_Res = *DMA0_Ready*&*DMA0_Valid*;
DMA1_Res = *DMA1_Ready*&*DMA1_Valid*;
Result = *DMA0_Res*&*DMA0_Res*.

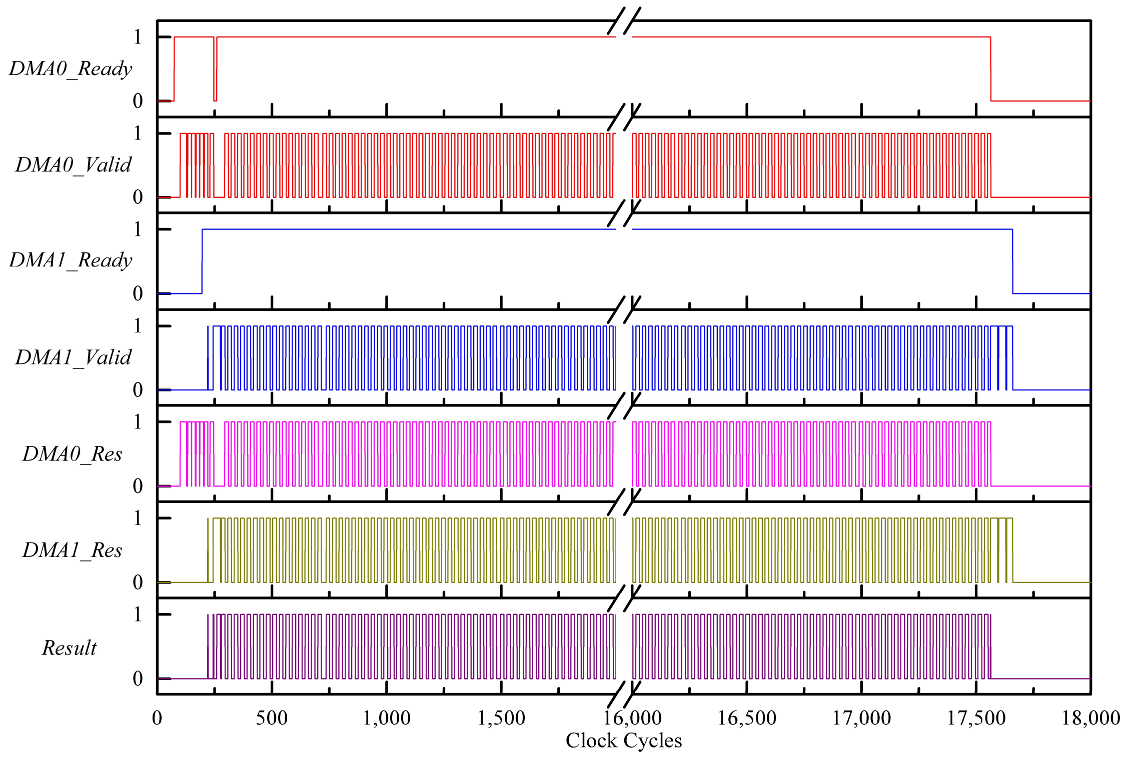

Figure 6. Transmission state diagram of DMA controller.

As shown in Figure 6, the clock interval between the rising edge of *DMA0_Ready* and *DMA1_Ready* corresponds to the parameter t_{delay} in Equation (1). *DMA0_Valid* and *DMA1_Valid* are randomly changed during processing, due to the data transmission synchronization, which indicates the discontinuity of matrix data transmission and leads to the output level changes of *Result*. The number of changes corresponds to the parameter *N* in Equation (1). Data processing begins at the initial rising edge of *DMA0_Ready* and ends at the final falling edge of *DMA1_Ready* in Figure 6. A total of 17,584 clock cycles are required. The first signal-synchronization phase requires 121 clock cycles. There are 618 synchronizations of the resulting signal, corresponding to the parameter *N* in Equation (1). Therefore, the effective utilization ratio η_1 is 56.57% according to Equation (3).

Based on the experimental results shown in Figure 6, the discontinuity of data transmission from DMA controller to the FPU Multiply causes the data transmission to be interrupted multiple times. A second handshake synchronization is required, which reduces the effective utilization of the FPU. In this paper, the structure of the PE in Figure 5 is optimized for the problem of discontinuous data transmission by adopting the following solutions: Adjusting the parameter setting of the AXI Interconnect IP core in Figure 5, adding FIFOs with a depth of 32 between the DMA and the AXI Interconnect, and using a FIFO bit width consistent with the AXI4 Memory Map (32-bit). The optimized PE architecture is shown in Figure 7.

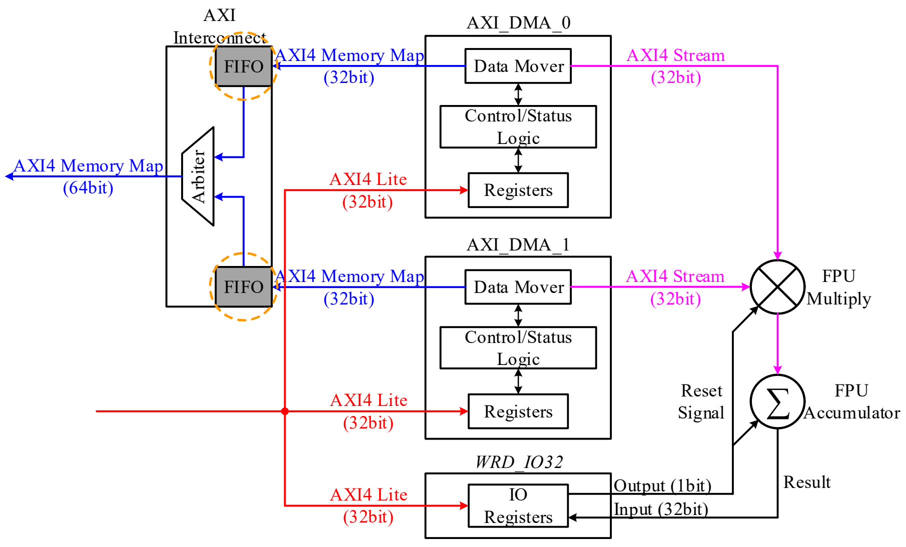

Figure 7. Processing element (PE) architecture with optimization.

An optimized architecture is used to replace the original architecture. Then the test files are reloaded, and System ILA is used to re-acquire the signals of the DMA controller. The optimized result in Figure 8 shows that the data processing clock cycles are reduced from 17,584 to 10,490, and the first signal synchronization also requires 121 clock cycles, which is consistent with the acquisition result before optimization. However, the number of synchronizations of the *Result* are reduced from 618 to 20, the effective utilization ratio η_1 is 94.82%.

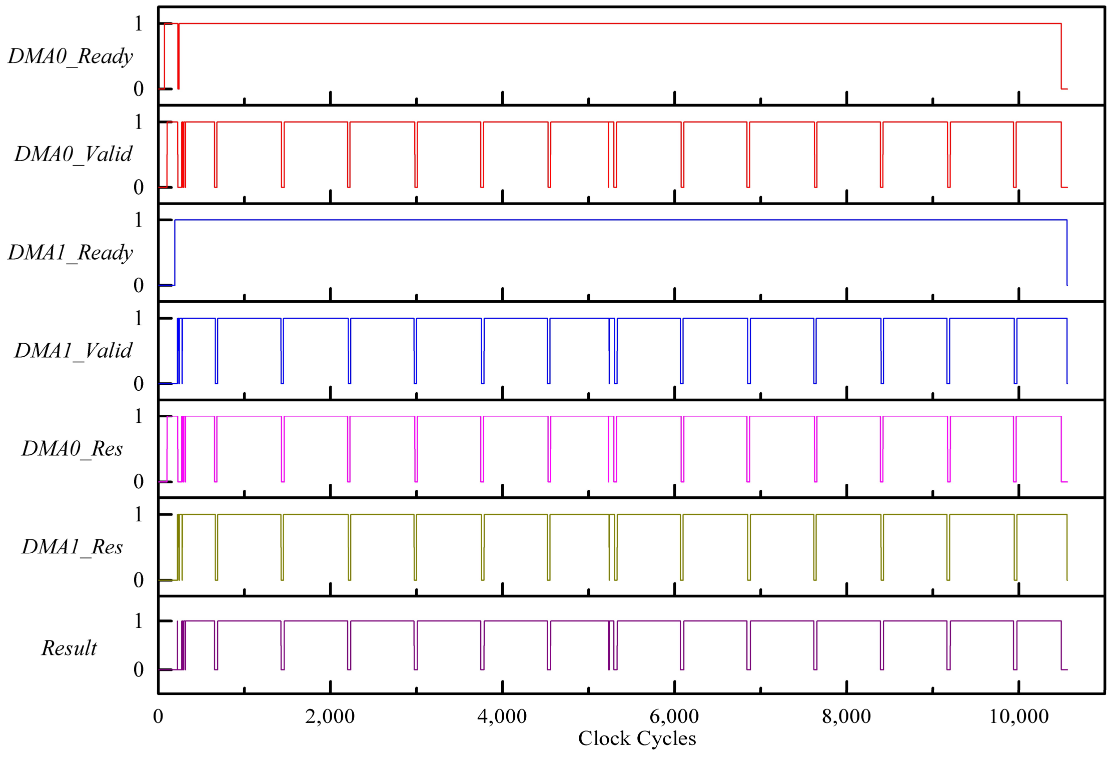

Figure 8. Transmission state diagram of DMA controller with FIFO.

The optimized PE architecture in Figure 8 is compared with the original PE architecture in Figure 6. Although the time interval of the first signal synchronization is a fixed value, as the PS cannot start multiple DMA controllers at the same time, the number of synchronizations are highly decreased.

Therefore, the effective utilization ratio η_1 is increased from 56.57% to 94.82%, which reduces the processing time and improves system efficiency.

5. Experimental Evaluation and Results

5.1. Resource Utilization

This paper builds multiple PEs in this system. The maximum number of PEs is restricted to 4 owing to the limitation of the bandwidth of DDR3 and the port number of HP in ZYNQ system. The bandwidth of DDR3 in this platform is 34,112 Mbps. Assuming that the effective utilization of bandwidth is 80%, the maximum bandwidth is 27,289.6 Mbps. Therefore, the frequency of PE selected in this system is 100 MHz considering the output frequency of internal Phase Locked Loop (PLL). To evaluate the resource utilization of the accelerated computing architecture, place and route of proposed architecture are shown in Figure 9, which is obtained from Xilinx Implemented Design (a design tool of Vivado) with timing constraints. It can be seen that every PE is tightly wrapped around the PS part through AXI Interconnect. The results of place and route make design having better uniformity within the constraint area. Table 1 shows the resource and power utilization of PL with different number of PEs in proposed system. With the number of PEs increasing, the resource utilization of LUT, LUTRAM and FF is increased, and the utilization of the BRAM and DSP are linearly increased among them, but the power consumption of proposed system increases slightly according to the Vivado Power Estimation Tool reports. Therefore, more PEs lead to higher energy efficiency ratio.

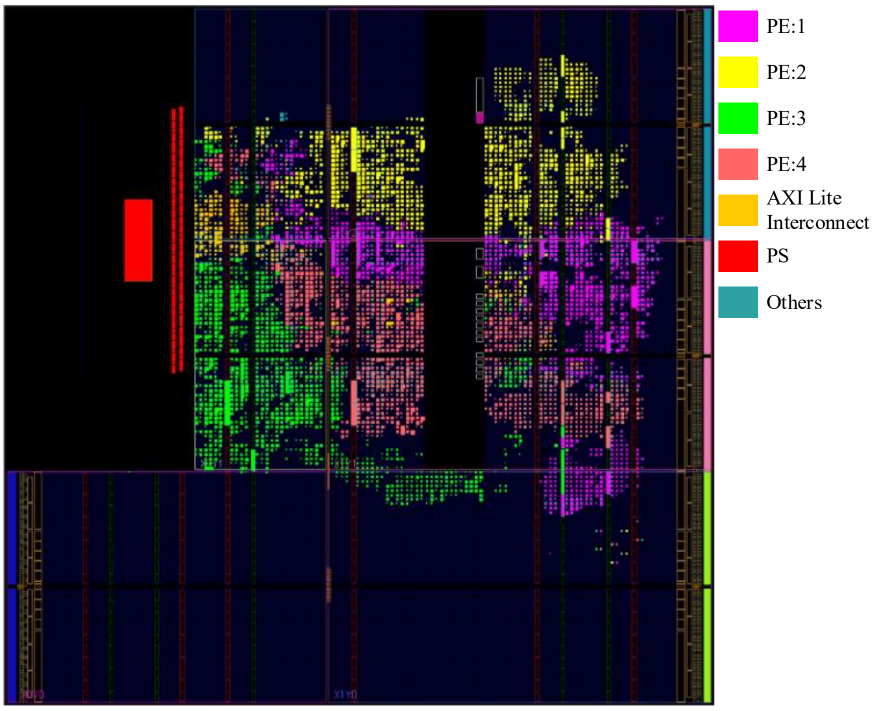

Figure 9. Place and route of proposed architecture inside ZYNQ.

Table 1. Resource utilization of PL.

Number of PEs	1	2	3	4
LUT	3389	6186	9214	12,090
LUTRAM	288	514	740	966
FF	4035	7344	10,649	13,951
BRAM	2	4	6	8
DSP	7	14	21	28
Power	0.164 W	0.165 W	0.171 W	0.175 W

5.2. Timing Comparison of PE

The system's computing performance is evaluated using the test data sets provided by SVM Light. The test contents include software (PS) and hardware (PL) test. This paper uses the system timer provided by the Cortex-A9 core (666.67 MHz) as the reference clock. The frequency of system timer is half of the frequency of Cortex-A9 core, so the clock period of system timer is 3 ns. The test result is the time consumed while processing the 600 sets of SVM classifications, which reflects the time consumption to process 600 sets of matrices. The results are shown in Figure 10.

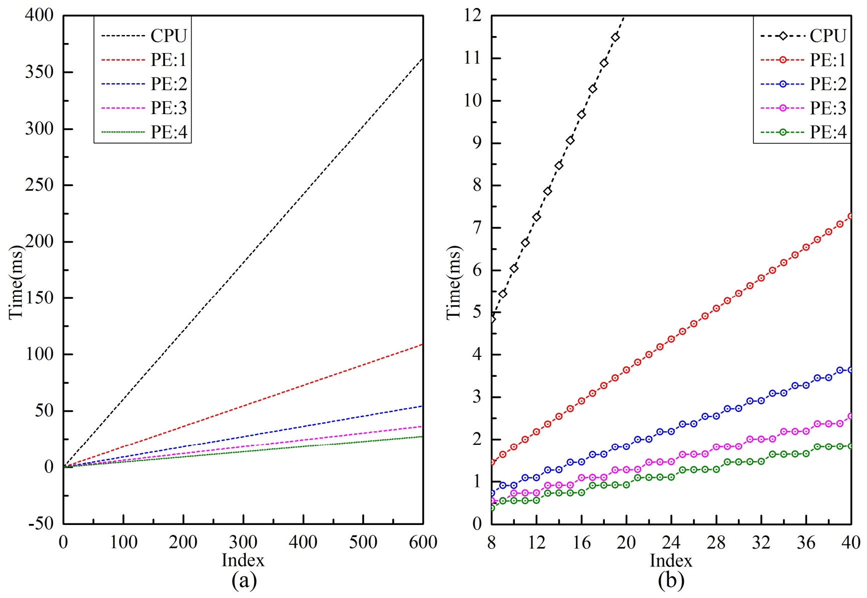

Figure 10. Time consumption graphs of CPU and PE: (**a**) time consumption of processing 600 sets; (**b**) partially enlarged view time consumption of processing 600 sets.

As shown in Figure 10, compared with software process, the hardware process has a smaller slope regarding the time consumption of the matrix computing, and the total time consumption is considerably smaller. The processing method using multiple PEs in PL part, completing multiple sets of matrix operations simultaneously in parallel condition, reduces the time consumption and improves the computational efficiency. Therefore, multiple PEs achieve higher computational performance than the CPU in terms of matrix computing. Next, this study applies the optimized PE architecture.

The performance of optimized PE architecture is also tested and compared with the original PE architecture in Figure 11. Due to the optimized PE has a higher effective utilization ratio, the optimized PE architecture has a smaller time slope than original PE. When multiple PEs are running at same time, the time consumption is reduced, which further improves the computing performance of the acceleration architecture.

A comparison of the time consumption to process 600 sets of matrices by different processing methods is shown in Figure 12. The compilation flag of the CPU is –O0 in XSDK, and the compilation flag of CPU With Optimization is –O3. The running time of CPU With Optimization is better than the time without optimization. However, the PE proposed in this paper is not affected by the compilation flag, which indicates that PE has excellent stability. The result shows that hardware is less than software about time consumption. As the number of PE increases, the time consumption is also proportionally reduced. Taking the optimized PE architecture as an example, four PEs work simultaneously under the same compilation flag, which is 21.18 times higher than the CPU software processing method.

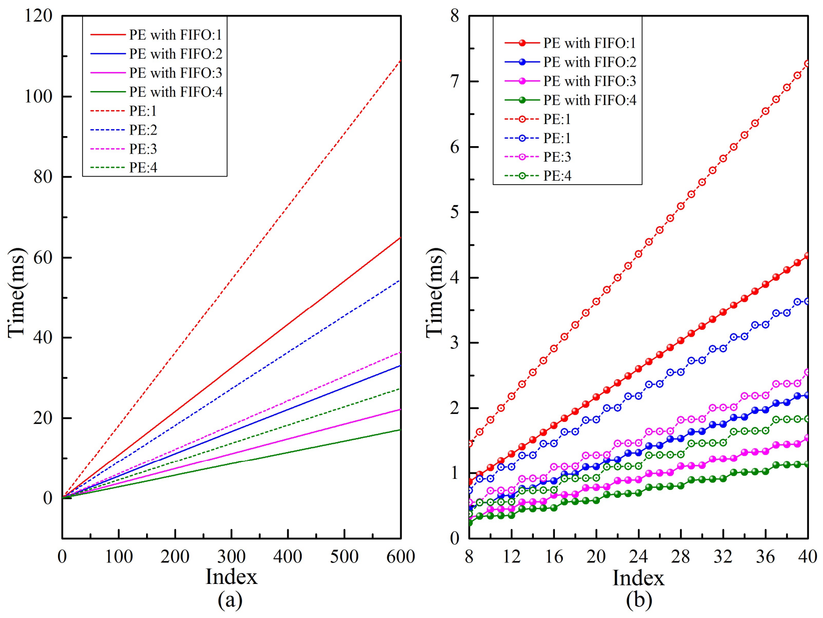

Figure 11. Time consumption graphs of PE and optimized PE: (**a**) time consumption of processing 600 sets; (**b**) partially enlarged view time consumption of processing 600 sets.

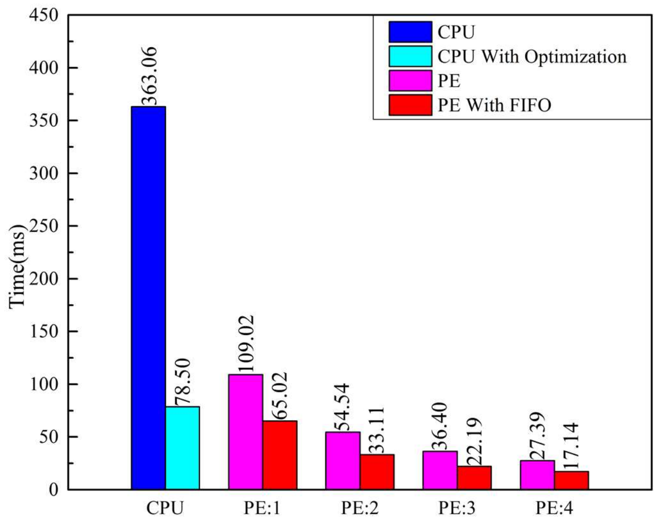

Figure 12. Time comparison chart with different methods.

5.3. Changeable-Dimension Matrix Computing Test

The changeable-dimension matrix computing is tested on the proposed architecture. Multiple data sets from UCI Machine Learning Repository are used in this paper, which include Adult [29], Breast-cancer [30] and Gisette [31]. Other data sets mentioned by LibSVM website are also tested, which include Colon-cancer [32], Rcv1.binary [33] and News20.binary [34]. The dimension of data sets is shown in Table 2, and time in Table 2 is the time consumption for 100 predictions in the condition of using 4 PEs with different data sets. The maximum size of dimensions of PE is dependent on the width of the buffer length register (WBLR) of DMA Controller, WBLR can be set from 8 bits to 23 bits. Therefore, if the data width of DMA Controller is 32 bits, the maximum size of dimensions is 2^{21}-1, which is 2,097,151. As is shown in Table 2, the time consumption is increased with increase in dimensions. But the minimum time is still very large in Breast-cancer, even though the dimension of Breast-cancer is smallest among the test data sets. By analyzing the architecture, it is found that most of time is used to configure PE when the dimension is small, and the time for configuration of DMA Controller can be ignored when the dimension is increased.

Table 2. Time comparison of changeable-dimension Matrix Computing.

Data Sets	Dimension	Time (ns)
Breast-cancer [30]	10	406,320
Adult [29]	119	407,268
Colon-cancer [32]	2000	668,772
Gisette [31]	5000	1,510,443
SVM Light [28]	9947	2,830,377
Rcv1.binary [33]	47,236	13,038,249
News20.binary [34]	1,355,191	371,995,314

The proportion of time and dimension (PTD) with different data sets is also evaluated in this paper. The ideal value of PTD is 250 ns if 4 PEs are synchronized in parallel at 100 MHz. Divide the time and dimension in Table 2 gets the result of Figure 13. When the dimension is smaller, the PTD is very large. As the dimension increases, the PTD gradually approaches the ideal value 250 ns, and the effective utilization ratio is also improved. Therefore, the proposed architecture has an obvious advantage when performing high-dimensional computing.

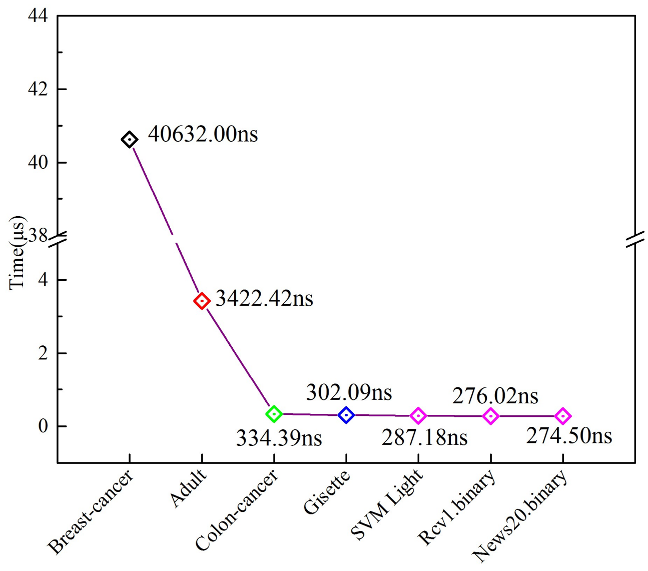

Figure 13. Proportion of time and dimension with different data sets.

5.4. Comparisons with others

The performance and resource utilization of the proposed architecture are compared with those of other FPGA-based SVM computing methods presented in the literature. Table 3 shows that the proposed architecture has a changeable dimension, while the dimension of other papers is generally fixed. Since the dimensions of the matrix are changeable, this paper uses T/D to evaluate the processing time. It can be seen that the method proposed in this paper is superior to others when executing the same dimension. Additionally, compared with other methods in the table, the proposed method consumes the least amount of LUT and BRAM resources. Therefore, the method of this paper can be applied to application areas with different dimensions without changing the hardware design.

Table 3. Comparison about performance and resource utilization with previous works.

Work	[14]	[35]	[36]	[13]	This Paper
Platform	Xilinx Virtex5 LX110T	Xilinx Virtex5 LX110T	Xilinx Spartan3	Xilinx Spartan6 LX150T	Xilinx ZYNQ 7Z020
Dimension	500	361/648/4000	1980	400/1062	Changeable (<2,097,151)
LUT	38,179	57,296	28,616/62,208	35,532/92,152	12,090
BRAM	576	83	100	256/268	8
Frequency	50 MHz	100 MHz	63 MHz	70 MHz	100 MHz
T/D [1]	25.9 µs	69.25 µs	–	62.50 µs	274.50–40,632 ns

[1] T/D is an abbreviation for Time per Dimension.

6. Conclusions

This paper proposes a general accelerator architecture for a changeable-dimension matrix computing method. The matrix data is stored in external memory. The data mover of PL transmits data to the PE with the control of PS, which solves the limited on-chip resources problem regarding the size of the matrix dimension. The transmission adopts a data synchronous mechanism to ensure the continuity of matrix data and the correctness of matrix processing. The synthesis and implementation of the computing architecture are completed on ZYNQ platform, and the architecture of PE is also optimized to improve the data transmission performance and effective utilization ratio of FPU. Multiple PEs are also implemented on the PL, and multiple matrices are processed in parallel to further improve performance. The experimental results show that when executing an SVM classification algorithm, 600 sets of matrix multiplication operations can be processed in parallel using four PEs at 100 MHz, which is 21.18 times faster than performing the processing in software on the CPU.

Furthermore, additional effort is still needed to improve the performance of this architecture. For the PE implemented in our platform, the proposed method depends on the bandwidth of data stream. The bandwidth of DDR3 and number of HP ports are the bottlenecks in this system, and they limit the number and frequency of PEs. A higher bandwidth of port is needed to increase the number and frequency of PEs. This paper only implements a simple multiply-add function in the PE, and does not fully utilize the performance of DSP resources. Further research should be carried out to improve the computing power. It needs to introduce more complex matrix computing to further improve performance.

Author Contributions: R.W. performed the experiments and wrote the paper, B.L. provided the idea of changeable-matrix computing method, P.F. and J.L. contributed analysis tools and ZYNQ platform, S.F. analyzed the data.

Funding: This research was funded by the National Natural Science Foundation of China under Grant No. 61671170, the Open Projects Program of National Laboratory of Pattern Recognition under Grant No.201700019.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FPGA	Field Programmable Gate Array
SVM	Support Vector Machine
PCIe	Peripheral Component Interconnect express
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
FIFO	First In First Out
CNN	Convolutional Neural Network
PE	Processing Element
ARM	Advanced RISC Machines

SRAM	Static Random-Access Memory
SDRAM	Synchronous Dynamic Random Access Memory
DDR	Double Data Rate
PIO	Parallel Input/Output
UART	Universal Asynchronous Receiver/Transmitter
FPU	Float Point Unit
AXI	Advanced eXtensible Interface
DMA	Direct Memory Access
SoC	System on Chip
PS	Processing System
PL	Programmable Logic
ILA	Integrated Logic Analyzer
XSDK	Xilinx Software Development Kit
LUT	Look Up Table
FF	Flip Flop
BRAM	Block RAM
DSP	Digital Signal Processing
CPU	Central Processing Unit
WLBR	Width of Buffer Length Register
PTD	Proportion of Time and Dimension
T/D	Time per Dimension

References

1. Wang, Q.; Yu, S.; Li, C.; Lu, J.; Fang, X.; Guyeux, C.; Bahi, J.M. Theoretical Design and FPGA-Based Implementation of Higher-Dimensional Digital Chaotic Systems. *Ieee Trans. Circuits Syst. I Regul. Pap.* **2016**, *63*, 401–412. [CrossRef]
2. Ricci, S.; Meacci, V. Data-Adaptive Coherent Demodulator for High Dynamics Pulse-Wave Ultrasound Applications. *Electronics* **2018**, *7*, 434. [CrossRef]
3. Cao, Y.; Chen, Y.; Khosla, D. Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition. *Int. J. Comput. Vis.* **2015**, *113*, 54–66. [CrossRef]
4. Jin, R.; Zhao, J.; Ma, Y.; Zhou, F. Scheme for variable-frequency digital circuit with data compression based on block-match process. *IET Circuits Devices Syst.* **2018**, *12*, 295–300. [CrossRef]
5. Liu, X.; Wang, Y.; Wu, R.; Wang, D.; Bai, Q.; Jin, B. Real-Time Phase-Sensitive OTDR Based on Data Matrix Matching Method. *Sensors* **2018**, *18*, 1883. [CrossRef] [PubMed]
6. Ribeiro, E.G.; Mendes, T.M.; Dias, G.L.; Faria, E.R.S.; Viana, F.M.; Barbosa, B.H.G.; Ferreira, D.D. Real-time system for automatic detection and classification of single and multiple power quality disturbances. *Measurement* **2018**, *128*, 276–283. [CrossRef]
7. Ilas, M.; Ilas, C. A New Method of Histogram Computation for Efficient Implementation of the HOG Algorithm. *Computers* **2018**, *7*, 18. [CrossRef]
8. Luo, J.; Lin, C. Pure FPGA Implementation of an HOG Based Real-Time Pedestrian Detection System. *Sensors* **2018**, *18*, 1174. [CrossRef]
9. Bharatiraja, C.; Jeevananthan, S.; Munda, J.L. A Timing Correction Algorithm-Based Extended SVM for Three-Level Neutral-Point-Clamped MLI in Over Modulation Zone. *IEEE J. Emerg. Sel. Top. Power Electron.* **2018**, *6*, 233–245. [CrossRef]
10. Wang, B.; Chen, Y.; Liu, D.; Peng, X. An embedded intelligent system for on-line anomaly detection of unmanned aerial vehicle. *J. Intell. Fuzzy Syst.* **2018**, *34*, 3535–3545. [CrossRef]
11. Venkateshan, S.; Patel, A.; Varghese, K. Hybrid Working Set Algorithm for SVM Learning with a Kernel Coprocessor on FPGA. *IEEE Trans. Very Large Scale Integr. Syst.* **2015**, *23*, 2221–2232. [CrossRef]
12. Papadonikolakis, M.; Bouganis, C. Novel Cascade FPGA Accelerator for Support Vector Machines Classification. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 1040–1052. [CrossRef] [PubMed]
13. Kyrkou, C.; Bouganis, C.; Theocharides, T.; Polycarpou, M.M. Embedded Hardware-Efficient Real-Time Classification with Cascade Support Vector Machines. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *27*, 99–112. [CrossRef] [PubMed]

14. Qasaimeh, M.; Sagahyroon, A.; Shanableh, T. FPGA-Based Parallel Hardware Architecture for Real-Time Image Classification. *IEEE Trans. Comput. Imag.* **2015**, *1*, 56–70. [CrossRef]
15. Bilal, M.; Khan, A.; Khan, M.U.K.; Kyung, C. A Low-Complexity Pedestrian Detection Framework for Smart Video Surveillance Systems. *IEEE Trans. Circuits Syst. Video Technol.* **2017**, *27*, 2260–2273. [CrossRef]
16. Ardakani, A.; Condo, C.; Ahmadi, M.; Gross, W.J. An Architecture to Accelerate Convolution in Deep Neural Networks. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 1349–1362. [CrossRef]
17. Wang, C.; Gong, L.; Yu, Q.; Li, X.; Xie, Y.; Zhou, X. DLAU: A Scalable Deep Learning Accelerator Unit on FPGA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2016**, *36*, 513–517. [CrossRef]
18. Xiao, T.; Qiao, Y.; Shen, J.; Yang, Q.; Wen, M. Unified virtual memory support for deep CNN accelerator on SoC FPGA. In Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing, Zhangjiajie, China, 18–20 November 2015; Springer: Berlin, Germany, 2015; pp. 64–76.
19. Du, L.; Du, Y.; Li, Y.; Su, J.; Kuan, Y.; Liu, C.; Chang, M.F. A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 198–208. [CrossRef]
20. Wang, J.; Lin, J.; Wang, Z. Efficient Hardware Architectures for Deep Convolutional Neural Network. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 1941–1953. [CrossRef]
21. Muller, J.; Wittig, R.; Muller, J.; Tetzlaff, R. An Improved Cellular Nonlinear Network Architecture for Binary and Grayscale Image Processing. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 1084–1088. [CrossRef]
22. Peemen, M.; Setio, A.A.; Mesman, B.; Corporaal, H. Memory-Centric Accelerator Design for Convolutional Neural Networks. In Proceedings of the 31st IEEE International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013; pp. 13–19.
23. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Not.* **2014**, *49*, 269–284.
24. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1097–1105.
25. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* **2018**, *275*, 1072–1086. [CrossRef]
26. Dominguez-Morales, J.P.; Jimenez-Fernandez, A.F.; Dominguez-Morales, M.J.; Jimenez-Moreno, G. Deep Neural Networks for the Recognition and Classification of Heart Murmurs Using Neuromorphic Auditory Sensors. *IEEE Trans. Biomed. Circuits Syst.* **2018**, *12*, 24–34. [CrossRef] [PubMed]
27. Xu, J.; Liu, Z.; Jiang, J.; Dou, Y.; Li, S. CaFPGA: An automatic generation model for CNN accelerator. *Microprocess. Microsyst.* **2018**, *60*, 196–206. [CrossRef]
28. Joachims, T. The Maximum-Margin Approach to Learning Text Classifiers: Methods Theory, and Algorithms. Ph.D. Thesis, University of Dortmund, Dortmund, Germany, 2000.
29. Kao, W.; Chung, K.; Sun, C.; Lin, C. Decomposition methods for linear support vector machines. *Neural Comput.* **2004**, *16*, 1689–1704. [CrossRef]
30. Antos, A.; Kégl, B.; Linder, T.; Lugosi, G. Data-dependent margin-based generalization bounds for classification. *J. Mach. Learn. Res.* **2002**, *3*, 73–98.
31. Guyon, I.; Li, J.; Mader, T.; Pletscher, P.A.; Schneider, G.; Uhr, M. Competitive baseline methods set new standards for the NIPS 2003 feature selection benchmark. *Pattern Recognit. Lett.* **2007**, *28*, 1438–1444. [CrossRef]
32. Alon, U.; Barkai, N.; Notterman, D.A.; Gish, K.; Ybarra, S.; Mack, D.; Levine, A.J. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. Natl. Acad. Sci. USA* **1999**, *96*, 6745–6750. [CrossRef]
33. Lewis, D.D.; Yang, Y.; Rose, T.G.; Li, F. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.* **2004**, *5*, 361–397.
34. Keerthi, S.S.; DeCoste, D. A modified finite Newton method for fast solution of large scale linear SVMs. *J. Mach. Learn. Res.* **2005**, *6*, 341–361.

35. Kyrkou, C.; Theocharides, T. A Parallel Hardware Architecture for Real-Time Object Detection with Support Vector Machines. *IEEE Trans. Comput.* **2012**, *61*, 831–842. [CrossRef]
36. Bauer, S.; Köhler, S.; Doll, K.; Brunsmann, U. FPGA-GPU architecture for kernel SVM pedestrian detection. In Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops, San Francisco, CA, USA, 13–18 June 2010; pp. 61–68.