A new hardware architecture of high-performance real-time texture classification system based on FPGA

Yanjun Zhang[1], Xin Guo[1], Hongchen Guo[1*], Yichen Zhang[2]

[1]School of Cyberspace Science and Technology, Beijing Institute of Technology, No. 5, South Street, Zhongguancun,Haidian District, Beijing, 100081, Beijing, China.

[2]School of Beijing, Beijing Institute of Technology, No. 5, South Street, Zhongguancun,Haidian District, Beijing, 100081, Beijing, China.

*Corresponding author(s). E-mail(s): guohongchen@bit.edu.cn;
Contributing authors: zhangyj@bit.edu.cn; 3120231278@bit.edu.cn;
1320221099@bit.edu.cn;

Abstract

The visual system is crucial as an important source for intelligent robots to obtain external information. However, the real-time performance of existing studies is still insufficient, so a new high-performance target classification system based on FPGA is designed as a part of the visual system. This system optimizes the hardware structure of the target classification algorithm and a new method focused on improving the parallelism is proposed to improve the real-time performance of this system.This system is implemented on the Xilinx Zynq-7045 FPGA. And the experimental results show that for the grayscale image with 128*128 resolution, the feature extraction time of the system is only 85.64 µs, whose speed is 3 orders of magnitude higher than the MATLAB platform. Although all data are truncted to minimal fixed-point width rather than floating-point to reduce the hardware consumption in this architecture,the classification pressision is similar with that in software with float-point data. Also,the resource consumption of this design is lower compared to the other existing hardware structure.

Keywords: Median Robust Extended Local Binary Pattern, incremental SVM, FPGA ,real-time texture classification system

1 Introduction

As an important source for intelligent robots to obtain external information, the visual system is very crucial. With the development of machine learning, the target classification system based on machine learning is applied in intelligent robots as a part of the visual system, which classifies the objects in images collected by the image acquisition system to guide the actions of the intelligent robot. Texture is a fundamental characteristic of the appearance of virtually all natural surfaces and is ubiquitous in natural images[1]. Because the texture of different object surfaces varies, texture can be used as a classification feature to classify the objects in images.

The texture classification system is usually divided into three functional modules: the texture feature extraction module, the training module, and the classification module. For texture feature extraction, Local Binary Pattern (LBP) [2] algorithm is widely used due to its invariance to monotonic illumination changes and low computational complexity. Numerous local binary pattern variation algorithms have emerged to improve the robustness and feature extraction ability of the LBP algorithm, such as complete local binary pattern[3], rotation invariant robust binary texture classification[1], scale variable local binary pattern[4], and Median Robust Extended Local Binary Pattern (MRELBP)[5]. Compared with the other variants, the MRELBP algorithm based on combining a median filter with multiresolution support, overcomes the disadvantages of sensitivity to image blur and noise, failing to capture texture macrostructure, and high feature dimensionality[5]. For training and classification algorithms, traditional batch training algorithms need to train all the data when new samples appear. Therefore, incremental training algorithms emerge,such as incremental training of random forests[6], semi supervised incremental support vector machines[7], incremental fuzzy classifiers[8] and incremental SVM[9], which can achieve training with additional samples and make the overall training effects similar to that of batch training algorithms.

With the development of FPGA technology, many researchers also try to implement texture classification system with FPGA, which can improve the real-time performance of the system by using the parallel characteristics of FPGA. Christos Kyrkou uses local binary mode for texture extraction, optimizes the hardware structure of cascaded support vector machines, and achieves a real-time processing rate of 40 frames per second for face detection[10]. Zhang proposes a hardware architecture for the facial recognition algorithm based on local binary patterns. The hardware resources of the entire system are 5975 LUTs, the clock frequency is 233 MHz, and the recognition speed is 74 times faster than that of software implementation[11]. However, there are relatively few cases of using MRELBP or incremental training methods to achieve texture classification on FPGA.

In this paper, a texture classification system based on FPGA using MRELBP algorithm and incremental SVM algorithm is designed, which optimizes the algorithm process to adapt to the hardware acceleration structure, completes the pipeline processing of each module, designs and optimizes the hardware acceleration structure by using Verilog. Compared with the current software implementation schemes, the hardware architecture of this system has stronger real-time performance, more practicality and effectiveness.

The remainder of this paper is organized as follows. Texture extraction algorithm and hardware architecture implementation are given in Section Two. Section Three presents the Incremental SVM algorithm and hardware architecture implementation in detail. Comprehensive experimental results and analysis of MRELBP module and Incremental SVM module are given in Section Four. Finally, Section Five concludes this paper.

2 Texture extraction algorithm and hardware architecture implementation

2.1 MRELBP texture extraction algorithm

In order to simultaneously obtain micro and macro texture information of images, and improve the performance of texture feature extraction and noise robustness, Liu proposes Robust Extended Local Binary Pattern (RELBP) based on Extended Local Binary Pattern (ELBP)[4]. RELBP descriptor improves the filter response Φ of a single pixel in ELBP. Using the joint histogram of center pixel descriptor RELBP_CI, neighborhood pixel descriptor RELBP_NI and radial pixel descriptor RELBP_RD to represents the texture features of the image, collectively referred to as multi-scale RELBP(MRELBP)[5].

Liu compares the effects of filter types and parameters on texture extraction performance, and tests the performance of various filters such as the Gaussian filter, average filter, and median filter in the RELBP algorithm. After testing and verifying a large number of parameters, it is found that median filtering can obtain the parameters with the best performance, as shown in Table 1.

Table 1 Table of optimal parameters for RELBP[5].

Parameter	Parameter value
Filter type	MRELBP
Radius combination	(2, 4, 6, 8)
Number of samples per radius	8
Sliding window size	3×3, 5×5, 7×7, 9×9

Based on this, the MRELBP algorithm based on median filtering is used to complete the hardware structure designing and optimization as the core algorithm of the texture extraction module in this paper.

2.2 Overall architecture of MRELBP module

The MRELBP algorithm process is shown in Figure 1, which includes 5 parts: median extraction, MRELBP_CI descriptor extraction, interpolation, MRELBP_NI & MRELBP_RD descriptor extraction and joint histogram output.

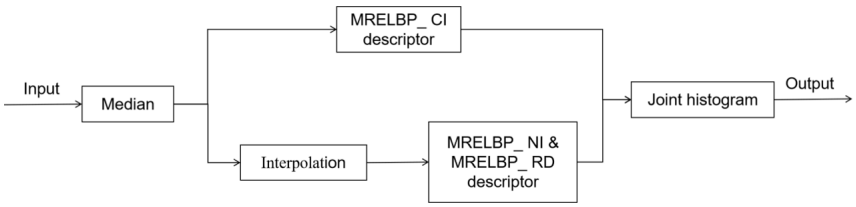

Fig. 1 MRELBP algorithm flowchart

This system first performs median extraction after inputting the image, then concurrently conducts MRELBP_CI descriptor extraction and interpolation, MRELBP_NI, and MRELBP_RD descriptor extraction, and finally outputs a joint histogram.

In order to improve the real-time performance of this system, the MRELBP module is implemented in a pipeline manner. The overall architecture of the MRELBP module is shown in Figure 2, including: the median preparation module, the median filtering module, the MRELBP_CI descriptor extraction module, the interpolation module, the MRELBP_NI & MRELBP_RD descriptor extraction module and the joint histogram output module.

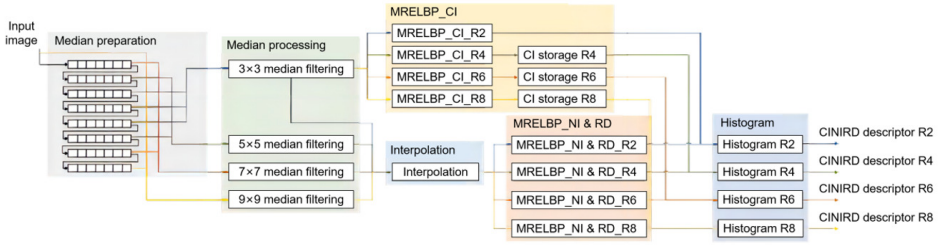

Fig. 2 Overall Hardware Architecture of MRELBP Module

The workflow of the circuit is as follows:

Step 1: Input the image into 8 shift registers constructed by block memory (BRAM) for caching, in order to align the signals of the 3×3, 5×5, 7×7 and 9×9 median filters.

Step 2: Input the cached results into a parallel median filter for median filtering.

Step 3: Input the 3×3 median filtering results into MRELBP_CI descriptor extraction module for descriptor extracting. In order to align sampling results with different radii, three shift register buffer areas are designed in this step.

Step 4: Input all four sets of median filter results into the interpolation module for interpolation. To meet real-time requirements, four sets of interpolation calculations are performed simultaneously, all implemented using a pipeline structure.

Step 5: Input the interpolated results into MRELBP_NI & MRELBP_RD descriptor extraction module for extracting neighborhood descriptor and radial descriptor.

Step 6: Input MRELBP_CI, MRELBP_NI & MRELBP_RD descriptors into the joint histogram output module for generating histogram output.

In this design, the hardware structure of the MRELBP module is implemented in a fully pipelined manner, which can achieve parallel processing of any radius to improve real-time performance. The Median preparation module is mainly implemented by the shift register cache. The following focuses on the implementation of the median module, the improvement and implementation of the MRELBP_CI descriptor, the interpolation module, and corresponding data storage structures, and the implementation of the MRELBP_NI & MRELBP_RD descriptor extraction module.

2.3 Implementation of median module

The median filtering sampling ranges corresponding to radius combinations (2, 4, 6, 8) are 3×3, 5×5, 7×7 and 9×9. Among them, the 5x5 median filter, 7×7 median filter and 9×9 median filter's real-time processes are relatively difficult. For a 5x5 median filter, Yang proposed a fast algorithm[12], as shown in Figure 3.

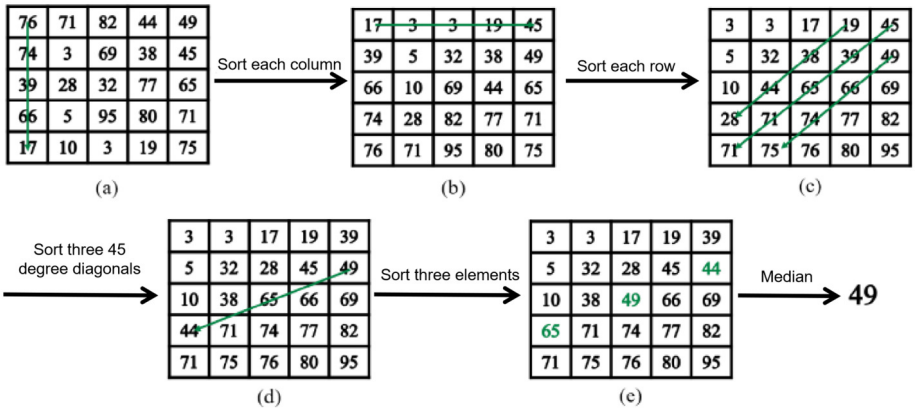

Fig. 3 Fast algorithm for 5×5 median filtering

The workflow of the fast algorithm proposed by the Yang is as follows: Step 1: As shown in Figure 3 (a), sort each column of pixels in the 5×5 filtering window in ascending order to obtain Figure 3 (b).

Step 2: Sort each row of pixels in ascending order to obtain Figure 3 (c).

Step 3: Sort the three sets of elements in the 45 degree diagonal direction in ascending order according to the arrow direction to obtain Figure 3 (d).

Step 4: Sort the three elements shown in Figure 3 (d) in ascending order. The result of the 5×5 median filter is the center point 49 in Figure 3 (e).

For 7×7 and 9×9 median filters, this paper proposes a new dimensionality reduction method. Taking the 7×7 median filter as an example, the implementation process of the median filter is shown in Figure 4.

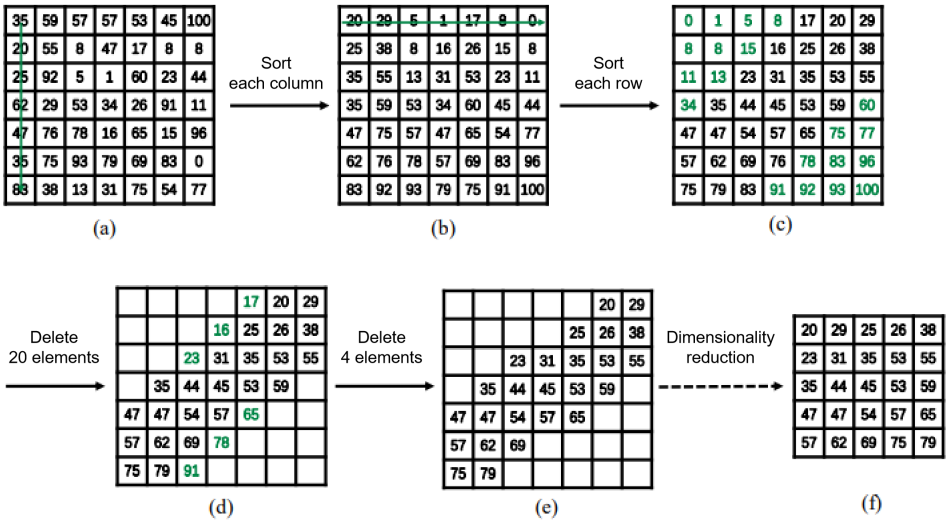

Fig. 4 Dimensionality reduction algorithm for 7×7 median filtering

The specific steps are as follows:

Step 1: Sort each column of pixels in the 7×7 filtering window in ascending order, as shown in Figure 4 (a) to obtain Figure 4 (b).

Step 2: On the basis of Figure 4 (b), sort each row of pixels in ascending order to obtain Figure 4 (c).

Step 3: Reduce the dimension of the 7×7 median filter to 5×5 median filter by selectively removing the sorted data in Figure 4 (c). The details include deleting the 10 data in the upper left corner and the lower right corner of Figure 4 (c) to obtain Figure 4 (d). After the sorting operation, these 20 data will not necessarily result from the median filter of 7×7.

Step 4: Sort the three elements in the upper left corner and the lower right corner of Figure 4 (d), and delete the two smaller and two larger elements to obtain Figure 4 (e).

Step 5: The remaining 25 data in Figure 4 (e) are directly output to the 5×5 median filter, repeating the operation in Figure 3.

The process of the 9×9 median filter is similar to that of the 7×7 median filter, in which the dimension of the 9×9 median filter is reduced to the dimension of the 7×7

median filter, Then the dimension of the 7×7 median filter is reduced to the dimension of the 5×5 median filter. The proposed dimensionality reduction and median methods can be implemented in a pipeline structure, improving the real-time performance.

2.4 Improvement and implementation of MRELBP_CI descriptor

MRELBP_NI & MRELBP_RD descriptor in the original MRELBP algorithm is sampled within a range of $(2 * radius + 1)^2$ pixels centered on pixel X_c. But,the MRELBP_CI descriptor proposed needs to calculate the average value of the entire image [4], and then perform the secondary traversal of each data to calculate the MRELBP_CI descriptor. Therefore, the data of the entire image must be stored. However, storing the entire image reduces real-time performance. Since the value ranges of the three descriptors are different, the value range of the original MRELBP_CI descriptors is modified to match the ranges of the MRELBP_NI & MRELBP_RD descriptor in order to improve the real-time performance without influencing the feature exaction performance.

In order to compare the performance of the improved MRELBP_CI descriptor with the original algorithm, this paper conducted experiments on a universal benchmark data set Outex-TC10 and Outex-TC12 on the MATLAB platform. In the experiment, the texture extraction module is completed using the source code provided by Liu[5], while the training and classification modules are carried out using the SVM universal toolkit LIBSVM[13]. This paper uses linear kernel functions and maintains the default parameters of the toolkit unchanged. In order to explore the impact of accuracy on MRELBP_CI, this paper rounds the average value in MRELBP_CI. The experimental results are shown in Table 2.

From the results in Table 2, it can be seen that the improved MRELBP_CI descriptor has slightly better classification performance in the range of $(2 * radius + 1)^2$ than the method of taking the average value in the entire image. At the same time, it can be seen from the table that the MRELBP_CI descriptor has a low requirement for accuracy. Therefore, in the hardware implementation process, sixteen-bit fixed point decimals are used as the bit width of the intermediate parameter.

2.5 Interpolation and MRELBP_NI & MRELBP_RD descriptor extraction

After passing through the median filter, the MRELBP_NI & MRELBP_RD descriptor should be sampled at 8 points within the radius range (2, 4, 6, 8). The 4 sampling points that don't completely fall at the center of the pixel within the sampling range need to be obtained through bilinear interpolation from the surrounding 4 points. Sampling range $(2*radius+1)^2$ involves data from $2*radius+1$rows, so it is necessary to consider data caching issues.

This paper caches the 3×3, 5×5, 7×7 and 9×9 median filter results in 4rows, 8rows, 12rows and 16 rows respectively and directly interpolates all points within the $(2 * radius+1)^2$ range when the data in $2*radius+1$ row arrives. Because the completion period of the 3×3, 5×5, 7×7, 9×9 median filters are different, the interpolation results

Table 2 Classification accuracy of MRELBP_CI within different radius and value ranges (%).

Data set	Radius (Pixels)	Original MRELBP_CI[5]		MRELBP_CI proposed in this paper		Rounding of MRELBP_CI proposed in this paper	
		Accuracy	Average	Accuracy	Average	Accuracy	Average
Outex_TC10	R2	100. 000	99. 648	100. 000	**99. 694**	100. 000	**99. 694**
	R4	98. 906		99. 141		99. 089	
	R6	99. 974		99. 922		99. 948	
	R8	99. 714		99. 714		99. 740	
Outex_TC12_000	R2	99. 896	99. 818	100. 000	99. 870	100. 000	**99. 922**
	R4	99. 688		99. 792		99. 896	
	R6	99. 792		99. 688		99. 896	
	R8	99. 896		100. 000		99. 896	
Outex_TC12_002	R2	95. 625	95. 469	95. 833	95. 573	95. 833	**95. 677**
	R4	95. 208		95. 417		95. 625	
	R6	95. 417		95. 208		95. 625	
	R8	95. 625		95. 833		95. 625	

of the inner circle need to be cached for 5 rows to align with the outer radius. After all the median results within the $(2 * radius + 1)^2$ range are calculated, extract all the median results required for calculating four interpolation sampling points and input them into four parallel interpolation calculation units. After bilinear interpolation, four interpolation sampling points with a 24-bit bit width are obtained. At the same time, another 4 sampling points with 8bits bit width are extracted from BRAM and entered into MRELBP_NI & MRELBP_RD descriptor extraction module to calculate MRELBP_NI & MRELBP_RD descriptors. This system uses BRAM to store the joint histogram during the MRELBP_NI & MRELBP_RD descriptors extraction process and update the histogram values using the NI and RD output from the interpolation module as addresses. Finally, four circumferences with 800-dimensional histograms are output as extracted texture features.

3 Incremental SVM algorithm and hardware architecture implementation

3.1 Incremental SVM algorithm

The input of the incremental SVM algorithm is a linear separable training set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ with data $x_i \in R^n$ and label $y_i \in \{-1, 1\}$. All samples are divided into Unlearned sets, Margin sets, Error sets, and Reserve sets. The incremental training process is the process of allocating an Unlearned set to Margin set, Error set, and Reserve set, causing transitions between sets. The difficulty and focus of incremental SVM in hardware implementation mainly lie in solving the coefficient sensitivity beta, edge sensitivity gamma, and inverse matrix R_s.

3.2 Overall architecture of incremental SVM module

The Incremental SVM module is mainly divided into 10 modules, as shown in Figure 5.

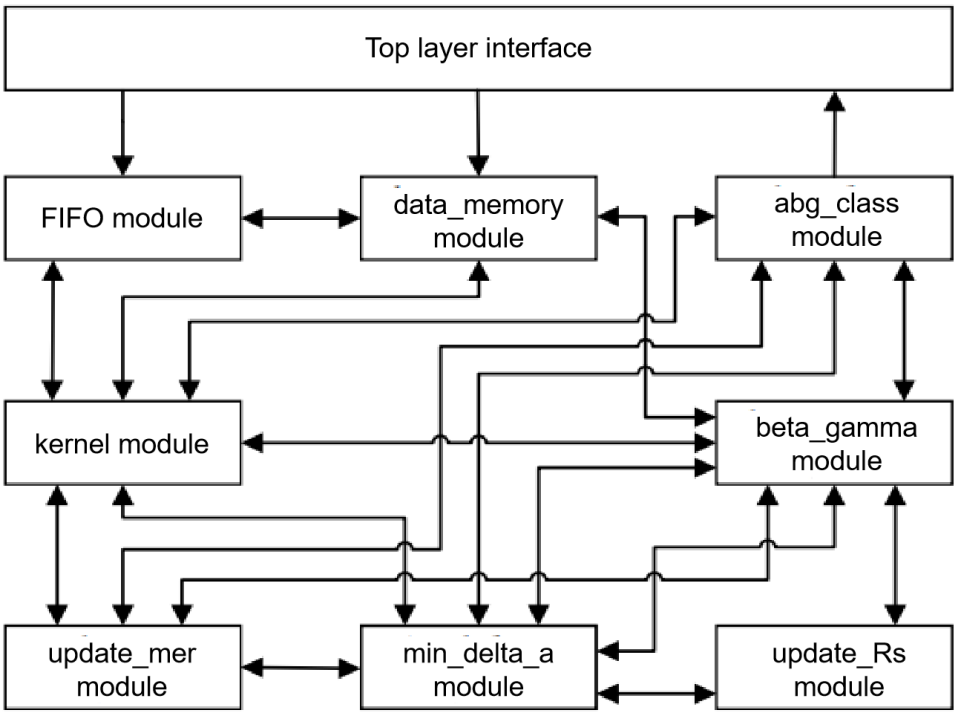

Fig. 5 Overall Architecture of Incremental SVM Module

The FIFO module is responsible for data synchronization between data_clk and svm_clk. The data_memory module is responsible for storing the feature data extracted by the MRELBP module. Although incremental training can achieve training on a single data, it still must store all the original data for interaction in the early stage. The kernel module is responsible for calculating the dot product between data in order to reduce storage resource consumption, store and control the dot product results in the form of an upper triangular matrix. The abg_class module is responsible for calculating the parameter alpha, hyperplane equation intercept b, and gradient of the KKT condition for incremental SVM in training mode, as well as calculating the classification result class_result and class_vld in classification mode. The beta_gamma module is responsible for calculating coefficient sensitivity beta and edge sensitivity gamma. The update_Rs module is responsible for calculating, updating, and storing the inverse matrix Rs. The triangle_RD module is responsible for parallel reading of upper triangular matrix in kernel module and the lower triangular matrix in the update_Rs module. The min_delta_a module is responsible for calculating the incremental parameter Δ_{pmin} for six categories of changes. The update_mer module is responsible for storing and updating the data numbers in the Margin set, Error set and Reserve set.

Among them, the beta_gamma module and the update_Rs module are the main difficulty and focus in the hardware implementation process, which will be emphasized in the following text.

3.3 Beta_gamma module

Beta_gamma module is responsible for calculating coefficient sensitivity beta and edge sensitivity gamma, and its structure is shown in Figure 6.

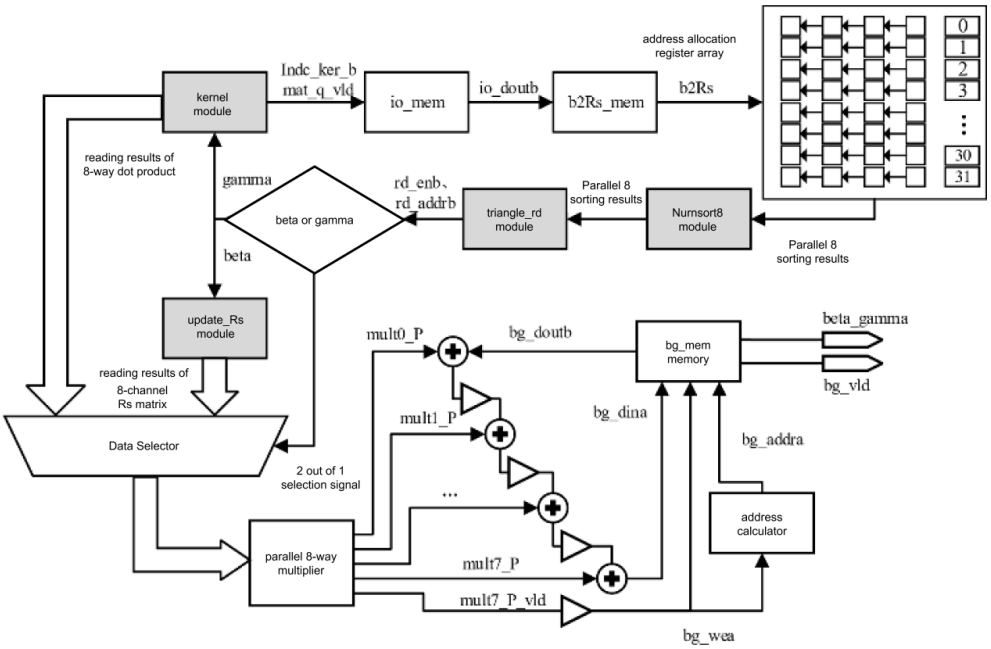

Fig. 6 Beta_gamma module structure diagram

When train_en is enabled, the beta_gamma module is responsible for calculating the coefficient sensitivity beta and edge sensitivity gamma. Because calculating gamma requires beta participation, beta's calculation priority is higher than gamma's. Because both beta and gamma vectors have an element called label, the memory a depth of 514 and the bit width of 53 bits in the beta_gamma module is set. Dynamically allocate the storage positions of beta and gamma in the memory based on margin_num, error_num, and reserve_num, and store them in the order of label(label_k_a), Margin set(beta), Error set(gamma), reserve set(gamma) and inner product (gamma) of the current data. Label signal indc_ker_a outputted by kernel module is passed to indc_bg to record the number of data currently participating in training in beta_gamma module. Signal indc_ker_b is passed to indc_bg_mem to represent the data number corresponding to the current calculated beta and gamma. To achieve beta and gamma calculations with the parallelism of 8, two sets of address mapping units have been set up in this

module. Io_mem is responsible for mapping the relationship between data numbers data_cnt and bg_mem, and storing position bg_cnt of beta and gamma in bg_mem by using data_cnt as the address. Oi_mem is responsible for mapping the relationship between data numbers bg_mem and data_cnt, and storing data_cnt using position of beta and gamma in bg_mem as the address. By reading io_mem and oi_mem, two types of information can be obtained: data number reading for bg_mem and data number corresponding to elements in bg_mem.

3.4 Update_Rs module

Update_Rs module is responsible for calculating, updating, and storing the inverse matrix Rs, as shown in Figure 7.

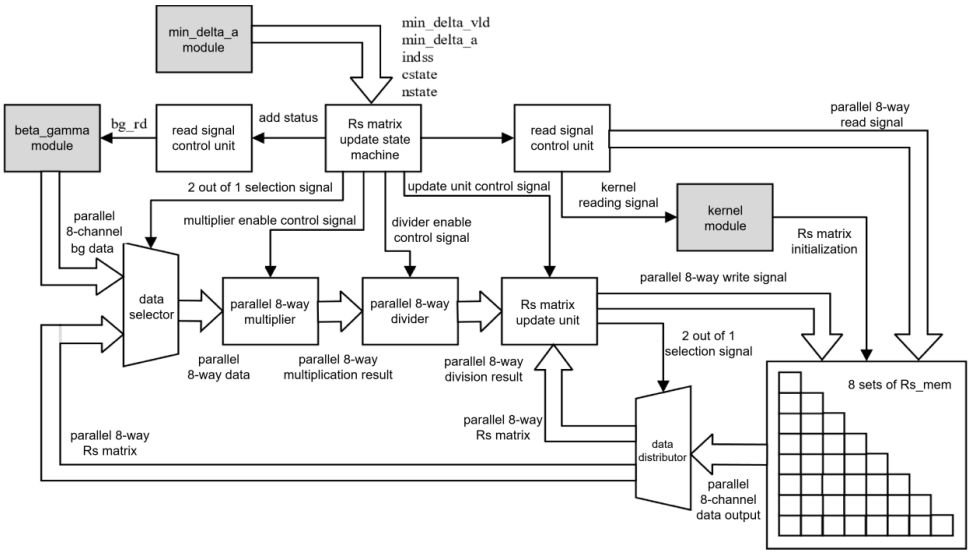

Fig. 7 Update_Rs module structure diagram

When train_en is enabled, the update_Rs module is responsible for updating Rs matrix. Rs matrix is a symmetric matrix, which is mainly related to the Margin set, and its dimension is exponentially related to the number of Margin sets. To ensure accuracy, the bit width during hardware acceleration is implemented in 53bits, so the storage capacity of the Rs matrix is $53bit(margin\_num + 1)(margin\_num + 1)$. In reality, the size of the Margin set is much smaller than the number of samples. Therefore, in the hardware implementation process, the maximum number of Margin sets is tentatively set to 1/4 of the sample data, which means that the Rs matrix requires up to 53bit×129×129 storage. In order to reduce the consumption of storage resources, this system adopts a lower triangular matrix storage method, and achieves parallel reading through read control. The storage amount is reduced by approximately half, occupying approximately 12.5 blocks of 36k BRAM.

Because samples may be migrating from Margin set to other sets during the training process, resulting in invalid rows and columns of the corresponding Rs matrix, subsequent sample data migrating into the Margin set needs to cover invalid storage data in Rs_mem. Therefore, the indicator signal empty is set to indicate whether the starting address of each column in Rs matrix is valid. Empty_mem is responsible for storing all empty signals. Addr_max signal records the maximum address stored in the current Rs_mem and addr_ready,which shows the minimum storable starting address in Rs_mem as the backup address for the next migration into the Margin set. After each update of Rs matrix, the address control signals mentioned above are updated correspondingly. If the address is added, set the corresponding empty in empty_mem to 1. Otherwise, set it to 0. Update addr_max when Rs_mem writes the latest data. After all updated, start traversing empty_mem to find the minimum available address as addr_ready. Due to the randomness of training, the storage addresses of each parameter do not correspond one-to-one in order. Therefore, b2Rs, Rs2b, and Rs2ker are used as address mappings to ensure correspondence between the Rs matrix and other parameters. The address mapping is updated after each update of the Rs matrix.

4 Results and Analysis

This system is implemented in XilinxZynq-7045 FPGA, with a chip speed level 2.

Because the texture extraction, incremental SVM training and incremental SVM classification functions are relatively independent, this chapter conducts detailed simulation and analysis of the MRELBP module and incremental SVM module respectively.

4.1 MRELBP module simulation results

The MRELBP module of this system supports texture extraction of up to 512×512 pixel images, with a clock frequency of up to 200MHz. The resource consumption on Zynq-7045 FPGA is shown in Table 3, with a total of 49033 LUTs and 41.5 BRAMs used.

Table 3 Resource consumption of MRELBP module on Zynq-7045 FPGA

Resource	Consume	Available resources	Availability
LUT	49033	218600	22%
FF	19969	437200	5%
BRAMs	41.5	545	8%

Due to the fixed number of storing data rows during the implementation of MRELBP module, there is a linear relationship between storage resources and image width. The storage resources used by the MRELBP module are only 8% of the available resources of Zynq-7045, so there is still a lot of room for expansion. This system implements a hardware architecture for pipeline MRELBP texture extraction, with

128*128 resolution input image. The extraction time from the input of the first pixel to the output of the last element of the histogram is only 85.64 μs, as shown in Table 4.

Table 4 Comparison of MRELBP texture extraction time and previous work in this system

Comparison parameters	Extraction time(ms)	Increase multiple
Extraction time of this architecture at a clock frequency of 200MHz	0.08564	/
Extraction time of MRELBP on MATLAB [8]	416.6	4864.55
Extraction time of raw LBP on FPGA achieved by Stekas N [14]	2.602	30.38

From Table 4, it can be seen that the proposed architecture accelerates by more than 4800 times compared to software algorithms. Compared with the LBP implemented by StekasN on FPGA [14], although the MRELBP algorithm is more complex than LBP, its speed has increased by more than 30 times.

4.2 Incremental SVM Module Simulation Results

Because the distributions of Margin set, Error set, and Reserve set in incremental SVM module are completely random, the running time is related to the statistical distribution of the samples, while the training time is related to the number of data that has been trained. Therefore, the reference value for a single sample's training time is not significant, and the training time for different training sets and labels is also different. Therefore, this paper estimates the maximum training time based on the hardware structure and simulation results. Assuming that the number of data has been trained is 511, the estimated training time for the 512th data based on the hardware circuit design structure is shown in Table 5. According to the actual situation, the number of support vectors is much smaller than the number of data in other categories. Assuming that the number of support vectors is 63, it is approximately 1/8 of the total data amount. And we assume that the total period number of each control signal and transition signal during the training process is set to N. Especially, N is approximately constant and its order of magnitude is much lower than the training period. After simulating the incremental SVM module on MATLAB, it is found that when the input is the feature vector extracted from MRELBP texture, the number of iterations for training has a small impact on accuracy due to significant features. Therefore, the system's iteration count is temporarily set to 5, and data that exceeds the number of iterations but does not converge is directly discarded.

As shown in Table 5, it takes 282146+2N cycles to complete the training, which is 2.23ms at the clock frequency of 125MHz. Finally, assuming that the data converges or is discarded after 5 iterations, it takes approximately 802282+10N cycles to complete, which is 6.42ms at a clock frequency of 125MHz.

Table 5 Clock period estimation for the 512nd data training process (N is approximately constant).

	Untrained convergence to R set	One training convergence	Iteration one convergence	Iteration 5 times
Calculation cycle of dot product	32×800	32×800	32×800	32×800
Calculation cycle of beta		8×64	2×8×64	6×8×64
Calculation cycle of gamma		56×2250	2×56×2250	6×56×2250
Update cycle of address mapping	512	512	2×512	10×512
Update cycle of alpha and gradient g	512	512	2×512	10×512
Update cycle of beta			8×64	5×8×64
Update cycle of gamma			450	5×450
Update cycle of Rs matrix			8×64	5×8×64
Other control cycles (constant)	N	N	2N	10N
Total cycle	26624+N	153136+N	282146+2N	802282+10N

The training and classification of incremental SVM modules are implemented on Zynq-7045 FPGA, and the resource consumption is shown in Table 6.

Table 6 Resource consumption of incremental SVM module on Zynq-7045 FPGA

Resource	Consume	Available resources	Availability
LUT	142604	218600	65%
FF	210394	437200	48%
BRAMs	362	545	66%
DSP	314	900	35%

Shao S utilizes fixed-point transformation of 50bit decimals to calculate intermediate parameters, and achieves parallel computation through block partitioning. However, the higher the parallelism, the more BRAM resources are consumed [15]. The method proposed in this paper avoids this deficiency by using S-type storage and parallel read control, achieving storage resources that are only related to the fixed-point accuracy of intermediate parameters. The size of training samples and Rs matrices are independent of parallelism.

The results of comparing the resource consumption of incremental SVM proposed in this paper with Shao S are shown in Table 7.

Table 7 Comparison results of resource utilization of incremental SVM modules

	Hardware architecture of Shao S	Hardware architecture proposed in this paper
Number of samples	420	512
Dimension of inverse matrix R	120	128
Total data (Kb)	252. 61	900
Parallelism	6	8
LUT consumption	253331	**142604**
FF consumption	423728	**210394**
BRAMS consumption	1892	**362**
DSP consumption	1916	**314**

Table 7 shows that the incremental SVM module can achieve better LUT, FT, BRAMS, and DSP resource consumption and higher parallelism compared to the Shao S hardware architecture with a small increase in samples. In this paper, resource consumption is lower when the data volume is larger, indicating that the proposed hardware architecture has higher computational efficiency.

5 Conclusion

In this paper, a hardware accelerated implementation of a texture classification system based on FPGA is completed. The texture classification system uses an improved

MRELBP texture extraction method to obtain texture features, which has the advantages of rotation invariance and noise robustness. The improved MRELBP hardware architecture proposed in this paper, uses fixed-point processing to replace double precision floating-point operations, inproving the real-time performance of texture extraction without reducing accuracy. According to the experimental results, the extraction speed of MRELBP texture extraction on hardware is three orders of magnitude faster than that on software, which can effectively meet the texture extraction requirements of large images. In addition, a parallel incremental SVM training and classification architecture is proposed, which utilizes triangular matrices to replace the storage of symmetric matrices in the original architecture. Conflict mechanisms are used to achieve parallel reading, parallel writing, and updating of triangular matrices, thereby reducing hardware consumption of storage resources. In summary, the texture classification system designed in this paper greatly accelerates processing speed and saves hardware consumption on FPGA while maintaining classification performance. It can be deployed on medium-sized FPGAs and has strong application value.

References

[1] Liu, L., Long, Y., Fieguth, P.W., Lao, S., Zhao, G.: Brint: binary rotation invariant and noise tolerant texture classification. IEEE transactions on Image Processing **23**(7), 3071–3084 (2014)

[2] Ojala, T., Pietikäinen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. Pattern recognition **29**(1), 51–59 (1996)

[3] Guo, Z., Zhang, L., Zhang, D.: A completed modeling of local binary pattern operator for texture classification. IEEE transactions on image processing **19**(6), 1657–1663 (2010)

[4] Guo, Z., Wang, X., Zhou, J., You, J.: Robust texture image representation by scale selective local binary patterns. IEEE Transactions on Image Processing **25**(2), 687–699 (2015)

[5] Liu, L., Lao, S., Fieguth, P.W., Guo, Y., Wang, X., Pietikäinen, M.: Median robust extended local binary pattern for texture classification. IEEE Transactions on Image Processing **25**(3), 1368–1381 (2016)

[6] Xie, T., Peng, Y., Wang, C.: hi-rf: incremental learning random forest for large-scale multi-class data classification. arXiv preprint arXiv:1608.08761 (2016)

[7] Wang, J., Yang, D., Jiang, W., Zhou, J.: Semisupervised incremental support vector machine learning based on neighborhood kernel estimation. IEEE Transactions on Systems, Man, and Cybernetics: Systems **47**(10), 2677–2687 (2017)

[8] Cheng, W.-Y., Juang, C.-F.: A fuzzy model with online incremental svm and

margin-selective gradient descent learning for classification problems. IEEE Transactions on Fuzzy systems **22**(2), 324–337 (2013)

[9] Diehl, C.P., Cauwenberghs, G.: Svm incremental learning, adaptation and optimization. In: Proceedings of the International Joint Conference on Neural Networks, 2003., vol. 4, pp. 2685–2690 (2003). IEEE

[10] Kyrkou, C., Bouganis, C.-S., Theocharides, T., Polycarpou, M.M.: Embedded hardware-efficient real-time classification with cascade support vector machines. IEEE transactions on neural networks and learning systems **27**(1), 99–112 (2015)

[11] Zhang, Y., Cao, W., Wang, L.: Implementation of high performance hardware architecture of face recognition algorithm based on local binary pattern on fpga. In: 2015 IEEE 11th International Conference on ASIC (ASICON), pp. 1–4 (2015). IEEE

[12] Yang, K., Wei, M., Sun, L.: Design of median filtering system based on fpga for large windows. In: 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), pp. 78–82 (2018). IEEE

[13] Chang, C.-C., Lin, C.-J.: Libsvm: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST) **2**(3), 1–27 (2011)

[14] Stekas, N., Van Den Heuvel, D.: Face recognition using local binary patterns histograms (lbph) on an fpga-based system on chip (soc). In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 300–304 (2016). IEEE

[15] Shao, S., Mencer, O., Luk, W.: Dataflow design for optimal incremental svm training. In: 2016 International Conference on Field-Programmable Technology (FPT), pp. 197–200 (2016). IEEE