# Template Matching using DSP slices on the FPGA

Kaoru Hashimoto, Yasuaki Ito, Koji Nakano
*Department of Information Engineering*
*School of Engineering, Hiroshima University*
*1-4-1 Kagamiyama, Higashi-Hiroshima, 739-8527, JAPAN*

***Abstract*—The main contribution of this paper is to propose an FPGA implementation of template matching using DSP slices. Template matching is a technique for finding small parts of an image which match a template image. In our approach, we use a pixel rearrangement technique that is a coarse-to-fine technique. Unlike ordinary coarse-to-fine techniques, it always can find a template image in a base image if the template image is included in the base image. In our implementation, we use multiple matching modules that compute similarity and work in parallel. In each matching module, we efficiently use embedded DSP slices on the Virtex-6 FPGA. We have implemented the template matching in a Xilinx Virtex-6 FPGA XC6VLX240T-FF1156. The implementation results show that it can be implemented in the FPGA with 352 DSP slices, 3 block RAMs and 455 CLBs. It runs in approximately 280MHz clock frequency. The computing time of our FPGA implementation is 348.88 and 3.66 times faster than that of CPU and GPU implementations, respectively.**

***Keywords*-Template matching, FPGA, DSP slice, Block RAM, Pipeline, Parallel processing**

## I. Introduction

A Field Programmable Gate Array (FPGA) is a logic device that can provide programmability for customers to implement their own logic. Since FPGA chip maintains relative lower price and its programmable features, it is widely used in those fields which need to update architecture or functions frequently such as communication and education areas. The most common FPGA architecture consists of an array of logic blocks, I/O pads, and routing channels. Furthermore, resent FPGAs have embedded DSP slices and block RAMs that make a higher performance and broader applications. They are widely used not only in consumer and industrial products but also in academic research for accelerating processor intensive algorithms [1]–[7].

The Xilinx Virtex-6 series FPGAs have DSP48E1 slices equipped with a multiplier, adders, logic operators, etc [8]. More specifically, as illustrated in Figure 1, the DSP48E1 slice has a two-input multiplier followed by multiplexers and a three-input adder/subtractor/accumulator. The DSP48E1 multiplier can perform multiplication of an 18-bit and a 25-bit two's complement numbers and produce one 48-bit two's complement production. Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput and improves the frequency. The DSP48E1 also has pipeline registers between operators to reduce the delay. The Xilinx FPGA XC6VLX240T has 768 DSP48E1 slices arranged in 8 columns of 96 adjacent DSP48E1 slices. Neighboring DSP48E1 slices are connected directly through pipeline registers that are not show in the figure. The block RAM in the Virtex-6 FPGA is an embedded memory supporting synchronized read and write operations. In Virtex-6 FPGAs, it can be configured as a 36k-bit dual-port RAMs, FIFOs, or two 18k-bit dual-port RAMs.

Template matching is one of the techniques for detecting a given *template image* from an image called a *base image*, and examining whether the template exists in the base image to be detected. It is widely used for industrial manufacturing, robot navigation, geographical research, image registration, etc [9]–[12]. Figure 2 shows an example of template matching. Given a template image and base image, template matching is to find a position such that a subimage in the base image is the most similar to the template image. There are some measurements of the similarity between a template image and a subimage of the base image. In this paper, we use *the normalized correlation coefficient* as the similarity measure [13]. It is a normalized measurement with the average and standard deviation of a template image and a base image.

To reduce the computing time of template matching, numerous methods have been developed. One of the most famous methods is coarse-to-fine template matching [9], [14]–[16]. It locates a low-resolution template image into the low-resolution base image, and then refines the search at higher resolution levels. In this algorithm, it is important to make low-resolution images because the low-resolution template image is not always found in the low-resolution base image. For example, let us consider the case that a low-resolution image is made by sampling every two pixels. When a base image and a template image are checkered patterns as shown in Figure 3, their low-resolution images may be different from each other. Although the base image includes the template image, it occurs that the template matching may be fault. To avoid such faults by sampling, blurred low-resolution images are made by low-pass filter such as Gaussian filter, Laplacian filter, wavelet transform, and so on [17]–[22]. However, it is not always possible to find the low-resolution template image. On the other hand, to accelerate the speed of template matching, various methods supported by hardware acceleration with GPUs [16], [21], [23]–[25] and FPGAs [5], [26] have been presented.

Paper [16] shows a template matching algorithm with pixel rearrangement. This algorithm is based on the coarse-to-fine template matching. Unlike usual coarse-to-fine algorithms, however, the feature of this algorithm is that if

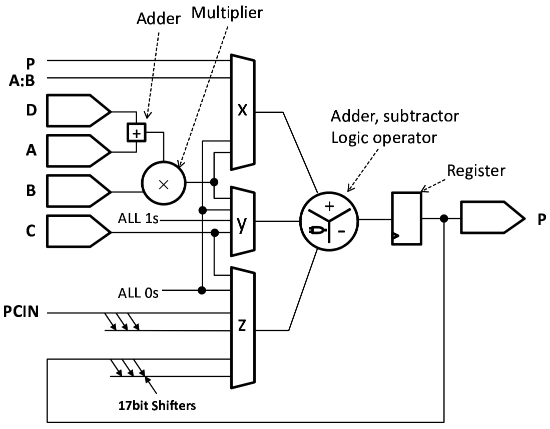


Figure 1. Architecture of DSP48E1

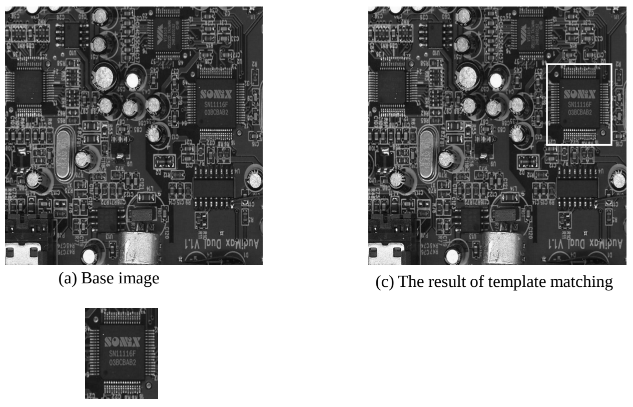
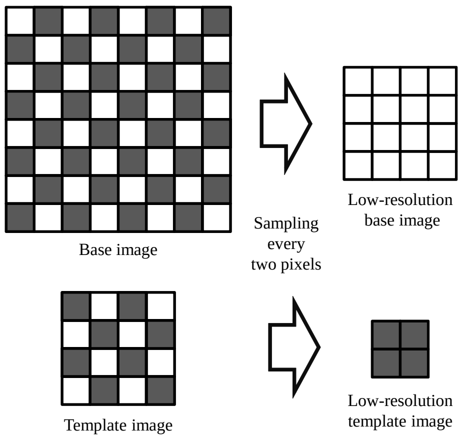


Figure 2. Example of Template Matching



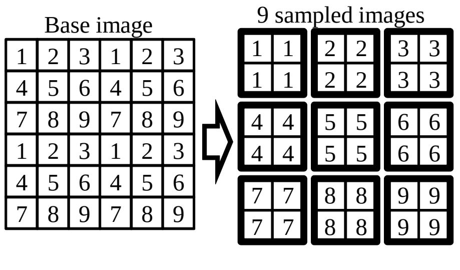
Base image
Sampling every two pixels
Low-resolution base image
Template image
Low-resolution template image

Figure 3. Low-resolution images by sampling

a template image is included in a base image, this algorithm always can find the template. In this algorithm, low-resolution images by rearranging pixels of the base image are generated. In the existing sampling-based algorithms, one low-resolution base and template images are generated, and template matching is performed for them. On the other hand, this algorithm generates $k^2$ low resolution base images by sampling every $k$ pixels. The sampling is performed for the base image shifted pixel by pixel from 0 to $k-1$ pixels. Therefore, in our algorithm, given an $n \times n$ base image, $k^2$ sampled base images whose size is $\frac{n}{k} \times \frac{n}{k}$ are generated. More specifically, let $I'_{s,t}$ $(0 \leq s,t \leq k-1)$ be $k^2$ sampled base images, and they are sampled from a base image such that each sampled image is

$$I'_{s,t}(x,y) = I(kx+s, ky+t) \quad (0 \leq x,y \leq \frac{n}{k}-1).$$

Figure 4 shows an example of sampling for $k=3$. Since the size of a base image is $6 \times 6$, $k^2$ sampled images of size $2 \times 2$ are generated. The readers should have no difficulty to confirm that the size of a base image is equal to the total size of the $k^2$ sampled images. Therefore, we can say that our sampling manner is equivalent to rearrangement of a template image. Also, a template image is reduced by sampling every $k$ pixels. Although the base image and template image are checkered patterns shown in Figure 3, at least one low-resolution image includes the same pattern as the low-resolution base image. In other words, if the original base image includes the original template image, it is always to find it in one of the low-resolution images using template matching with pixel rearrangement. After that, we perform template matching for the low-resolution template image and each sampled base image. Form the results thus obtained, template matching is performed for the corresponding positions in the original base image and template image.



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 1 | 2 | 3 |
| 4 | 5 | 6 | 4 | 5 | 6 |
| 7 | 8 | 9 | 7 | 8 | 9 |
| 1 | 2 | 3 | 1 | 2 | 3 |
| 4 | 5 | 6 | 4 | 5 | 6 |
| 7 | 8 | 9 | 7 | 8 | 9 |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 3 | 3 |
| 1 | 1 | 2 | 2 | 3 | 3 |
| 4 | 4 | 5 | 5 | 6 | 6 |
| 4 | 4 | 5 | 5 | 6 | 6 |
| 7 | 7 | 8 | 8 | 9 | 9 |
| 7 | 7 | 8 | 8 | 9 | 9 |

Figure 4. Pixel rearrangement for $k=3$

The main contribution of this paper is to present a new implementation of template matching. Our new idea includes: **(1) Pixel rearrangement:** Our template matching circuit is based on the idea of the pixel rearrangement method as shown the above. The difference is that in our proposed approach, a template image is rearranged. On the other hand, in the original pixel rearrangement method, a base image is rearranged instead of a template image. However, the readers can easily find that there is no difference between them in essence. Namely, this algorithm is a coarse-to-fine template matching and if a template image is included in a base image, it always can find the template, too. Sampling a template image, the template matching is performed for a low-resolution base image and $k^2$ template low-resolution images. In our implementation, $k^2$ template matching modules are implemented and work in parallel. **(2) Use of DSP slices and block RAMs:** DSP slices are used to compute the similarity between a low-resolution base image and template images. DSP slices basically perform multiplication and accumulation to compute the dot-product. They are directly connected and work pipeline fashion. Also, we avoid the computation of square by pre-loading them into the block RAMs as look-up-tables.

Using these ideas, we have evaluated the performance of our template matching circuit using the Xilinx Virtex-6 family FPGA XC6VLX240T-FF1156. The implemented circuit can perform the template matching for $n = 1024$, $m = 16$, and $k = 4$. Namely, the sizes of a base image and a template image are $1024 \times 1024$ and $16 \times 16$, respectively. Also, the sizes of the low-resolution base and template images are $256 \times 256$ and $4 \times 4$, respectively. The readers may think that the size of the template image is too small. However, by sampling the base image and the template image beforehand or using a part of the template image as a template image, we can perform the template matching for larger template images [20]. We note that our proposed implementation performs the template matching only for the low-resolution images. This is because in [16], the ratio of the computing time for the low-resolution images to the whole computing time is more than 95%. The key to accelerate the whole computing time is to shorten the running time for the low-resolution images. Therefore, in our implementation, we suppose the other processes are performed by another processor, for example, an embedded processor in the same FPGA, a host PC connected to the FPGA, or etc. According to the results of our evaluation, we achieved speed-up factors of 348.88 and 3.66, respectively.

The remainder of this paper is organized as follows: Section II introduces the similarity between two images. In Section III, we show the template matching with pixel rearrangement. The FPGA implementation using DSP slices and block RAMs is shown in Section IV. Section V exhibits the performance of our proposed algorithm on the FPGA. Finally, Section VI offers concluding remarks.

## II. The Image Similarity between a Template Image and a Base Image

The main purpose of this section is to define the similarity $R(T, I)$ for a template image $T$ and a base image $I$ to clarify our work in this paper. There are many measurements of the similarity in template matching. In this paper, we use *the normalized correlation coefficient* as the similarity [13]. The normalized correlation coefficient is used to measure the correlation between two variables. We use it to evaluate the similarity of a template image and a base image in template matching as follows.

First, let us define the similarity of two images $A$ and $B$ of the same size. For simplicity, we assume that they are square, that is, the size of two images is $m \times m$. Let $A(i, j)$ and $B(i, j)$ denote the intensity level of an $(i,\ j)$ pixel $(0 \leq i, j \leq m - 1)$ of $A$ and $B$, respectively. The normalized correlation coefficient $R(A, B)$ between the two images $A$ and $B$ is computed by the following formula:

$$R(A,B) = \frac{\sum(A(i,j) - \overline{A})(B(i,j) - \overline{B})}{\sqrt{\sum(A(i,j) - \overline{A}) \sum(B(i,j) - \overline{B})}}, \qquad (1)$$

where $\overline{A} = \frac{1}{m^2}\sum A(i,j)$ and $\overline{B} = \frac{1}{m^2}\sum B(i,j)$ are the average pixel values of $A$ and $B$, respectively. The normalized correlation coefficient $R(A, B)$ takes a real number in the range $[-1, +1]$. Larger value of the normalized correlation coefficient implies that two images $A$ and $B$ are more similar. It should be clear that the normalized correlation coefficient $R(A, B)$ can be computed in $O(m^2)$ time by a sequential algorithm in an obvious way.

Suppose that a base image $I$ and a template image $T$ are given. Let $n \times n$ and $m \times m$ $(n > m)$ be the size of a base image $I$ and a template image $T$, respectively. Also, let $I(x, y)$ and $T(x, y)$ denote the intensity levels of $(x, y)$ pixels in $I$ and $T$, respectively. Let $I[x, y]$ $(0 \leq x, y, \leq n - m)$ denote an $m \times m$ subimage of $I$ that includes all pixels $I(i', j')$ $(x \leq i' \leq x+m-1$ and $y \leq j' \leq y+m-1)$. We define the similarity $R(T, I)$ between a template $T$ and a base image $I$ as follows:

$$R(T,I) = \max_{0 \leq x,y \leq n-m} R(T, I[x,y]).$$

Clearly, $R(T, I)$ is larger if $I$ has a more similar subimage to $T$. Also, the position $(x, y)$ that gives the maximum value of $R(T, I[x, y])$ corresponds to the most similar subimage $I[x, y]$ to the template image $T$. Let us evaluate the computing time necessary to compute $R(T, I)$ and the most similar position $R(T, I)$ by a sequential algorithm. For an $m \times m$ template image $T$ and a subimage $I[x, y]$, the value of $R(T, I[x, y])$ can be computed in $O(m^2)$ time. Hence, the evaluation of $R(T, I[x, y])$ for all $I[x, y]$ $(0 \leq x, y \leq n-m)$ takes $(n - m + 1)^2 \times O(m^2) = O(n^2m^2)$ time.

## III. Template Matching with Pixel Rearrangement

This section describes our proposed template matching algorithm with pixel arrangement. Given an $n \times n$ base image $I$ and an $m \times m$ template image $T$, in this algorithm, $k^2$ low-resolution template images $T'_{s,t}$ $(0 \leq s, t \leq k - 1)$ by pixel rearrangement such that

$$T'_{s,t}(x,y) = T(kx + s, ky + t) \quad (0 \leq x, y \leq \frac{m}{k} - 1),$$

as shown in Figure 5. The size of each $T'_{s,t}$ is $\frac{m}{k} \times \frac{m}{k}$.

Also, a low-resolution base image $I'$ is generated by sampling every $k$ pixels. After that, we perform template matching for the low-resolution base image and each sampled template image. Form the results, template matching is performed for the corresponding positions in the original

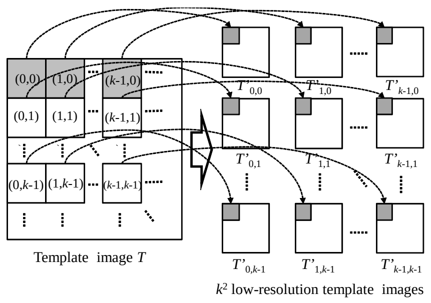


Figure 5. Pixel rearrangement

base image. When an $n \times n$ base image and an $m \times m$ template image are given, our proposed template matching algorithm with pixel rearrangement works as follows.

**Template Matching Algorithm with Pixel Rearrangement**

Step 1. A low-resolution base image $I'$ is generated by sampling every $k$ pixels from a base image. After that, $k^2$ low-resolution template images $T'_{s,t}$ ($0 \leq s, t \leq k-1$) are generated by pixel rearrangement.

Step 2. For each $T'_{s,t}$, the similarity $R(T'_{s,t}, I')$ is computed. If the similarity is larger than a threshold value $t$, its coordinate makes a candidate position for the next step.

Step 3. The candidate positions are transformed to corresponding positions in the original base image. For each position, the similarity between original base image and the template image is computed.

Step 4. The position that has the largest similarity is output as the result.

The details of each step are shown, as follows.

**Step 1:** In this step, a low-resolution base image $I'$ is generated by sampling every $k$ pixels from an $n \times n$ base image. Since the size of $I'$ is $\frac{n}{k} \times \frac{n}{k}$, it takes $O(\frac{n^2}{k^2})$-time. After that, to obtain $k^2$ low-resolution template images $T'_{s,t}$ ($0 \leq s, t \leq k-1$), pixel rearrangement is performed such that

$$T'_{s,t}(x, y) = T(kx + s, ky + t) \quad (0 \leq x, y \leq \frac{m}{k} - 1),$$

as shown in Figure 5. The size of each $T'_{s,t}$ is $\frac{m}{k} \times \frac{m}{k}$. Since the above operation is just rearranging pixels in the base image, its computing time is $O(m^2)$.

**Step 2:** In Step 2, template matching between the low-resolution base image $I'$ and each low-resolution template image $T'_{s,t}$ is performed, that is computing similarities $R(T'_{s,t}, I')$ ($0 \leq s, t \leq k-1$). In the template matching, if the similarity is larger than a certain threshold $t$, its coordinate is stored as a candidate position for the next step. Since the sizes of each low-resolution base image and the low-resolution template image are $\frac{n}{k} \times \frac{n}{k}$ and $\frac{m}{k} \times \frac{m}{k}$, $O(\frac{n^2m^2}{k^4})$-time is necessary to perform each template matching. Since there are $k^2$ low-resolution template images, this step takes $O(\frac{n^2m^2}{k^2})$ in total.

**Step 3:** In this step, the candidate positions are transformed to corresponding positions in the original base image. We assume that the number of the candidate positions is $l$ found in Step 2 and let $p_i = (x_i, y_i)$ ($1 \leq i \leq l$) be the transformed candidate positions. For each $p_i$, template matching is performed, that is computing similarities $R(T, I[x_i, y_i])$. It takes $O(m^2)$-time to perform the template matching for each $p_i$. Therefore, the total computing time in this step is $O(lm^2)$.

**Step 4:** In Step 4, the maximum similarity position in Step 3 is output as the result. To find the maximum position from $l$ candidates, it takes $O(l)$-time.

According to the above, the total running time is $O(\frac{n^2m^2}{k^2} + lm^2)$. If $l$ is small, it is close to $O(\frac{n^2m^2}{k^2})$. We note that the above algorithm is different from the original algorithm in [16]. In the original algorithm, a base image is rearranged instead of a template image. On the other hand, in our approach, a template image is rearranged. However, the readers can easily find that there is no difference between them in essence and the time complexity is exactly the same.

## IV. FPGA Implementation

This section describes the architecture of our template matching. The main idea of our architecture is to introduce pixel rearrangement and utilize DSP slices and block RAMs in Xilinx Virtex-6 FPGA. We use Xilinx Virtex-6 family FPGA XC6VLX240T-FF1156 as the target device [8]. It consists of columns of Configurable Logic Blocks (CLBs) each of which includes two slices, programmable Input/Output Blocks (IOBs), DSP48E1 slices, and 36k-bit dual-port block RAMs.

We note that our proposed implementation performs the process in Step 2 shown in Section III. This is because in [16], the ratio of the computing time of Step 2 to the whole computing time is more than 95%. The key to shorten the whole computing time is to shorten the running time of Step 2. Therefore, in our implementation, we suppose the other processes are performed by another processor, for example, an embedded processor in the same FPGA, a host PC connected to the FPGA, or etc.

The idea of our architecture is to perform the computation of $R(T'_{s,t}, I')$ simultaneously. Figure 6 illustrates an outline of our architecture. Our architecture mainly consists of *line buffers, sum module, squared sum module, and* $k^2$ *template matching units*. In our architecture, $k^2$ low-resolution template images, and threshold value are given beforehand. After that pixels in an input low-resolution base image in raster scan order are input for each clock cycle one by one. For each subimage of size $\frac{m}{k} \times \frac{m}{k}$, the circuit performs template matching between the subimage and $k^2$ low-resolution template images. As illustrated in the figure, using one template matching unit, template matching for each low-resolution template image is performed and they work in parallel. Also, our circuit is a fully pipelined architecture. Given a pixel in raster scan order, the result of the template matching for the subimage is output after several clock cycles. In the followings, we will show the details of each module.

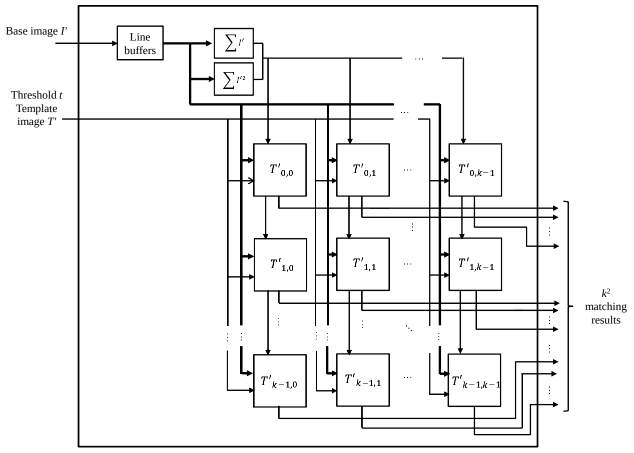


Figure 6. Outline of our architecture

## A. Line buffers

Since pixels of the low-resolution base image are input in raster scan order, it is necessary to supply $\frac{m}{k} \times \frac{m}{k}$ subimages for the pixels. To active it, we use line buffers illustrated in Figure 7. The line buffers consist of $\frac{m}{k} - 1$ FIFOs (First In First Out). Each FIFO is composed of a block RAM and stores $\frac{n}{k}$ pixels in a row of the low-resolution base image.

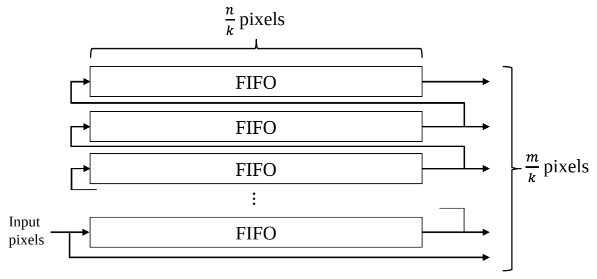


Figure 7. Line buffers with $\frac{m}{k} - 1$ FIFOs

## B. Sum module

Figure 8 illustrates the circuit that computes $\Sigma I'$. Given $\frac{m}{k}$ pixels from line buffers, the total of pixel values in $I'$ is computed using two adder trees and $\frac{m}{k}$ shift registers for each clock cycle.

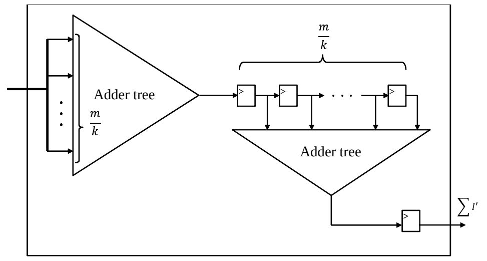


Figure 8. Sum module for $\Sigma I'$

## C. Squared-sum module

Figure 9 illustrates the circuit that computes $\Sigma I'^2$. In this module, given $\frac{m}{k}$ pixels from line buffers, each pixel value is squared. To square it, we take a look-up-table (LUT) using a block RAM. In the block RAM, the value of $x$ is stored in the address of $x$ in advance. Using the block RAM, the number of DSP slices is reduced. Using the LUT, squared values are computed. After that, the total of them is computed using two adder trees and $\frac{m}{k}$ shift registers for each clock cycle.

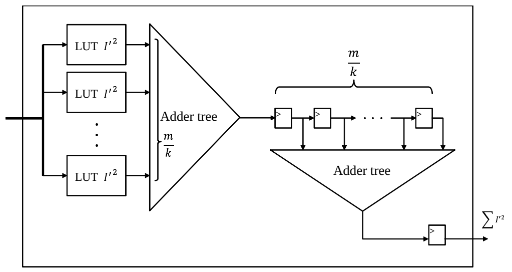


Figure 9. Squared-sum module for $\Sigma I'^2$

## D. Template matching unit

Here, we focus on the template matching unit. This unit performs the template matching between one of the $k^2$ low-resolution template image $T'_{s,t}$ and a subimage of the low-resolution base image $I'[x, y]$ of size $\frac{m}{k} \times \frac{m}{k}$. In the followings, for simplicity, let $T'$ and $I'$ denote a low-resolution template image and a subimage of the low-resolution base image, respectively, and $m' \times m'$ be the size of each, that is, $m' = \frac{m}{k}$. As shown in Section II, in the template matching between $T'$ and $I'$, the normalized coefficient correlation in Eq. (1) is computed as

$$R(T', I') = \frac{\sum (T' - \overline{T'})(I' - \overline{I'})}{\sqrt{\sum (T' - \overline{T'}) \sum (I' - \overline{I'})}}.$$

This can be represented as

$$R(T', I') = \frac{m'^2 \sum I'T' - \sum I' \sum T'}{\sqrt{(m'^2 \sum I'^2 - (\sum I')^2)(m'^2 \sum T'^2 - (\sum T')^2)}}.$$

In the template matching, if the similarity is larger or equal to a certain threshold value $t$, that is $R(T', I') \geq t$, the input two images are matched. Squaring the both sides and transforming it, we have the matching conditions $(m'^2 \sum I'T' - \sum I' \sum T')^2 \leq t^2(m'^2 \sum T'^2 - (\sum T')^2)(m'^2 \sum I'^2 - (\sum I')^2)$ and $(m'^2 \sum I'T' - \sum I' \sum T')^2 \geq 0$. Instead of the computation of the normalized correlation coefficient directly, we use these inequalities without the square root and division computations.

In the template matching module, since values $m'^2$, $\sum T'$ and $t^2(m'^2 \sum T'^2 - (\sum T')^2)$ in the conditions only depend on the template image and the threshold value, they can be computed beforehand. Therefore, before template matching is performed, they are input to the registers in the circuit in advance. The template matching module computes the other terms that depend on the low-resolution base image in the above equations and evaluates the similarity using them.

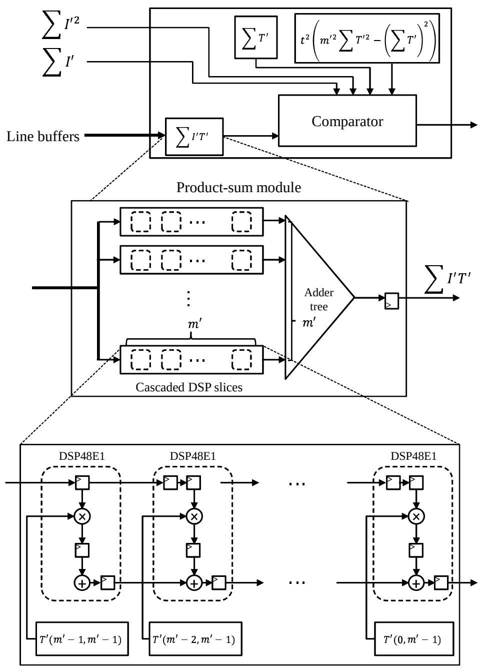


Figure 10. The template matching unit

**Product-sum module:** The product-sum module computes $\sum I'T'$ using DSP slices. The computation is similar to the 2-dimensional convolution. As illustrated in Figure 10, to perform the computation, the product-sums for each row-direction are computed using $m'$ multipliers. In our FPGA implementation, we use one column of $m'$ cascaded DSP slices for each. Therefore, in total, $m'$ columns of $m'$ cascaded DSP slices, that is, $m'^2$ multipliers are utilized. After that the resulting product-sum is computed by summing them with the adder tree.

**Comparator module:** The comparator module computes the resulting template matching between $T'$ and $I'$ by evaluating the matching conditions in the matching conditions using the results of the above circuit and precomputed values. To make the circuit simple, we assume that $m'$ is a power of 2. By the assumption, the products of $m'$ can be computed by bit-shift operation without multipliers. In this module, 4 multipliers are necessary to evaluate the matching conditions.

## V. Performance Evaluation and Experimental Results

Let us evaluate the performance of our architecture of the template matching. As shown in Section IV, our proposed circuit performs the template matching in Step 2 in Section III. Therefore, in this section, we focus on Step 2 and evaluate its performance. Again, let $n \times n$, $m \times m$, and $k$ denote the sizes of a base image, a template image, and an interval of sampling, respectively. The sizes of the low-resolution base and template images are $\frac{n}{k} \times \frac{n}{k}$ and $\frac{m}{k} \times \frac{m}{k}$, respectively. Recall that our circuit works in fully pipelined manner. Let $L$ denote a latency of the circuit, that is, after an input pixel is given $L$ clock cycles are necessary to output the corresponding result. The total clock cycles to perform the template matching is $\frac{n^2}{k^2} + L$. In our template matching implementation, we use $k^2$ template matching units for $k^2$ low-resolution template images. These $k^2$ units work in parallel and each of them can work in the fully pipelined architecture. Since we use $m'^2 + 4 = (\frac{m}{k})^2 + 4$ multipliers in each template matching unit, $m^2 + 4k^2$ multipliers are necessary in total.

We have evaluated the performance of our template matching circuit using the Xilinx Virtex-6 family FPGA XC6VLX240T-FF1156. Table I summarizes the evaluation results for our template matching circuit using ISE Foundation 13.1. The implemented circuit can perform the template matching for $n = 1024$, $m = 16$, and $k = 4$. Namely, the sizes of a base image and a template image are $1024 \times 1024$ and $16 \times 16$, respectively. Also, the sizes of the low-resolution base and template images are $256 \times 256$ and $4 \times 4$, respectively. The readers may think that the size of the template image is too small. However, by sampling the base image and the template image beforehand or using a part of the template image as a template image, we can perform the template matching for larger template images [20]. The latency $L$ in our implemented circuit is 15 clock cycles. Therefore, the computing time is $\frac{1024^2}{4^2} + 15 = 65551$ clock cycles, that is, 234.107 [$\mu$s].

Table I
Performance evaluation

| | |
|---|---|
| DSP48E1 slices (out of 768) | 352 |
| 36k-bit block RAMs (out of 416) | 3 |
| CLBs (out of 18840) | 455 |
| Clock frequency [MHz] | 280.004 |

For the purpose of estimating the speed up of our FPGA implementation, we have also implemented a conventional software approach of the template matching using GNU C. We have used Intel Core i7 860 running 2.93GHz and 8GB memory. Also, we have implemented a software approach with the GPU support shown in [16]. We have used NVIDIA GTX 580 with 512 processing cores running 1.54GHz and 3GB memory. Table II shows the comparison of the computing time between them. Our FPGA implementation achieved 3.66 and 348.88 times speed-up over the GPU and the conventional software implementations, respectively.

Table II
Comparison of the computing time

| | Computing time[ms] | Speed-up |
|---|---|---|
| This work | 0.234 | — |
| GPU [16] | 0.857 | 3.66 |
| CPU | 81.676 | 348.88 |

## VI. Conclusion

In this paper, we have presented an FPGA architecture for template matching using DSP slices. The template matching algorithm in our architecture is based on the pixel rearrangement method. In our circuit, multiple matching modules that compute the similarity and work in parallel are used. We have implemented it in the Xilinx Virtex-6 FPGA. The experimental result shows that the computing time of our FPGA implementation is approximately 348.88 and 3.66 times faster than that of CPU and GPU implementations, respectively.

## References

[1] J. L. Bordim, Y. Ito, and K. Nakano, "Accelerating the CKY parsing using FPGAs," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 5, pp. 803–810, May 2003.

[2] ——, "Instance-specific solutions to accelerate the CKY parsing for large context-free grammars," *International Journal on Foundations of Computer Science*, pp. 403–416, 2004.

[3] Y. Ito and K. Nakano, "Efficient exhaustive verification of the Collatz conjecture using DSP blocks of Xilinx FPGAs," *International Journal of Networking and Computing*, vol. 1, no. 1, pp. 19–62, 2011.

[4] Y. Ito, K. Nakano, and S. Bo, "The parallel FDFM processor core approach for CRT-based RSA decryption," *International Journal of Networking and Computing*, vol. 2, no. 1, pp. 79–96, 2012.

[5] K. Nakano and E. Takamichi, "An image retrieval system using FPGAs," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 5, pp. 811–818, May 2003.

[6] K. Nakano and Y. Yamagishi, "Hardware n choose k counters with applications to the partial exhaustive search," *IEICE Trans. on Information & Systems*, 2005.

[7] Y. Ago, Y. Ito, and K. Nakano, "An FPGA implementation for neural networks with the FDFM processor core approach," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 28, no. 4, pp. 308–320, 2013.

[8] Xilinx Inc., *Virtex-6 Family Overview*, 2010.

[9] S. Yoshimura and T. Kanade, "Fast template matching based on the normalized correlation by using multiresolution eigenimages," in *Proceedings of International Conference on Intelligent Robots and Systems*, 1994, pp. 2086–2093.

[10] Y. Abe, M. Shikano, T. Fukuda, F. Arai, and Y. Tanaka, "Vision based navigation system by variable template matching for autonomous mobile robot," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, 1998, pp. 952–957.

[11] D. Rainsford and W. Mackaness, "Template matching in support of generalisation of rural buildings," in *Proceedings of International Symposium on Spatial Data Handling*, 2002, pp. 137–152.

[12] B. Zitová and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[13] J. Rodgers and W. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[14] A. Rosenfeld and G. Vanderbrug, "Coarse-fine template matching," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, no. 2, pp. 104–107, 1977.

[15] S. Tanimoto, "Template matching in pyramids," *Computer Graphics and Image Processing*, vol. 16, no. 4, pp. 356–369, 1981.

[16] A. Uchida, Y. Ito, and K. Nakano, "Fast and accurate template matching using pixel rearrangement on the GPU," in *Proceedings of International Conference on Networking and Computing*, 2011, pp. 153–159.

[17] E. H. Adelson and P. J. Burt, "Image data compression with the laplacian pyramid," in *Proceedings of the Conference on Pattern Recognition and Image Processing*, 1981, pp. 218–223.

[18] G. Bonmassar and E. L. Schwartz, "Improved cross-correlation for template matching on the Laplacian pyramid," *Pattern recognition letters*, vol. 19, no. 8, pp. 765–770, 1998.

[19] A. C. Berg and J. Malik, "Geometric blur for template matching," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, pp. 607–614.

[20] D. Keysers, T. Deselaers, and T. Breuel, "Optimal geometric matching for patch-based object detection," *Electronic Letters on Computer Vision and Image Analysis*, vol. 6, no. 1, pp. 44–54, 2007.

[21] S. Ludwig, "Implementation of a spatio-temporal Laplacian image pyramid on the GPU," Ph.D. dissertation, Universität zu Lübeck, Feburary 2008.

[22] H. Cho and T. Park, "Wavelet transform based image template matching for automatic component inspection," *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 9–12, pp. 1033–1039, 2010.

[23] R. Cabido, A. S. Montemayor, and Á. Sánchez, "Hardware-accelerated template matching," in *Proceedings of Iberian Conference on Pattern Recognition and Image Analysis*, 2005, pp. 691–698.

[24] R. F. Anderson, J. S. Kirtzic, and O. Daescu, "Applying parallel design techniques to template matching with GPUs," in *Proceedings of IEEE VECPAR 2010*, 2010.

[25] N. A. Vandal and M. Savvides, "CUDA accelerated iris template matching on graphics processing units (GPUs)," in *Proceedings of Fourth IEEE International Conference on Biometrics: Theory Applications and Systems*, 2010, pp. 1–7.

[26] Y. Ren, J. Zhu, X. Yang, and S. Ye, "The application of Virtex-II Pro FPGA in high-speed image processing technology of robot vision sensor," *Journal of Physics: Conference Series*, vol. 48, pp. 373–378, 2006.