Received November 30, 2019, accepted December 14, 2019, date of publication December 23, 2019, date of current version December 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2961334

A Novel Parallel Architecture for Template Matching based on Zero-Mean Normalized Cross-Correlation

XIAOTAO WANG[1], XINGBO WANG[2], AND LIANGLIANG HAN[3]
[1]College of Astronautics, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[2]College of Automation and the College of Artificial Intelligence, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
[3]Shanghai Institute of Aerospace System Engineering, Shanghai 201108, China

Corresponding author: Xingbo Wang (sinbowang@163.com)

This work was supported in part by Advance Research Project on Civil Aerospace Technology under Grant D030103, and in part by Advance Research Project on Manned Aerospace Technology under Grant 030601.

ABSTRACT Template matching based on zero-mean normalized cross-correlation measure (ZNCC) has been widely used in a broad range of image processing applications. To meet the requirements for high processing speed, small size, and variable image size in automatic target recognition systems, a novel field-programmable gate array (FPGA)-based parallel architecture is presented in this paper for the ZNCC computation. The proposed architecture employs two groups of RAM blocks, one of which is used for the multiply-accumulate operations of the real and the reference images and the other for data rearrangement of the reference image, and their functions are switched through 2-input multiplexers when searching at the next row. Moreover, the sum of the pixels in the searching area of the real image is computed through serially accumulating the differences between the new column in the current searching area and the old column in the last searching area using one dual-port RAM. Simultaneously, the sum of the squares of the pixels is calculated in the same way. Using the Altera Stratix II FPGA chip (EP2S90F780I4) as the target device, the compilation results with Quartus II show that compared with the traditional architecture, the synthesis logic utilization decreases from 63% to 35% and the usage of DSP blocks decreases from 59% to 39%, while the memory bits only increase by 8% and the usage of other resources is nearly the same. The simulation and practical experimental results show that the proposed architecture can effectively improve the performance of the practical automatic target recognition system.

INDEX TERMS FPGA, normalized cross-correlation measure, parallel architecture, template matching.

I. INTRODUCTION

Template matching has been widely used in a broad range of applications related to computer vision and image processing, such as automatic target recognition, medical image fusion for diagnosis [1], satellite image monitoring, and binocular stereo vision [2], etc. The basic algorithms of template matching are tasked to find the possible location of a template image in a real image through calculating the similarity measure between the template and the searching area within that real image. The typical similarity measures adopted in template matching algorithms include but are not limited to, nonnormalized cross-correlation, normalized cross-correlation (NCC), zero-mean normalized cross-correlation (ZNCC), sum of absolute differences (SAD), sum of squared differences (SSD), and so on. Due to their invariance to brightness and/or contrast variations, NCC and ZNCC are by far the most popular similarity measures used in template matching [3].

To obtain the precise location of a template in a real image, it is required to compare portions of the two images in a large number of relative positions. Therefore, the computational burden required for template matching may be unaffordable for many embedded applications, such as automatic target recognition and tracking, with the requirements for real-time processing, small size, and low-power consumption [4]. Several techniques have been developed to speed up the computation of the basic searching procedure [5].

The associate editor coordinating the review of this manuscript and approving it for publication was Yizhang Jiang.

However, these techniques can be trapped into a local extreme which may lead to a wrong localization of the target [3].

In the applications with a high window overlap between frames, such as motion estimation [1], feature detection [6], defect detection [7], etc., an efficient method has been proposed in [6], [7] to calculate the ZNCC measure, which uses precalculated sum tables to compute the terms in the denominator of the ZNCC measure and uses the fast Fourier transform (FFT) to compute the numerator (the standard cross-correlation (CC)) in the spectral domain. The proposed approach has been further improved in [1] through using precalculated sum tables to calculate both the numerator and the denominator of the ZNCC measure to eliminate the redundancy of the repeated ZNCC calculations between different frames. However, this method is not appropriate for real-time images without a window overlap, such as those employed in automatic target recognition. In addition, although the FFT in the frequency domain can be used to calculate the standard CC, it will dramatically increase the computational cost as the size of the template image increases [7].

One better choice of meeting the real-time requirement is to implement the computation of the ZNCC measure with an application-specific integrated circuit (ASIC). In [8], an efficient VLSI architecture has been proposed to accelerate the ZNCC computation for image registration. The architecture is very suitable for small templates since it needs M^2 window processors for a reference block of size $M \times M$ pixels. For a relatively larger template, cascaded chip configuration can be used to speed up the process of locating the best match; however, it will consume a large amount of resources and complicate the logic architecture and its workflow. Furthermore, the ASIC-based implementation can be expensive and is inflexible to support a variety of matching algorithms for different stages of image processing tasks.

In fact, the computation time of the ZNCC measure can be reduced by exploiting the intrinsic parallelism in the matching processing procedure. In [4], [9], several approaches have been proposed to utilize the parallelism to accelerate the template matching process for image correlation in a multiprocessor system. Moreover, a graphics processing unit (GPU), a processor customized primarily for graphics processing, also uses the intrinsic parallelism for such purposes [10]. However, these approaches cannot meet the requirements for small size and low-power consumption, especially for many embedded applications such as an automatic target recognition system.

An alternative approach is to use a field programmable gate array (FPGA) for the parallel implementation of the ZNCC computation. On one hand, FPGAs can greatly accelerate image processing with parallel operations; on the other hand, FPGAs contain many multiply-accumulate (MAC) operations available for the ZNCC computation. In [11], several efficient architectures based on FPGAs have been proposed for the implementation of the ZNCC computation. Although the computation can be achieved during data reading, the spatial architecture needs $m \times m$ MAC operations, and $(n\text{-}m) \times m \times p$ shift registers to implement the needed delay lines, where n and m refer to the sizes of the rows of the real-time and the reference images, respectively, and p is pixel bit-width. For a relatively large image, the above ZNCC computation needs more MAC operations than a general FPGA chip can afford. In [12], a real-time FPGA-based template matching module has been presented for visual inspection applications. The subsampling method is incorporated to reduce the number of pixels within the reference image and the searching block of the target image, which greatly improves the efficiency of the ZNCC computation but decreases the accuracy of the system. In the architectures proposed in [11], [12], the sizes of images cannot be varied according to the practical requirements due to the fixed number of delay lines and MAC operations. In [13], an efficient FPGA architecture has been proposed to compute multiple ZNCC-based template matchings. The architecture utilizes feedback FIFOs for the correlation window computations, which results in a large amount of resource consumption. In [14], an FPGA-based ZNCC architecture is proposed for robotic visual tracking. The architecture consumes a relatively small amount of hardware resources but has difficulty in meeting the requirements for variable parameters and high precision in automatic target recognition applications. In [15], two architectures based on FPGA have been proposed for the implementation of the ZNCC-based image matching with a variable image size. In the architectures, a large number of multiple-input multiplexers (MUXs) are used to select the RAM blocks for the reference image to correspond to those for the real-time image when searching at different rows, which greatly increases the usage of logic resources of the FPGA chips used.

To meet the requirements for high processing speed, small size, and variable image size in an automatic target recognition (ATR) system, a novel FPGA-based parallel architecture is proposed to further reduce the resource utilization and processing time of the ZNCC computation with a variable image size in this paper. In the proposed architecture, two groups of RAM blocks, each of which is allocated to buffer the whole reference image, are used for the MAC operations of the real and reference images and data rearrangement of the reference image. The data rearrangement is achieved through reading the reference image into one group of RAM blocks according to the sequence of the RAM blocks (of the real image) used for the MAC computation at the next row of the real image. When computing the ZNCC measure at the next row, the functions of the two groups of RAM blocks are swapped through using 2-input multiplexers. In addition, the sum of the real image within the region under examination is implemented through serially accumulating the results of subtracting the old column in the last searching area from the new column in the current searching area using one dual-port RAM buffering the corresponding rows of the real image. Simultaneously, the corresponding sum of squares of the pixels can also be implemented in the same way. Consequently, compared with the first architecture in [15], referred to as the traditional architecture in this paper, the logic gates and DSP blocks used in the proposed architecture are dramatically reduced.

Using the Altera Stratix II FPGA chip (EP2S90F780I4) [16] as the target device in an automatic target recognition system with a reference image with maximum size of 80 × 80 and a real-time image with maximum size of 512 × 512, the compilation results with Quartus II 8.0 [17] have shown that, compared with the traditional architecture, the usage of ALUTs (adaptive look-up tables) of the proposed architecture decreases from 46% to 19%, the usage of DSP blocks decreases from 59% to 39%, and the logic utilization decreases from 63% to 35%, while the memory bits only increase by 8% and the usage of other resources is nearly the same. The simulation and practical experimental results show that the proposed architecture can effectively improve the speed and localization precision of the target recognition system. Furthermore, the hardware implementation of the proposed architecture with a variable image size based on an FPGA can meet the space requirements for a smaller system size and has the flexibility to adapt to the changing image matching strategies.

The remainder of this paper is organized as follows. The basic principles of the proposed architecture are introduced in Section II. The simulation results as well as the experimental results are presented in Section III. Some practical issues are discussed in Section IV. Finally, conclusions are presented in Section V.

II. BASIC PRINCIPLES

A. ZERO-MEAN NORMALIZED CROSS-CORRELATION

The objective of template matching is to find the location of a reference image (template) within a larger real image, and the most popular method is to compute the ZNCC measure between the template and the portion of the real image under examination, as shown in Fig. 1. Let A and B be the real image of $K \times L$ pixels and the reference image of $M \times N$ pixels, respectively. Given a searching location (u,v), $(0 \le u \le K\text{-}M$, $0 \le v \le L\text{-}N)$, the zero-mean normalized cross-correlation measure is defined as,

$$C(u,v) = \frac{\sum\sum[A(i+u,j+v)-\overline{A(u,v)}]\times[B(i,j)-\overline{B}]}{\left\{\sum\sum[A(i+u,j+v)-\overline{A(u,v)}]^2\right\}^{1/2}\left\{\sum\sum[B(i,j)-\overline{B}]^2\right\}^{1/2}}, \tag{1}$$

where $\sum\sum$ denotes $\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}$. $\overline{B}$ is the mean value of the reference image (B), which is given as follows:

$$\overline{B} = \frac{1}{MN}\sum\sum B(i,j),$$

$\overline{A(u,v)}$ is the mean value of the real image (A) within the region under examination, which is given as follows:

$$\overline{A(u,v)} = \frac{1}{MN}\sum\sum A(i+u,j+v).$$

Let $Acc(u,v) = \sum\sum A(i+u,j+v)$, $A2cc(u,v) = \sum\sum A(i+u,j+v)^2$, $ABcc(u,v) = \sum\sum A(i+u,j+v)B(i,j)$, $Bcc = \sum\sum B(i,j)$, and $B2cc = \sum\sum(B(i,j))^2$. Then, equation (1) can be further rewritten as follows:

$$C(u,v) = \frac{[M\cdot N\cdot ABcc(u,v)-Acc(u,v)\cdot Bcc]}{\left\{MN\cdot A2cc(u,v)-(Acc(u,v))^2\right\}^{1/2}\left\{MN\cdot B2cc-(Bcc)^2\right\}^{1/2}}. \tag{2}$$

Then, the matching result is given by the location where $C(u,v)$ is maximized and exceeds a prescribed threshold. For convenience, these abbreviations are also available in the following descriptions, figures, and tables.

Compared with (1), equation (2) is simpler and easier to be implemented using parallel architectures. Furthermore, a more precise result of the ZNCC measure can be obtained by (2) through using fixed-point arithmetic with bit growth for the adder and multiplier, since there are no rounding errors introduced by the division operations in the computation of $\overline{B}$ and $\overline{A(u,v)}$. Therefore, equation (2) is more suitable for FPGA implementation, especially for the fixed-point implementation [15]. In addition, the $ABcc(u,v)$ term in the numerator of (2) also denotes the standard cross-correlation between the template and the portion of the real image under examination, i.e., the standard cross-correlation operation is contained in the ZNCC computation.

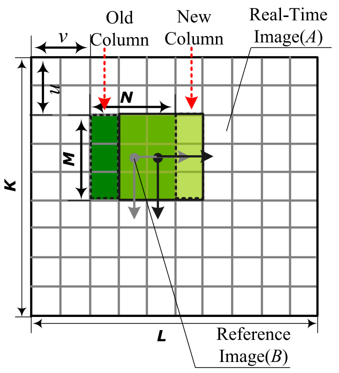

FIGURE 1. The image matching.

B. THE PROPOSED ARCHITECTURE

The above ZNCC computation requires computing all terms in the numerator and the denominator of (2), which will consume large computational time and logic resources. In this paper, a novel approach is proposed to reduce the utilization and processing time of the ZNCC-based template matching with a variable image size. Since the ZNCC computation is the principal operation of template matching, we mainly consider the implementation of the ZNCC computation using FPGAs and leave the subsequent searching for the maximum value of the ZNCC measures to be performed by an external microprocessor. The proposed architecture is illustrated in

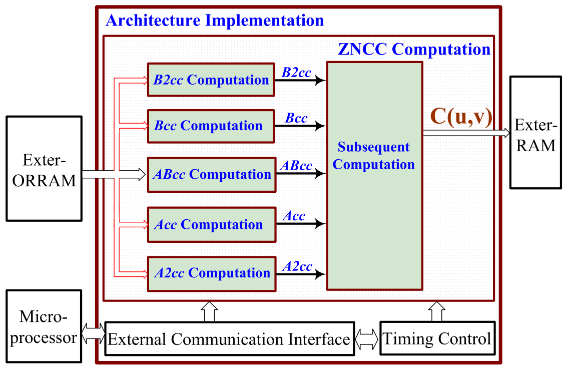

FIGURE 2. The block diagram of the proposed parallel architecture.

Fig. 2. For the sake of clarity, Fig. 2 and the following figures mainly illustrate the block diagram of the data flow for the ZNCC computation, and omit the addresses and control signals, such as write, read, etc. In this work, the numbers of rows and columns of the real and the reference image matrices, K, L, M, and N, are variable parameters input from the external microprocessor, with $K \leq Kmax$, $L \leq Lmax$, $M \leq Mmax$, and $N \leq Nmax$, where $Kmax$, $Lmax$, $Mmax$ and $Nmax$ are, respectively, the maxima of the numbers of rows and columns of the two images given by the system requirements.

In the proposed architecture, the Exter-ORRAM is an external RAM used for buffering the reference image and the real image. The module "Timing Control" mainly serves as a finite state machine to control the whole workflow of the ZNCC computation and to calculate the relevant parameters. The module "External Communication Interface" is used to perform communication with the external microprocessor for parameter input, command input, and status output. The module "ZNCC computation" is the main processing unit of the ZNCC computation, which is composed of several submodules, including *ABcc* computation, *Acc* computation, *A2cc* computation, *Bcc* and *B2cc* computation, and subsequent computation submodules. These submodules are implemented based on the following analyses.

1) ABCC COMPUTATION

According to (2), the *ABcc* computation involves calculating the products of any two corresponding pixels of the reference image and the portion of the real image under examination and then computing the sum of all the above products. To balance the computational time and the logic resource utilization, *Mmax* parallel MAC operations are needed to compute the*ABcc* in the column direction, corresponding to the maximum number of rows of the reference image matrix. For the practical input parameters, K, L, M, and N, it is necessary to disable the unused computation units. The M MAC operations can be accomplished in parallel in one clock period; therefore, the *ABcc* computation at a given searching position can be finished in N clock periods.

According to the system requirements for image sizes, *Mmax* RAM blocks of size $1 \times Nmax$ (ORAM[0], …, ORAM[*Mmax*-1]) are allocated for buffering the reference image, and *Mmax* RAM blocks of size $1 \times Lmax$ (RRAM[0], …, RRAM[*Mmax*-1]) for the real image. For the specific parameters, only M rows of both the reference and the real image matrices are used for the ZNCC computation at each searching location. When computing the *ABcc* at the next row of the real image, the pixels in the new row of the image must be input to one of the RAM blocks to replace its old content which will not be used anymore. Thus, the correspondences between the RAM blocks for the reference image and those for the real image are changed, as shown in Fig. 3. Fig. 3(a) illustrates the correspondences between the RAM blocks of the two images when computing the *ABcc* at the first row of the real image. When computing at the 2nd row, the $(M+1)$-th row of the real image is input to RRAM[0] to replace its old content (the first row of the real image that is not used anymore). In this case, RRAM[0] does not correspond to ORAM[0] but to ORAM[M-1], RRAM[1] corresponds to ORAM[0], and so on, as shown in Fig. 3(b).

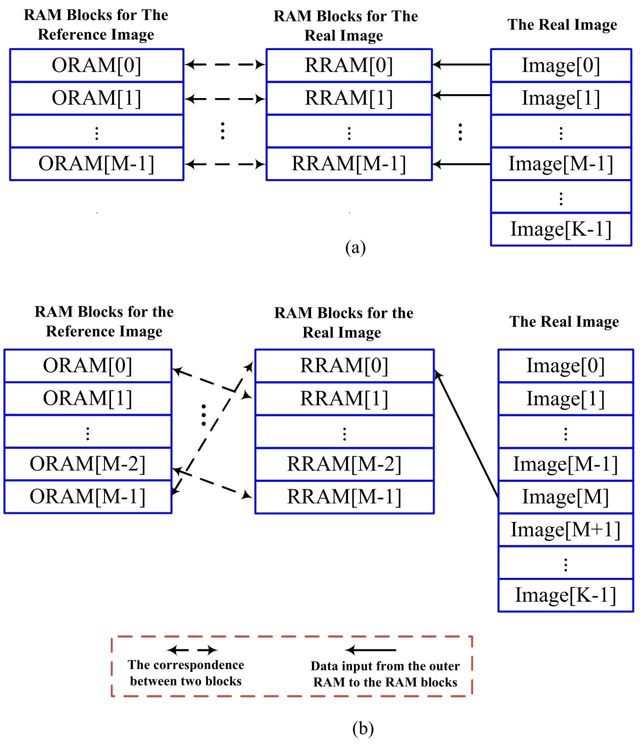
RAM Blocks for The Reference Image
RAM Blocks for The Real Image
The Real Image
ORAM[0]
ORAM[1]
ORAM[M-1]
RRAM[0]
RRAM[1]
RRAM[M-1]
Image[0]
Image[1]
Image[M-1]
Image[K-1]
(a)
RAM Blocks for the Reference Image
RAM Blocks for the Real Image
The Real Image
ORAM[M-2]
RRAM[M-2]
Image[M]
Image[M+1]
The correspondence between two blocks
Data input from the outer RAM to the RAM blocks
(b)

FIGURE 3. The correspondences between ORAM and RRAM when searching at the first row (a) and at the second row (b).

From the above analysis, it is necessary to rearrange the RAM blocks of the reference image or those of the real image, to make their data properly correspond while computing the *ABcc*. Since the capacity of each RAM block of the reference image is relatively small, it will be simple and easy to implement logic synthesis and routing for the data rearrangement of these RAM blocks. The traditional architecture proposed in [15] directly uses *Mmax* multiplexers with *Mmax* inputs to implement the data rearrangement of the RAM blocks for the reference image. For a relatively large reference image, it will cost a large amount of logic resources.

In the proposed architecture, an additional group of RAM blocks (ORAMB) for buffering the reference image is introduced to further reduce the logic utilization of the FPGA chip used. Therefore, there are two groups of RAM blocks (ORAMA and ORAMB) used for the *ABcc* computation and the data rearrangement, respectively. The functions of the two groups of RAM blocks are switched between the *ABcc* computation and the data rearrangement through *Mmax* 2-to-1 multiplexers. When computing the ZNCC measure at the first row of the real image, one group of RAM blocks (ORAMA) is used for the *ABcc* computation, while the other (ORAMB) is used for the data rearrangement of the reference image. When computing the ZNCC measure at the second row, the $(M + 1)$-th row of the real image will be input to replace the old content of RRAM[0]. Therefore, the data rearrangement is performed through reading the reference image again from the outer RAM into ORAMB according to the sequence of the updated RAM blocks of the real image (RRAM) used for the *ABcc* computation at the second row, i.e., the M-th row of the reference image is input to ORAMB[0], the first row to ORAMB[1], and so on, as shown in Fig. 4(a). When computing the ZNCC measure at the second row, the content of RRAM[0] is updated with the $(M + 1)$-th row of the real image. ORAMB is switched to be used for the *ABcc* computation and ORAMA to be used for the data rearrangement of the reference image through 2-to-1 multiplexers. During the *ABcc* computation at this searching row, the reference image is input to ORAMA according to the sequence of RRAM used for computing at the third row of the real image, as shown in Fig. 4 (b). In this way, the functions of the two groups of RAM blocks are switched back and forth between the *ABcc* computation and the data rearrangement through the 2-to-1 multiplexers when computing the ZNCC measure at different rows.

The block diagram of the *ABcc* computation is shown in Fig. 5, with the legend of the operations shown in Fig. 6, which are also available in the following figures. In the *ABcc* computation, one group of RAM blocks (ORAMA or ORAMB) for the reference image is selected for the *ABcc* computation by the 2-to-1 multiplexers under the control of the module "Timing Control". The data of the selected RAM blocks are simultaneously multiplied by the corresponding data of the RAM blocks (RRAM) for the real image. Then, a parallel adder "PAdd1" is used to sum the outputs of the multipliers for each column of the reference and the real images, $\sum_{i=0}^{M-1} A(i+u, j+v)B(i, j)$. Finally, the accumulator module "Accu3" is used to accumulate the outputs of "PAdd1", and thus the $ABcc(u,v)$ at a given searching position (u,v) is achieved.

In the traditional architecture, for each RAM block of the real image, there is one *Mmax*-input multiplexer used to select one of the *Mmax* RAM blocks of the reference image. Therefore, *Mmax* multiplexers with *Mmax* inputs are needed

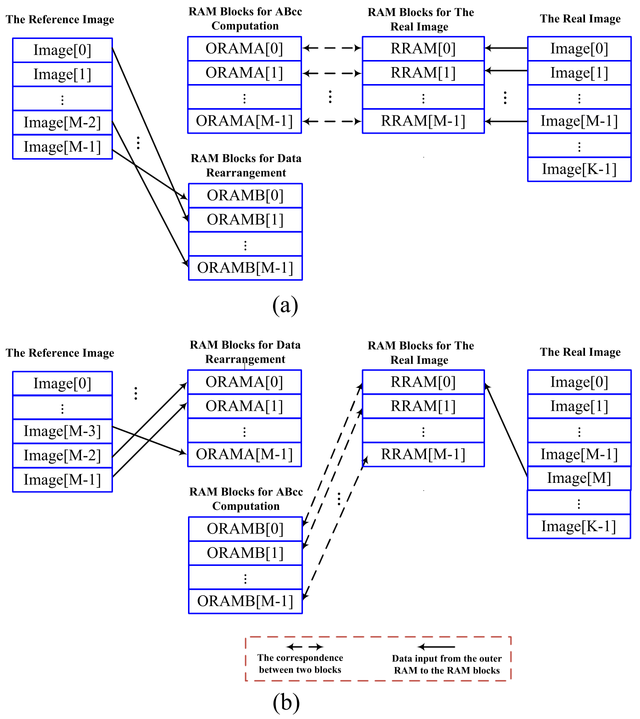

FIGURE 4. The correspondences between ORAMA or ORAMB and RRAM: (a) when searching at the first row, (b) when searching at the second row.

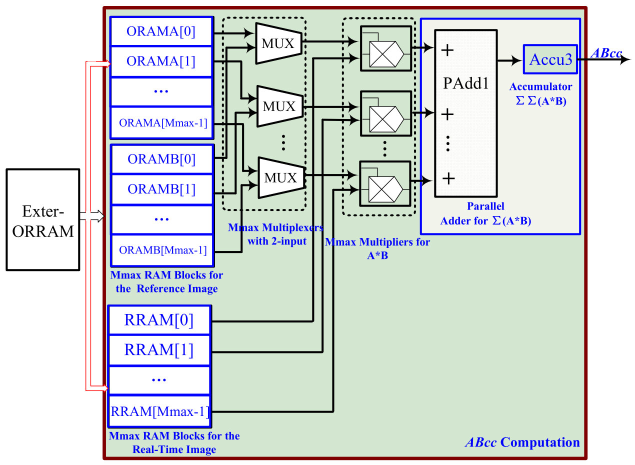

FIGURE 5. The block diagram of the *ABcc* computation.

to achieve the correspondences between the RAM blocks of the reference image and the real image. While in the proposed architecture, only one 2-to-1 multiplexer is required to select one of the two RAM blocks (in ORAMA and ORAMB, respectively) for each RAM block of the real image.

Although some logic resources are needed to control the extra group of RAM blocks in the proposed architecture, a 2-to-1 multiplexer will consume far less logic resources than an *Mmax*-to-1 multiplexer. Therefore, the total logic resources will be significantly decreased through the replacement of *Mmax Mmax*-input multiplexers with *Mmax* 2-to-1 multiplexers, as demonstrated in Section III.

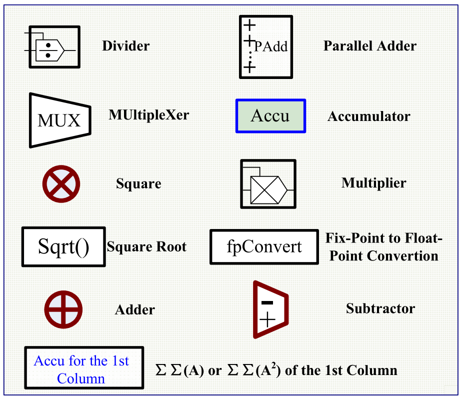

FIGURE 6. The legend for operations.

2) ACC AND *A2CC* COMPUTATION

Since the *Acc* and *A2cc* computations involve only the portion of the real image under examination, the overlapping portion of the real image at the adjacent searching positions can be taken advantage of to reduce the computational time and logic resources of the two modules.

At the first searching position of each row, $Acc(u,0)$ can be calculated as follows:

$$\begin{aligned} Acc(u,0) &= \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} A(i+u,j) \\ &= \sum_{i=0}^{M+u-1}\sum_{j=0}^{N-1} A(i,j) - \sum_{i=0}^{u-1}\sum_{j=0}^{N-1} A(i,j). \end{aligned}$$

$Acc(u,0)$ can be implemented using the module "Accu1 for the 1st Column", as shown in Fig. 7. The detail of the logic schematic of this module is shown in Fig. 8, where the accumulator "RowAccu1" accumulates the first N pixels of the u-th row of the real image (i.e., $\sum_{j=0}^{N-1} A(u,j)$), the accumulator "ColAccu1" directly accumulates the outputs of "RowAccu1" (i.e., $\sum_{i=0}^{M+u-1}\sum_{j=0}^{N-1} A(i,j)$), and each delay register (DFF) delays its input with one clock cycle. Then, $Acc(u,0)$ is obtained by subtracting the output of the M-th DFF ($\sum_{i=0}^{u-1}\sum_{j=0}^{N-1} A(i,j)$) from the output of "ColAccu1". Therefore, $Acc(u,0)$ can be computed during inputting the real image from the external RAM (Exter-ORRAM in Fig. 7) to the internal RAM blocks (RRAM[0], . . ., RRAM[M-1]) of the FPGA.

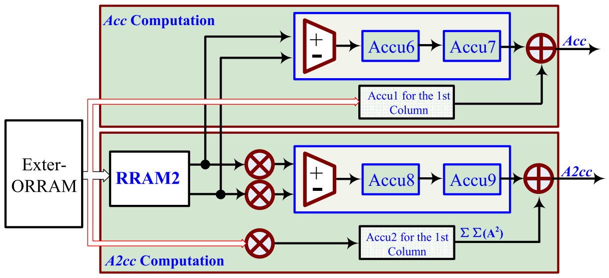

FIGURE 7. The block diagram of the *Acc* and *A2cc* computation.

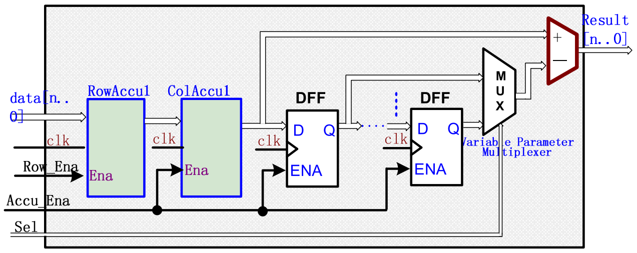

FIGURE 8. The logic schematic of the module "Accu1 for the 1st Column" in Fig. 7.

In the same way, $A2cc(u,0)$ can be implemented with the module "Accu2 for the 1st Column" and a square operation module, as shown in Fig. 7.

From the second searching position of each row, i.e., for any given $v_0(v_0 \geq 0)$, as shown in Fig. 1, $Acc(u, v_0+1)$ can be calculated as follows:

$$\begin{aligned} &Acc(u,v_0+1) \\ &= Acc(u,v_0) + \sum_{i=0}^{M-1} A(i+u,N+v_0) - \sum_{i=0}^{M-1} A(i+u,v_0) \\ &= Acc(u,0) + \sum_{z=0}^{v_0} \\ &\quad \times \left[\sum_{i=0}^{M-1} A(i+u,N+z) - \sum_{i=0}^{M-1} A(i+u,z)\right] \\ &= Acc(u,0) + \sum_{z=0}^{v_0-1} \\ &\quad \times \left[\sum_{i=0}^{M-1} A(i+u,N+z) - \sum_{i=0}^{M-1} A(i+u,z)\right] \\ &\quad + \left[\sum_{i=0}^{M-1} A(i+u,N+v_0) - \sum_{i=0}^{M-1} A(i+u,v_0)\right] \end{aligned}$$

According to the above equation, $Acc(u, v_0+1)$ can be directly implemented via a parallel adder, an accumulator, and an adder in two clock periods, as proposed in traditional architecture [15]. The M-input parallel adder adds the pixels ($A(i+u,N+v_0)$, $0 \leq i \leq M-1$) in the $(N+v_0)$-th column (the "new column" as shown in Fig. 1) in parallel, and then subtracts the pixels ($A(i+u,v_0)$, $0 \leq i \leq M$-1) in the v_0-th column (the "old column") in parallel. Then, the accumulator adds its current content with the output from the parallel adder. Lastly, $Acc(u,v_0+1)$ is obtained through adding the above accumulated result with $Acc(u,0)$ via the adder. With additional M parallel square operations, $A2cc(u,v_0+1)$ can also be computed in the same way in only two clock periods.

$Acc(u,v_0+1)$ can also be rewritten as,

$$\begin{aligned} &Acc(u,v_0+1) \\ &\quad = Acc(u,v_0)\sum_{i=0}^{M-1}[A(i+u,N+v_0) - A(i+u,v_0)] \\ &\quad = Acc(u,0)\beta(u,v_0) + \alpha(u,v_0) \qquad (3) \end{aligned}$$

where

$$\begin{aligned} \alpha(u,z) &= \sum_{i=0}^{M-1}[A(u+i,N+z) - A(u+i,z)], \\ \beta(u,v_0) &= \sum_{z=0}^{v_0-1} \alpha(u,z). \end{aligned}$$

$\alpha(u,z)$ can be calculated in serial mode through accumulating each $A(u+i,N+z)$-$A(u+i,z)$ $(0\leq i\leq M$-1). The above idea can be used to implement the $Acc(u,v_0+1)$ and $A2cc(u,v_0+1)$ computation within M clock periods using one dual-port RAM (RRAM2), as shown in Fig. 7. During the $ABcc$ computation at a given searching location (u, v_0), the pixels in the $(N+v_0)$-th column and the v_0-th column $(A(i+u, N+v_0)$ and $A(i+u, v_0)$, $(0\leq i\leq M$-1)) are output simultaneously from the dual-port RAM to the subtractor under the control of the module "timing control". The module "Accu6" accumulates the subtracted result $(A(i+u,N+v_0)$-$A(i+u,v_0))$ of the subtractor for the $(i+u)$-th row, and $\alpha(u,v_0)$ is obtained. Then, the second accumulator "Accu7" adds its current content with the result from "Accu6" and outputs $\beta(u,v_0)+\alpha(u,v_0)$. Finally, $Acc(u,v_0+1)$ is obtained through adding the output of "Accu7" to that of "Accu2 for the 1st Column" $(Acc(u,0))$ via an adder.

In the same way as the Acc computation, the $A2cc$ computation can also be implemented in such a serial mode. The $A2cc$ module consists of a subtractor, an adder, the modules "Accu8" and "Accu9", except for two extra squaring operations, as shown in Fig. 7.

Therefore, the parallel-addition operations of the Acc and $A2cc$ computation, along with M-2 square operations for the $A2cc$ computation used in the traditional architecture can be cancelled, where M and 2 are the numbers of square operations used in the $A2cc$ computation of the traditional architecture and the proposed architecture, respectively. Consequently, compared with traditional architecture, the resource usage, including logic gates and DSP blocks, is reduced.

3) BCC AND B2CC COMPUTATION

For a given reference image, Bcc and $B2cc$ are constants and can be computed only once in the whole searching process. Therefore, Bcc and $B2cc$ can be calculated using an accumulator, and a square operation module followed by an accumulator, respectively, during inputting the reference image from the external RAM (Exter-ORRAM) to the internal RAM blocks (ORAMA[0], …, ORAMA[M-1]), as shown in Fig. 5.

4) MORE ABOUT THE ZNCC COMPUTATION

Once Bcc, $B2cc$, Acc, $A2cc$, and $ABcc$ have been obtained, it is easy to calculate $M\cdot N\cdot B2cc$-$(Bcc)^2$, $M\cdot N\cdot A2cc$-$(Acc)^2$, and $M\cdot N\cdot ABcc$-$Acc\cdot Bcc$. In the proposed architecture, the denominator of the ZNCC measure is computed first by two square-root modules "Sqrt" and then by one multiplier module, as shown in Fig. 9. To facilitate the subsequent processing, the numerator and the denominator of (2) are first converted from the fixed-point to the floating-point format using the modules "fpConvert1" and "fpConvert2", respectively. Then, the ZNCC measure can be obtained through a floating-point division operation with the output in the floating-point format.

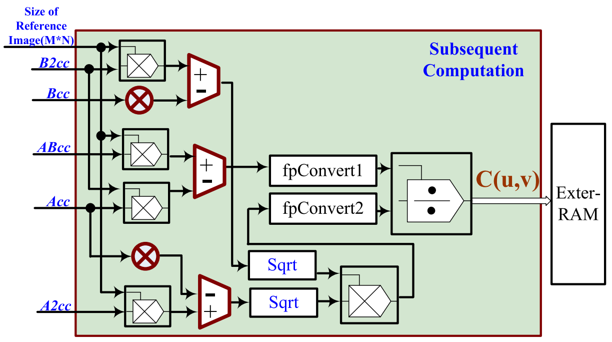

FIGURE 9. The block diagram of the subsequent computation.

C. THE WORKFLOW OF THE PROPOSED ARCHITECTURE

The workflow of the proposed architecture consists of the following sequential and parallel steps after the system initialization with $m = 0$, $n = 0$, R(0) = ORAMA, and R(1) = ORAMB, as shown in Fig. 10, where all subprocesses in every step are performed in parallel. The sequential steps are denoted by the symbol Sx, where x is an integer number.

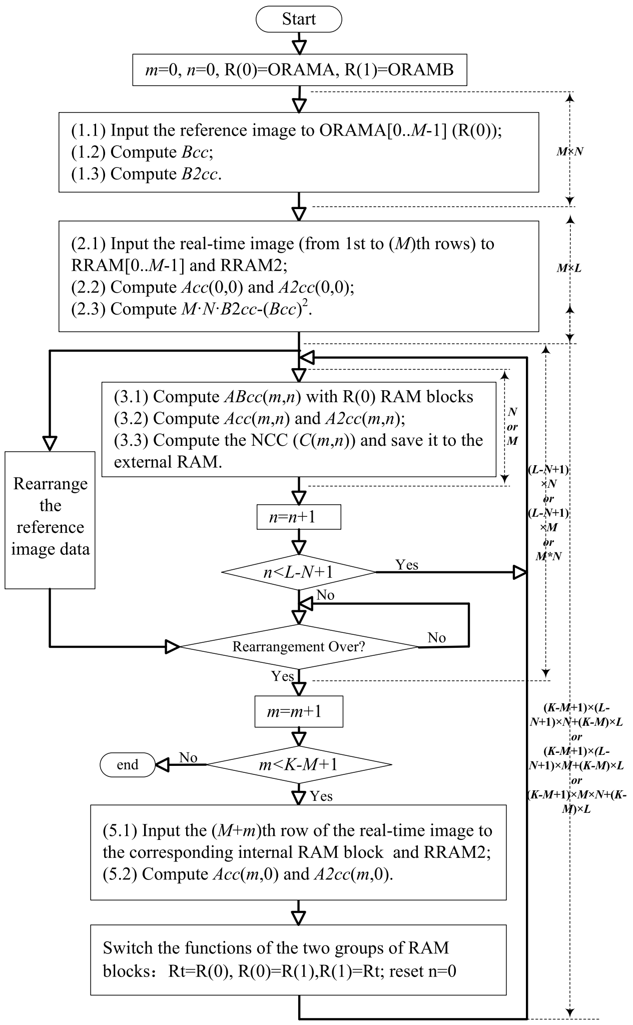
Start
m=0, n=0, R(0)=ORAMA, R(1)=ORAMB
(1.1) Input the reference image to ORAMA[0..M-1] (R(0));
(1.2) Compute Bcc;
(1.3) Compute B2cc.
M×N
(2.1) Input the real-time image (from 1st to (M)th rows) to RRAM[0..M-1] and RRAM2;
(2.2) Compute Acc(0,0) and A2cc(0,0);
(2.3) Compute M·N·B2cc-(Bcc)².
M×L
(3.1) Compute ABcc(m,n) with R(0) RAM blocks
(3.2) Compute Acc(m,n) and A2cc(m,n);
(3.3) Compute the NCC (C(m,n)) and save it to the external RAM.
N or M
Rearrange the reference image data
n=n+1
n<L-N+1
Yes
No
Rearrangement Over?
No
Yes
(L-N+1)×N or (L-N+1)×M or M*N
m=m+1
end
No
m<K-M+1
Yes
(K-M+1)×(L-N+1)×N+(K-M)×L or (K-M+1)×(L-N+1)×M+(K-M)×L or (K-M+1)×M×N+(K-M)×L
(5.1) Input the (M+m)th row of the real-time image to the corresponding internal RAM block and RRAM2;
(5.2) Compute Acc(m,0) and A2cc(m,0).
Switch the functions of the two groups of RAM blocks: Rt=R(0), R(0)=R(1),R(1)=Rt; reset n=0

FIGURE 10. The workflow of the proposed architecture.

Step S1: (1.1) Input the reference image from the external RAM (Exter-ORAM) to one group (R(0)) of the internal RAM blocks (ORAMA[0], ORAMA[1], …, ORAMA[M-1]) in the row order; at the same time, (1.2) accumulate these data with the module "Accu2" for the *Bcc* computation; and (1.3) accumulate the squares of these data with the module "Accu1" for the *B*2*cc* computation.

Step S2: (2.1) Input the data from the first up to the M-th rows of the real image from the external RAM (Exter-RRAM) to the internal RAM blocks (RRAM[0], RRAM[1],…, RRAM[M-1]) and RRAM2 in the row order; at the same time, (2.2) compute *Acc*(0,0) and *A*2*cc*(0,0) of the portion of the real image under examination with the modules "Accu1 for the 1st column" and "Accu2 for the 1st column", respectively; and (2.3) compute $M \cdot N \cdot B2cc$ - $(Bcc)^2$.

Step S3: (S3A) Compute *ABcc*(*m*,*n*), *Acc*(*m*,*n*), *A*2*cc*(*m*,*n*) in the order from the first to the (L-N + 1)-th columns of the m-th searching row; at the same time, (S3B) rearrange the reference image.

Step S4: $m = m + 1$. If $m < K$-$M + 1$, then proceed to Step S5; else, stop the whole process and change the status of the completion indicator.

Step S5: Once the steps from S2 to S4 for the ZNCC computation and the data rearrangement of the reference image have been finished, (5.1) input the $(M + m)$-th row of the real image from the external RAM to the corresponding internal RAM block and RRAM2; at the same time, (5.2) compute *Acc*(*m*,0) and *A*2*cc*(*m*,0) with the modules "Accu1 for the 1st column" and "Accu2 for the 1st column", respectively.

Step S6: Switch the functions of the two groups of RAM blocks between the ZNCC computation and the data rearrangement for the ZNCC computation at the next row: Rt = R(0), R(0) = R(1), and R(1) = Rt. Reset $n = 0$. Return to Step S3.

In the above workflow, Step S3 comprises two parallel processes, Step S3A and S3B. The details of Step S3A and Step S3B are given, as shown in Fig. 10.

Step S3A: (3.1) Compute *ABcc*(*m*,*n*) with the group of RAM blocks (R(0)) of the reference image within N clock periods; at the same time, (3.2) compute *Acc*(*m*,*n*) and *A*2*cc*(*m*,*n*) in serial mode within M clock periods; and (3.3) compute C(*m*,*n*) and save it into the external RAM.

Step S3B: Rearrange the reference image through reading the image from the external RAM to the other group of RAM blocks (R(1)) according to the order of the internal RAM blocks of the real image (RRAM) that will be updated in Step S5 for the ZNCC computation at the $(m + 1)$-th row.

In the proposed architecture, since the data of the reference and the real images are stored in the same external RAM (Exter-ORRAM), the data input from the outer RAM can be performed separately for the reference and the real images, as shown in Fig. 2. The data input can also be performed simultaneously if the data of the two images are stored in two individual outer RAMs. Then, the corresponding steps in the workflow can be further optimized.

D. PRACTICAL IMPLEMENTATION

From the above analyses, the computational time of the above proposed architecture can be calculated from the workflow, which has been labeled with the computation time in clock periods for each step, as shown in Fig. 10. Step S3A can be finished in T_1 clock periods until all the computations of *ABcc*(*m*,*n*), *Acc*(*m*,*n*), and *A*2*cc*(*m*,*n*) at a given searching position are finished, where T_1 is given as follows:

$$T_1 = \begin{cases} M, & M > N \\ N, & \text{otherwise} \end{cases}$$

Step S3 can be finished in T_2 clock periods.

$$T_2 = \begin{cases} (L - N + 1) \cdot T_1, & \text{if } (L - N + 1) \cdot T_1 > M \cdot N \\ M \cdot N, & \text{otherwise} \end{cases}$$

Step S5 will be performed only after all the steps from S3 to S4 have been finished. Therefore, the ZNCC computation can be finished in *Ct* clock periods,

$$Ct = K \cdot L + M \cdot N + (K - M + 1) \cdot T_2$$

Then, the total computation time of the ZNCC computation is given as *Ct*/f*clk*, where *fclk* is the global clock frequency of the system.

In this work, 8-bit grayscale image data are input to the system with a reference image with variable size of $M \times N$ ($2 \leq M \leq 80$, $2 \leq N \leq 80$), and a real-time image with variable size of $K \times L$ ($2 \leq K \leq 512$, $2 \leq L \leq 512$), i.e., $Mmax = Nmax = 80$, and $Kmax = Lmax = 512$. Accordingly, the proposed architecture with the desired data width was implemented with 80 parallel MAC operations on the EP2S90F780I4 (the FPGA device from Altera Corporation) [16]. The integrated development software, Quartus II 8.0 with SP1 [17], was used to perform logic analysis, synthesis, placement, routing, and function and timing simulation. All of the modules in the proposed architecture were implemented in the VHDL (VHSIC Hardware Description Language) [18] and synthesized with Quartus II. The system global clock was chosen to be 70 MHz, which was generated by PLL with an external 25 MHz clock input. For $M = Mmax$, $N = Nmax$, $K = Kmax$, and $L = Lmax$, since $M = N$ and $(L$-$N + 1) \cdot N > M * N$, $T_2 = 34640$ and the total processing time is theoretically 218.1 ms.

To achieve sufficient positioning accuracy of the practical ATR system, the numerical accuracy of the result of the ZNCC calculation is required to reach the order of 10^{-6}. Therefore, the data in the fixed-point format for all the operations of both the proposed and the traditional architectures have been sufficiently extended.

III. SIMULATION AND EXPERIMENTAL RESULTS

A. COMPILATION RESULTS

According to the compilation result of Quartus II, an 80-to-1 multiplexer needs 237 ALUTs, while a 2-to-1 multiplexer needs only 8 ALUTs. Therefore, 18320 (80 × (237-8))

ALUTs will be saved in the proposed architecture with 80 2-to-1 multiplexers, compared to the traditional architecture with 80 80-to-1 multiplexers. In addition, a portion of the ALUTs will be saved for the cancellation of the parallel-addition modules for the Acc and $A2cc$ computation. The resource usages of the two architectures are listed in Table 1, including the numbers of ALUTs, registers and MACs (included in DSP blocks), given by the reports from the compiler. As shown in Table 1, compared with the traditional architecture, the usage of ALUTs of the proposed architecture is decreased from 33684 to 13929, and the reduced number is approximately equal to the theoretical value, 18320. As a result, the utilization of the ALUTs is decreased from 46% to 19%, and the logic utilization is decreased from 63% to 35%. The usage of RAM bits is increased by 378880 bits (equal to the theoretical value, $80 \times 80 \times 8 + 512 \times 80 \times 8$) and is only increased by 8% compared with the traditional architecture. The usage of DSP blocks is decreased from 226 to 148 (decreased by 20%), and the reduced number is equal to the theoretical value (M-2 = 78).

TABLE 1. The resource usage of the proposed architecture.

Resource Type	Total Resources	Traditional Used(Percent)	Proposed Used(Percent)
Logic Utilization	72,768	49,782(63%)	25,757(35%)
ALUTs	72,768	33,684(46%)	13,929(19%)
Registers	72,768	22,497(31%)	20,852 (29%)
Memory bits	4,520,488	590,592(13%)	969,472(21%)
DSP blocks	384	226(59%)	148(39%)

The propagation delays and the total thermal power dissipation have also been estimated for the two architectures as a reference for practical applications using the classic timing analyzer tool and the power analyzer tool (PowerPlay) integrated with Quartus II, respectively. Compared with the traditional architecture, the propagation delay of the proposed architecture is decreased from 9.05 ns to 8.29 ns (decreased approximately by 8.4%), and the total thermal power dissipation of the proposed architecture is decreased from 2347.00 mW to 1878.29 mW (decreased approximately by 19.9%).

B. SIMULATION RESULTS

To verify the logical correctness of the proposed architecture, a group of simulated image data was input to the system with the reference image of size 17×17 and the real image of size 40×40, where 17 and 40 are the input parameters of the system. For functional verification and easy simulation, the data input for the real image starts from 0 and increases by 1 along the direction of the row, the data input for the reference image starts from 64 and increases by 1 along the direction of the row. The data of the two images are constrained in the range from 0 to 255, the upper bound of an 8-bit unsigned fixed-point datum. If the input value is greater than 255, any carry bit is considered as overflow and will be discarded.

The simulation waveform of the proposed architecture is shown in Fig. 11, where the nodes Bcc, $B2cc$, Acc, $A2cc$, and $ABcc$ are defined as before. The output nodes Result_S, Result_E, and Result_M denote the sign, exponent, and mantissa components of the results of the ZNCC measure in the 32-bit floating-point format, respectively. The input node clk5 is the system clock.

The results of the proposed architecture have been compared with the theoretical ones, as listed in Table 2, where only the first 6 groups of the results are shown for the sake of space. According to the simulation waveform of the proposed architecture, Bcc and $B2cc$ are constants ($Bcc = 35280$ and $B2cc = 5773872$), and the values of Acc, $A2cc$, and $ABcc$ are the same as the theoretical ones. In the table, $Result\_M2$ denotes the mantissa of the theoretical result in the 32-bit floating-point format. It clearly illustrates that the proposed architecture can achieve the accuracy on the order of 10^{-6}.

TABLE 2. The simulation and theoretical results.

No.	*Acc*	*A2cc*	*ABcc*	*Result_M*	*Result_M2*
0	33608	5434712	4277336	7271790	7.2717890e+006
1	33641	5436681	4266280	5945662	5.9456620e+006
2	33674	5438716	4255480	4637394	4.6373932e+006
3	33707	5440817	4244936	3347143	3.3471433e+006
4	33740	5442984	4234648	2075074	2.0750735e+006
5	33773	5445217	4224616	821344	8.2134461e+005

C. PRACTICAL EXPERIMENTAL RESULT

In our practical automatic target recognition system, the block diagram of the ZNCC-based image matching subsystem is shown in Fig. 12, where Exter-ORRAM and Exter-RAM are the external RAMs used for buffering the reference and the real images and the results of the ZNCC computation, respectively. Addr and Data are the address and data buses, respectively. RD, WR and CS are the control signals used for data input from and output to the external RAM. ADSP-TS201S [19] is an embedded processor from ADI Corporation.

The whole work process of the above subsystem is given as follows. First, the embedded processor (ADSP-TS201S) issues the commands to send the data of the reference and the real images to the RAMs (Exter-ORRAM and Exter-RAM) and to send the parameters of the image sizes and the start command to the FPGA. Then, the FPGA starts to perform the ZNCC computation and saves the results to the external RAM (Exter-RAM). When the computation is finished, the FPGA transmits an interrupt signal to ADSP-TS201S to indicate the completion of the image matching procedure. In return, ADSP-TS201S will also query the associated status register of the FPGA to confirm the completion before further processing. The long-term stability test has been conducted for the proposed architecture using infrared image data with

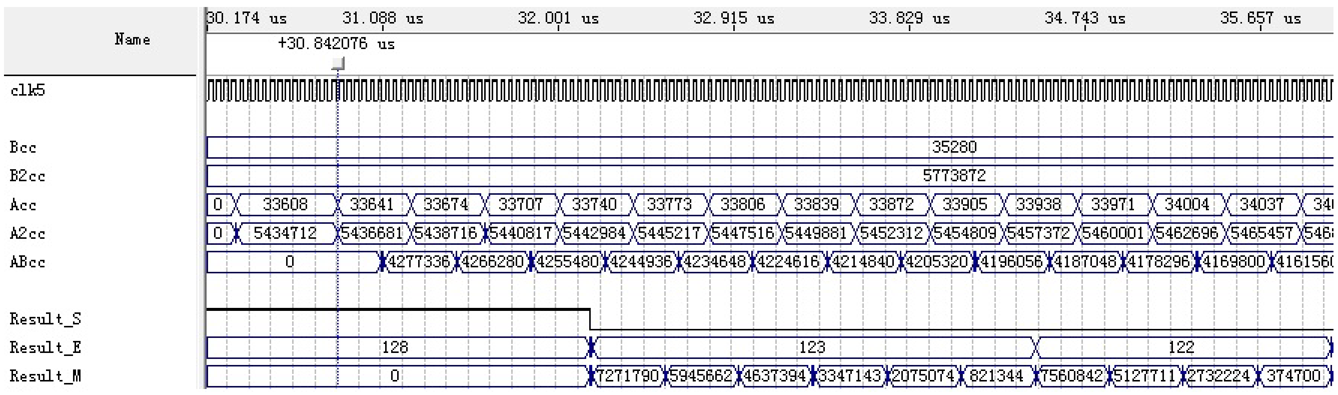

FIGURE 11. The simulation waveform of the proposed architecture.

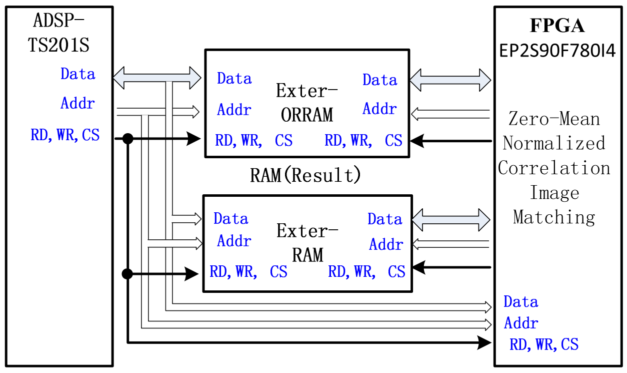

FIGURE 12. The block diagram of zero-mean normalized cross-correlation image matching subsystem.

different parameters. It has shown that the results of the ZNCC computation in template matching met the requirements for practical localization precision, and the system can work well. The computation time of the architecture has been evaluated using ADSP-TS201S and was identical to the theoretical time given in the previous section. For a reference image of size 80 × 80 and a real-time image of size 512 × 512, at a system clock of 70 MHz, the time consumed for the proposed architecture is lower than the one (224 ms in [15]) for the traditional architecture. It is obvious that the proposed architectures can further improve the system speed.

The performance comparisons of the implementations of the FPGA and the multiple parallel microprocessors using ADSP-TS201S have also been conducted. The speed of the FPGA implementation is approximately equal to that of implementation with four parallel microprocessors in 32-bit single precision floating-point format at the full speed of 500 MHz. However, the precision that the latter implementation can achieve is significantly lower than the precision requirement of the order of 10^{-6} due to the rounding errors of the intermediate results. If the implementation with four parallel microprocessors uses the 64-bit double precision floating-point format, although it can meet the precision requirement, its speed will be decreased by 3 to 4 times. In addition, it is apparent that the implementation based on FPGA can greatly reduce the system size and power consumption, compared with the implementation based on multiple microprocessors.

IV. DISCUSSION

As mentioned above, since the *ABcc* computation in the numerator of the ZNCC definition is just a standard CC, the module of the *ABcc* computation can be used to compute the standard CC measure. Accordingly, the workflow for the ZNCC computation can also be adapted to the computation of the standard CC.

From a practical point of view, the numerical precision of the algorithm must be taken into consideration because of the possible existence of false maximum values induced by the numerical truncation in the computation process. To obtain the subpixel accuracy of localization, the first several maximum values of the ZNCC results are required for further processing, such as surface fitting, and the architecture proposed in [20] can also be used with slightly extra computation time and resource consumption.

In addition, FPGAs have been increasingly used as an efficient platform for the ASIC prototype development and verification. Although we mainly consider the parallel implementation of the ZNCC based on the FPGA in this paper, the proposed architecture can also be used to develop an efficient ASIC for the ZNCC-based template matching.

V. CONCLUSION

To meet the requirements for small size and low-power dissipation in an embedded automatic target recognition system, a novel resource-efficient parallel architecture based on an FPGA has been proposed in this paper for the implementation of the ZNCC computation. In the proposed architecture, two groups of RAM blocks, each of which is used to buffer the reference image, are alternately used for the *ABcc* computation and the data rearrangement of the reference image. The functions of the two groups of RAM blocks are switched back and forth through 2-input multiplexers when computing the ZNCC measure at different rows of the real image.

Moreover, the *Acc* computation is implemented through serially accumulating the results of subtracting the old column in the last searching area from the new column in the new searching area using one dual-port RAM. Simultaneously, the *A*2*cc* is implemented in the same way. Consequently, compared with the traditional architecture, the logic utilization and DSP blocks of the proposed architecture are enormously decreased with a slight increase in memory utilization. Compilation and simulation results on a Stratix II FPGA device (EP2S90F780I4) as well as practical experiments in an automatic target recognition system have shown that the proposed architecture can effectively improve the performance of the practical target recognition system.

REFERENCES

[1] J. Luo and E. Konofagou, "A fast normalized cross-correlation calculation method for motion estimation," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 57, no. 6, pp. 1347–1357, Jun. 2010.

[2] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Hoboken, NJ, USA: Wiley, 2011.

[3] L. D. Stefano, S. Mattoccia, and M. Mola, "An efficient algorithm for exhaustive template matching based on normalized cross correlation," in *Proc. 12th IEEE Int. Conf. Image Anal. Process.*, Sep. 2003, pp. 322–327.

[4] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*. New York, NY, USA: Academic, 1982, pp. 200–212.

[5] J. P. Lewis, "Fast template matching," in *Proc. Can. Image Process. Pattern Recognit. Soc. Vis. Interface*, Quebec City, QC, Canada, May 1995, pp. 120–123.

[6] D. M. Tsai and C. T. Lin, "Fast normalized cross correlation for defect detection," *Pattern Recognit. Lett.*, vol. 24, no. 15, pp. 2625–2631, Nov. 2003.

[7] L. J. Siegel, H. J. Siegel, and A. E. Feather, "Parallel processing approaches to image correlation," *IEEE Trans. Comput.*, vol. C-31, no. 3, pp. 208–218, Mar. 1982.

[8] N. Gupta and N. Gupta, "A VLSI architecture for image registration in real time," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 9, pp. 981–988, Sep. 2007.

[9] M. Cavadini, M. Wosnitza, and G. Troster, "Multiprocessor system for high-resolution image correlation in real time," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 3, pp. 439–449, Jun. 2001.

[10] S. Mori, M. Kumagai, K. Miki, R. Fukuhara, and H. Haneishi, "Development of fast patient position verification software using 2D-3D image registration and its clinical experience," *J. Radiat. Res.*, vol. 56, pp. 818–829, Sep. 2015.

[11] A. Lindoso and L. Entrena, "High performance FPGA-based image correlation," *J. Real-Time Image Process.*, vol. 2, pp. 223–233, Dec. 2007.

[12] J.-Y. Chen, K.-F. Hung, H.-Y. Lin, Y.-C. Chang, Y.-T. Hwang, C.-K. Yu, C.-R. Hong, C.-C. Wu, and Y.-J. Chang, "Real-time FPGA-based template matching module for visual inspection application," in *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, Jul. 2012, pp. 1072–1076.

[13] E. S. Albuquerque, A. P. A. Ferreira, J. G. M. Silva, J. P. F. Barbosa, R. L. M. Carlos, D. S. Albuquerque, and E. N. S. Barros, "An FPGA-based accelerator for multiple real-time template matching," in *Proc. 29th Symp. Integr. Circuits Syst. Design, Belo Horizonte*, Brazil, Brasilia, Aug./Sep. 2016, pp. 1–6.

[14] Z. Chen, "Eye-to-hand robotic visual tracking based on template matching on FPGAs," *IEEE Access*, vol. 7, pp. 88870–88880, 2019.

[15] X. Wang and X. Wang, "FPGA based parallel architectures for normalized cross-correlation," in *Proc. 1st Int. Conf. Inf. Sci. Eng. (ICISE)*, 2009, pp. 225–229.

[16] *Stratix II Device Handbook*, Altera, San Jose, CA, USA, 2007.

[17] *Quartus II Version 8.0 Handbook*, Altera, San Francisco, CA, USA, 2007.

[18] B. Brown and Z. Vranesic, *Fundamentals of Digital Logic With VHDL Design*, 2nd ed. Toronto, ON, Canada: McGraw-Hill, 2005.

[19] Analog Devices. (2006). *TigerSHARC Embedded Processor ADSP-TS201S*. Norwood, MA, USA. [Online]. Available: http://www.analog.com/media/en/technicaldocumentation/data-sheets/ADSPTS201S.pdf

[20] S. Dong, X. Wang, and X. Wang, "A novel high-speed parallel scheme for data sorting algorithm based on FPGA," in *Proc. 2nd Int. Congr. Image Signal Process. (CISP)*, Tianjin, China, 2009, pp. 1–4.

XIAOTAO WANG was born in Shandong, China, in 1976. He received the B.S. degree in automation, the M.S. degree in navigation guidance and control, and the Ph.D. degree in control theory and engineering from the Harbin Institute of Technology, Harbin, China, in 1999, 2001, and 2005, respectively.

Since 2010, he has been an Associate Professor with the College of Astronautics, Nanjing University of Aeronautics and Astronautics. His current research interests include high-speed parallel implementation of image processing algorithms and motion control technique for space mechanism.

XINGBO WANG received the Ph.D. degree from Shandong University, Jinan, China, in 2011. Since 2012, he has been a Lecturer with the College of Automation and the College of Artificial Intelligence, Nanjing University of Posts and Telecommunications. His main research interests include high-speed parallel implementation of image processing algorithms and wireless sensor networks.

LIANGLIANG HAN received the M.S. degree from the Shanghai Academy of Spaceflight Technology, Shanghai, China, in 2013.

Since 2010, he has been an Engineer with the Shanghai Institute of Aerospace System Engineering. His current research interests include mechanism design and control algorithms of space robot and its end-effectors.