An Efficient Implementation of the One-Dimensional Hough Transform Algorithm for Circle Detection on the FPGA

Xin Zhou, Yasuaki Ito, and Koji Nakano
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527 JAPAN

***Abstract*—The main contribution of this paper is to present an efficient implementation of the Hough transform algorithm that uses only one-dimensional parameter spaces for circles detection on a Xilinx Virtex-7 FPGA. We implemented the circuit using 398 DSP48E1 slices and 309 block RAMs with 18Kbits. The experimental results show that the architecture runs in 181.812MHz. For an edge image of size 400×400, our circuit can perform in at most 970434 clock cycles, i.e., 5337.568μs. Our implementation attains a speed-up factor of approximately 189 over the sequential implementation on the CPU.**

***Keywords*-Circles detection, Hough transform, FPGA, Embedded DSP slices, Embedded block RAMs, Edge image**

I. Introduction

The Hough transform is a technique to find shapes in images [1]. The Hough transform defines a mapping from an image into a parameter space represented by an accumulate array. Let us consider circle detection using the Hough transform. A circle can be defined by the three parameters, its center coordinate (x_c, y_c) and the radius r. Therefore, $O(N^3)$ space is necessary to store the parameter space, where N is the size of each dimension of the parameter space. Moreover, it takes $O(N^3)$-time to vote for each edge point and search intensive elements in the accumulate array.

Recent FPGAs (Field Programmable Gate Arrays) have embedded DSP48E1 slices and block RAMs. The DSP slices are equipped with a multiplier, adders, logic operators, etc [2]. The block RAM is an embedded memory supporting synchronized read and write operations, and can be configured as a 36Kbit or two 18Kbit dual port RAMs [3]. The key technique for accelerating the algorithm is an efficient usage of DSP slices and block RAMs. However, in the conventional Hough transform algorithm for circles detection, even the state-of-the-art FPGA such as the Xilinx Virtex-7 series FPGAs cannot handle the $O(N^3)$ space without off-chip memories.

The parameter space decomposition is used to reduce the parameter space. Many of methods based on the Hough transform that use two-dimensional parameter spaces [4], [5] and one-dimensional parameter spaces [6] have been proposed. Specifically, in the one-dimensional Hough transform algorithm [6], the x-coordinate of center, y-coordinate of center, and radius are detected in series. In each detection, one-dimensional parameter spaces is used in the same way as the Hough transform. Moreover, various hardware algorithms for circle detection have been proposed. These existing researches use the template matching [7], [8] and the Hough transform algorithms [9], [10], [11]. Shafer *et al.* proposed an FPGA implementation to detect the iris position [7]. However, it detects only one circle in an image. Jen *et al.* proposed an FPGA implementation to detect circles using any three nonlinear pixels to form a circle [9]. However, because of the huge size of parameter spaces, this method uses off-chip memories. Elhossini *et al.* proposed a pipelined FPGA architecture for circles detection [11]. Four specific radii are fixed due to the limitations of on-chip memories on the FPGA.

The main contribution of this paper is to present an efficient FPGA implementation of the one-dimensional Hough transform algorithm for circles detection [6]. Our ideas include:

One-Dimensional Parameter Spaces: Since the circle detection algorithm of our implementation requires only one-dimensional parameter spaces, the memory storage for the parameter space can be implemented only with block RAMs, Therefore, any additional off-chip memory is not necessary.
Voting Space Partitioning: The parameter spaces for x- and y-coordinates of center candidates are partitioned into multiple block RAMs that are voted in parallel. Also, the voting operations of radius for each center candidate is also concurrently performed using multiple block RAMs.
Efficient Usage of DSP slices: DSP slices are used to merge the partitioned voting spaces for x- and y-coordinates of center candidates in a pipelined fashion. Furthermore, DSP slices are used to compute the Euclidean distance between each center candidate and edge points.

The one-dimensional Hough transform algorithm consists of the following four steps: (1) x-coordinates of center candidates of circles are detected by voting midpoints of every two edge points in each row. (2) y-coordinates of center candidates of circles are detected by voting midpoints of every two edge points in each column. (3) Center candidates are listed from x- and y-coordinates of center candidates. (4) For each center candidate, radii of the circles are detected. Also, detected circle candidates are checked whether the candidate is a true circle by voting the Euclidean distances

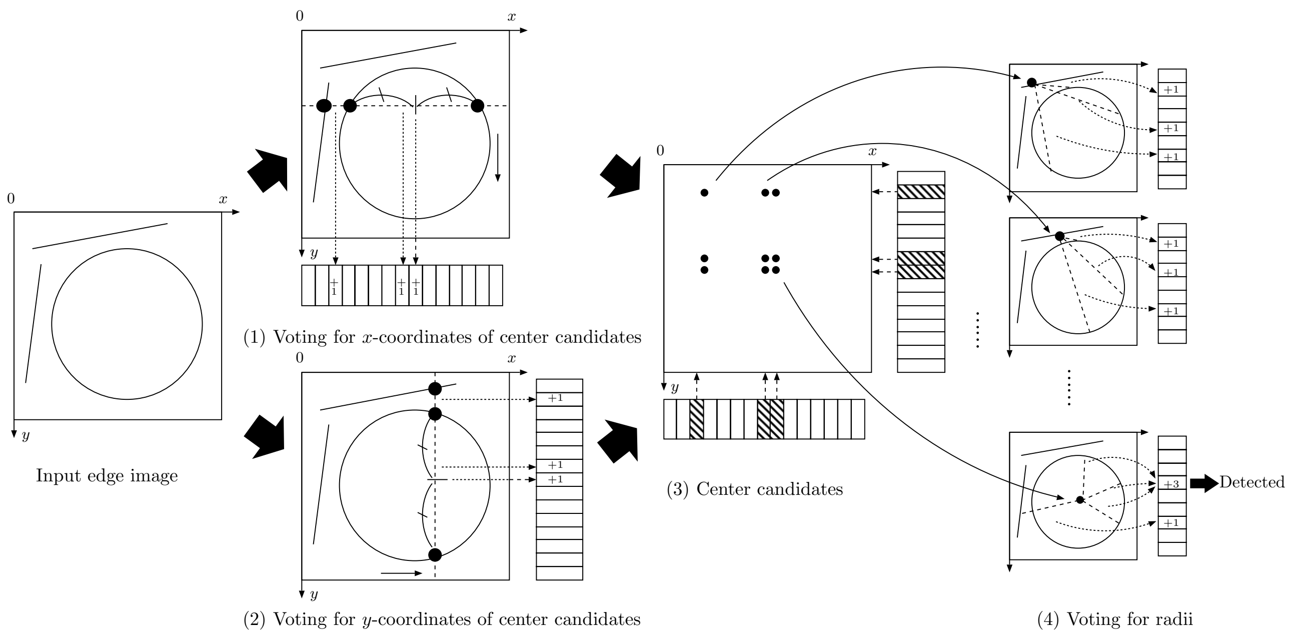
0
x
y
Input edge image
(1) Voting for x-coordinates of center candidates
(2) Voting for y-coordinates of center candidates
(3) Center candidates
(4) Voting for radii
+1
+3
Detected

Figure 2. The outline of the one-dimensional Hough transform algorithm for circle detection

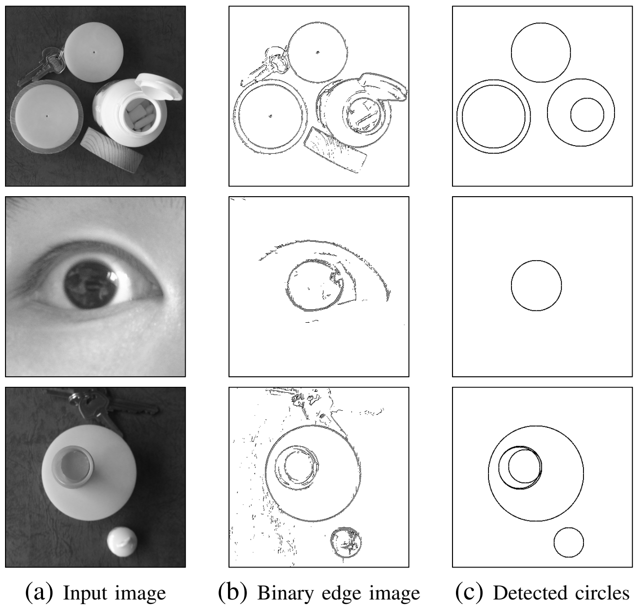

Figure 1. Example of circles detection using the one-dimensional Hough transform algorithm

between each center candidate and every edge point.

Figure 1 shows an example of circles detection using this method. For an input image (Figure 1(a)), its edge image (Figure 1(b)) is obtained using the edge detector such as Canny edge detector [12]. Figure 1(c) draws detected circles using the one-dimensional Hough transform algorithm.

We have implemented the one-dimensional Hough transform algorithm on a Xilinx Virtex-7 XC7VX485T-2. Our new architecture uses 398 DSP48E1 slices and 309 block RAMs with 18Kbits. Our proposed circuit runs in 181.812MHz. For a binary image of size 400×400, our circuit performs the circles detection in at most 970434 clock cycles, i.e., $5337.568\mu s$. Our implementation attains a speed-up factor of approximately 189 over the sequential implementation on the CPU.

This paper is organized as follows. Section II reviews the conventional three-dimensional Hough transform algorithm and introduces our proposed one-dimensional Hough transform algorithm. We show the FPGA architecture for the proposed algorithm in Section III. Section IV shows the experimental results. Finally, Section V concludes the paper.

II. One-dimensional Hough Transform algorithm for circle detection

The main purpose of this section is to show the one-dimensional Hough transform algorithm for circles detection. Figure 2 illustrates an outline of this algorithm. The detail of this algorithm is shown as follows.

Step 1: We compute midpoints of every two edge points on each row. After that, the x-coordinates are voted to a one-dimensional accumulate array. Namely, the element that corresponds to the x-coordinate of each midpoint is incremented by one. This operation is performed for every row in the input image. If there is a circle, the voting to the x-coordinate of its center is concentrated since a circle

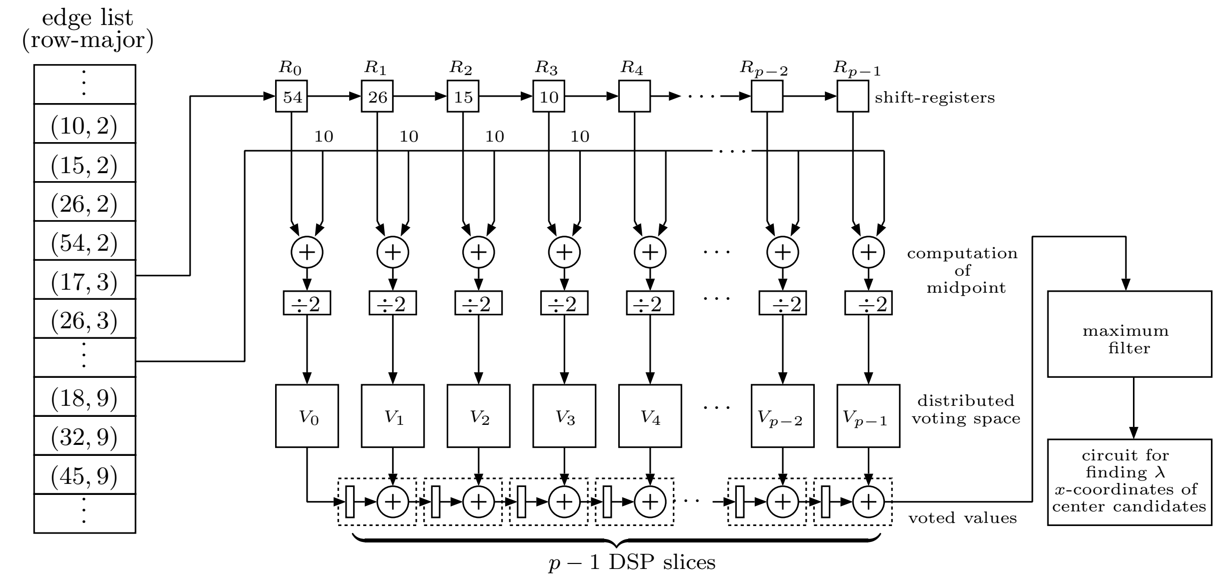

Figure 3. Architecture of voting for x-coordinates of center candidates

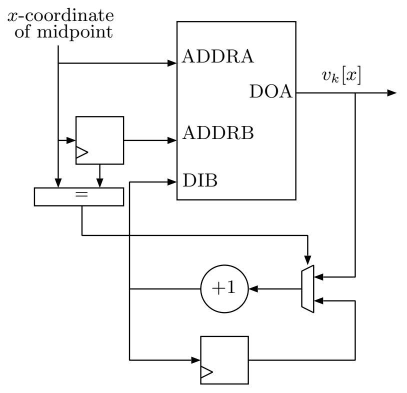

Figure 4. A block RAM to store the voted values

is symmetrical to the vertical bisector through its center. To cope with quantization error, after voting, we use a maximum filter for the voted values. In here, for each value in the accumulate array, this filter copies the value if it is the maximum within a local range, otherwise, the filter outputs zero. After that, the top λ largest elements are extracted as x-coordinates of center candidates of circles to detect multiple circles.

Step 2: In the same way as Step 1, we compute midpoints of every two edge points on each column. After that, the y-coordinates are voted to a one-dimensional accumulate array for every column. If there is a circle, the voting to the y-coordinate of its center is concentrated since a circle is symmetrical to the horizontal bisector through its center. The voted values are filtered by the maximum filter. After filtering, the top λ largest elements are obtained as y-coordinates of center candidates of circles.

In the following FPGA implementation, we perform Steps 1 and 2 in parallel since these steps can be executed independently.

Step 3: We list center candidates that are all combinations from x- and y-coordinates obtained in the above steps. Since each step obtains λ coordinates, λ^2 center candidates are listed in total.

Step 4: For each center candidate, the Euclidean distances between the center candidate and all edge points are computed and the distances are voted to a one-dimensional accumulate array as radii of circles. If a center candidate is a center of true circle in the image, an element that corresponds to the radius is intensely voted. Therefore, we verify whether the center candidate and each radius represent a true circle using the voted value. In digital images, it is known that the number of pixels on the circumference of circles with radius r is $4\sqrt{2}r$ [13]. However, practical circles may be broken or disturbed. Therefore, for each radius, we determine whether the center candidate and its radius represent a true circle using the threshold $f \times 4\sqrt{2}r$, where f is a threshold factor that is a constant value within the range $(0, 1]$. If the voted value of radius is larger than the threshold, the circle is verified as a true circle. The validation for each radius, circles with the center candidate are extracted.

For each center candidate, the above operation is performed. In our FPGA implementation, this operation is performed in parallel and circles are detected.

III. Our FPGA architecture for the one-dimensional Hough transform

This section describes our FPGA architecture of the one-dimensional Hough transform algorithm. The input data for our implementation is given as two edge lists consisting of coordinates of edge pixels in row- and column-major order, respectively. These edge lists can be obtained by edge detection easily, and each list is stored into a block RAM.

A. Structure for voting operation of center candidates

Figure 3 shows our architecture for the voting operation of x-coordinates of center candidates, where p is the number of voting modules. Namely, up to p voting operations are concurrently performed. We utilize one series of p shift-registers that transfer data in the left-to-right direction. In order to compute the midpoints of any two edge points on each row, we read them from the block RAM that stores the edge list in row-major order. To give all pairs of x-coordinates of which y-coordinates are identical for each row, we input x-coordinates which have the same y-coordinate to the register and transferred them with shift-registers one by one. If all or p x-coordinates which have the same y-coordinate are transferred to the registers, then all x-coordinates which have the same y-coordinate are continuously read out from the block RAM and broadcast to pair with the x-coordinates in the registers. If the number of edge points whose y-coordinates are identical is larger than p, the above operation is repeated. For each given pair of x-coordinates, the coordinate of the midpoint is computed using an adder and a 1-bit right-shifter that divides by two.

After that, the x-coordinates of midpoints are voted to the block RAM. Namely, an element that corresponds to the x-coordinate in the block RAM is incremented by one. We use p block RAMs V_k $(0 \leq k \leq p-1)$ to store the voted values and at most p midpoints are concurrently voted. Figure 4 illustrates the architecture of V_k using a block RAM. The block RAM is utilized in dual port mode, where port set A and B are operated for read and write operation, respectively.

In the block RAMs on the target device of this work, read/write operations can be configured as either RF (Read First) mode or WF (Write First) mode. In the dual port mode, there is a restriction that if read and write operation to the same address are performed for each port, the setting of block RAMs must be RF [3]. In the RF mode, if reading and writing operations are performed to the same address, reading operation is performed before the writing operation.

We use the block RAM to store the values of $v_k[x]$ $(0 \leq x < n)$, where the size of image is $n \times n$ and x is an x-coordinate of midpoints. Let $v_k[x]$ denote a data of address x in the block RAM V_k. Since x is given to its $ADDRA$, $v_k[x]$ is output from DOA after the rising clock edge as illustrated before. After that, $v_k[x]+1$ is computed and it is given to DOB. Since x is given to $ADDRB$, $v_k[x]+1$ is written in $v_k[x]$. According to the restriction stated in the above, since the same value of x may be input continuously, we used an additional register to store the latest voted value and if the same value of x is input continuously, the stored value is used instead of the value read from the block RAM.

After the voting operations, we combine the voted values stored in p block RAMs. We read every element in each block RAM one by one. These values are added and transferred left-to-right for each clock cycle to compute the sum of each element with $p-1$ registers and $p-1$ adders. To optimize the circuit resources, we use a cascaded DSP slice for each pair of register and adder.

B. Structure for finding the center candidates of circles

A maximum filter is used to cope with the quantization error for the voted values above. After filtering, the x-coordinates to which the λ largest values correspond are obtained as x-coordinates of center candidates of circles. Each voted value is input to maximum filter for each clock cycle. For each value, the filter verifies whether it is the maximum comparing with its neighboring 2 values. If it is the maximum in the local range, the filter copies the largest value, otherwise, the filter outputs it as zero. Therefore, the largest value in the local range and zero are alternately output after filtering.

Figure 5 illustrates the structure that finds the x-coordinates of center candidates. An array of registers and comparators are used to obtain the largest λ values. Every register is initialized to zero. The filtered values are continuously input to the left-most register for each clock cycle. Each register of c_i $(0 \leq i < \lambda)$ compares the value with its left register. If this register has smaller value, the value of its left register is then transferred to it. If this register has larger value, it will compare the value with its right register, and if the register has larger value, this value is transferred to its right neighboring register. All the values of registers are transferred in parallel. Since input values are obtained through the maximum filter, the input values are given at more than one clock cycle intervals. Hence, the larger values will be gradually transferred to the right side through the registers. Finally, the top λ largest values are stored in the registers. Namely, the x-coordinates that have the top λ largest values are chosen to be the x-coordinates of center candidates of circles.

Similarly, using another circuit whose structure is the same as the above, the voting operation for y-coordinates of center candidates are performed with edge lists stored in column-major order as input. Also, y-coordinates with the top λ largest values are obtained as the y-coordinates of center candidates. Every x- and y-coordinates of centers candidates are combined to construct λ^2 center candidates.

C. Structure for the voting operation of radius

The voting operation of radius is performed for each center candidate in parallel (Figure 6). Since λ^2 center

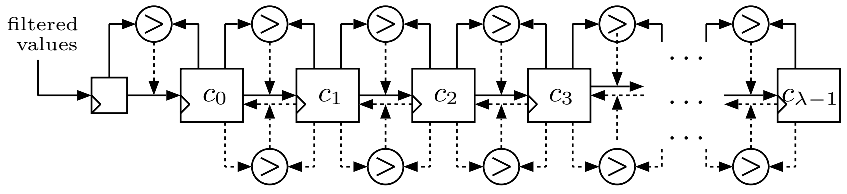

Figure 5. Architecture for finding λ x-coordinates of center candidates

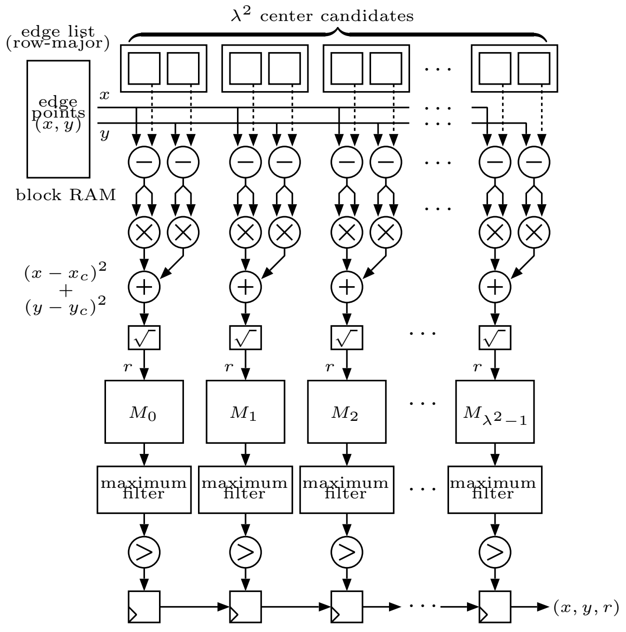

Figure 6. Architecture of voting for radius

candidates are obtained, we use λ^2 voting spaces to store the voted values. For each center candidate, the Euclidean distances between the center candidate and all edge points are computed. The circuit runs in a pipeline fashion for coordinates of edge points that are given for each clock cycle. The computation of the Euclidean distances include the computation of the square root. Since it is difficult to compute them directly on the circuit, we use the CORDIC IP provided by Xilinx that provides a hardware module that is fully pipelined architecture [14]. Since the computed distances are voted to a block RAM for each center candidate, totally, λ^2 block RAMs M_k $(0 \leq k < \lambda^2)$ are used. The architecture of M_k is the same as shown in Figure 4.

D. Structure for verifying true circles

Finally, we verify whether the center candidates are true circles. If a center candidate is that of true circle in the image, an element of the block RAM that corresponds to the radius is intensely voted. A maximum filter is also used to cope with the quantization error for the voted values. If the value is the maximum in a local range, the filter copies the largest value, otherwise, the filter output it as zero. After filtering, each radius is verified whether it is the radius of a true circle by comparing its voted value with the threshold $f \times 4\sqrt{2}r$ (Section II). All the values of $f \times 4\sqrt{2}r$ are precomputed to store in a block RAM that is used as a Look-up-table. If it is larger than the threshold, the center candidate and the radius that represent a circle is input to the shift registers and output through the registers.

IV. Performance Evaluation And Experimental Results

We have implemented the proposed architecture for circles detection and evaluated it on the Xilinx Virtex-7 FPGA XC7VX485T-2 [15]. For our implementation, 398 DSP48E1 slices, 309 block RAMs with 18Kbit and 20452 slices of the FPGA are used. The FPGA with the architecture proposed in this paper works in 181.812MHz.

In our implementation, the x- and y-coordinates of edge list stored in row- or column-major are 9-bit integer. The voted values of x- and y-coordinate of center candidates are set to be 17-bit integer. The number p of voting modules for x- and y-coordinates of center candidates is set to be 100. The number λ of x- or y-coordinates of center candidates is set to be 10, therefore, 100 center candidates are constructed. The data format of the voted values for radius is 13-bit integer.

Since the latency of our architecture depends on the input image, we suppose that all pixels of input image of size $n \times n$ are edge points. Let p be the number of voting modules for x- or y-coordinates of center candidates and λ be the number of x- or y-coordinates of center candidates. For simplicity, we assume that n is a multiple of p. For a certain row or column i, it takes $\frac{n}{p}(p + n + 6)$ clock cycles to complete the voting operations of row or column i. Therefore, it takes $\frac{n^2}{p}(p+n+6)$ clock cycles to complete the voting operations for all rows or columns. After that, it takes $p + n + \lambda + 4$ clock cycles to find x- or y-coordinate of center candidates. The combination of every x- and y-coordinates of center candidates takes 1 clock cycle. The voting operation for radius takes $n^2 + 11$ clock cycles, and it takes $\frac{n}{2} + \lambda^2 + 8$ clock cycles to output all detected true circles. Finally, our circuit totally takes $\frac{n^3}{p} + \frac{(2p+6)n^2}{p} + \frac{3}{2}n + p + \lambda^2 + \lambda + 24$ clock cycles to implement the one-dimensional Hough transform.

For estimating the speed up of our FPGA implementation, we have also implemented a software approach of the one-dimensional Hough transform using GNU C. We have used Intel Xeon X7460 running in 2.66GHz and 128GB memory to run the sequential one-dimensional Hough transform algorithm. For the image above, the software implementation can perform the one-dimensional Hough transform for circles detection in $1008.658ms$. On the other hand, our circuit can perform it in 970434 clock cycles, i.e., $5337.568\mu s$. Therefore, our FPGA implementation attains a speed-up factor of approximately 189 over the sequential implementation on the CPU.

There are a number of literatures reported to implement circles detection using the FPGA shown in Table I, where Int. means internal (on-chip) and Ext. means external (off-chip). It is difficult to directly compare to other works because utilized FPGAs and supported size of images differ. Considering the throughput, our implementation compares favorably with other works. The deficiencies of these existing researches such as detecting only one circle or using off-chip memories are not existing in our implementation. Our implementation detects multiple circles with variable radii using only the block RAMs on the FPGA.

Table I
COMPARISON WITH RELATED WORKS FOR HOUGH TRANSFORM

	Shafer [7]	Tokunaga [8]	Jen [9]
Base algorithm	Template matching	Template matching	Hough transform
Device	Altera EP4SGX530	Xilinx XC4025E	Altera Stratix 1S25
Memory	Int. 6.75Mbit	—	Int. 1.6Mbit & Ext
Frequency	159MHz	—	—
Throughput	9.362Mpixel/s	0.0512Mpixel/s	0.01524Mpixel/s

	Geninatti [10]	Elhossini [11]	This work
Base algorithm	Hough transform	Template matching	Hough transform
Device	Xilinx Spartan 3	Xilinx Virtex-4	Xlinix XC7VX485T-2
Memory	Ext. 1Mbit	Int. 256Kbit	Int. 5.4Mbit
Frequency	—	27MHz	181.812MHz
Throughput	12.32Mpixel/s	14.4Mpixel/s	29.976Mpixel/s

V. Conclusions

We have presented an efficient implementation of the one-dimensional Hough transform using 398 DSP slices, 309 block RAMs with 18Kbits on the Virtex-7 Family FPGA XC7VX485T-2. The architecture runs in 181.812MHz and for an image of size 400×400 that all pixels are edge points, our circuit performs the one-dimensional Hough transform in 970434 clock cycles, i.e., $5337.568\mu s$ which theoretically attains a speed-up factor of approximately 189 over the sequential implementation on the CPU.

References

[1] P. V. C. Hough, "Method and means for recognizing complex patterns," U.S. Patent 3,069,654, 1962.

[2] Xilinx Inc., *7 Series FPGAs DSP48E1 Slice(v1.6)*, 2013.

[3] ——, *Virtex-7 Series FPGAs Memory Resources (v1.10)*, 2014.

[4] D. Ioannou, W. Huda, and A. F. Laine, "Circle recognition through a 2D Hough transform and radius histogramming," *Image and vision computing*, vol. 17, no. 1, pp. 15–26, 1999.

[5] H.-S. Kim and J.-H. Kim, "A two-step circle detection algorithm from the intersecting chords," *Pattern recognition letters*, vol. 22, no. 6, pp. 787–798, 2001.

[6] A. Goneid, S. El-Gindi, and A. Sewisy, "A method for the Hough transform detection of circles and ellipses using a 1-dimensional array," in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, vol. 4. IEEE, 1997, pp. 3154–3157.

[7] J. L. Shafer, H. Ngo, and R. W. Ives, "Using an FPGA to accelerate pupil isolation in iris recognition," in *Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on*. IEEE, 2010, pp. 1774–1777.

[8] Y. Tokunaga and T. Inoue, "A method for circular pattern recognition in a binary image and its implementation onto an FPGA," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 82, no. 2, pp. 246–254, 1999.

[9] J. R. Jen, M. C. Shie, and C. Chen, "A circular Hough transform hardware for industrial circle detection applications," in *Industrial Electronics and Applications, 2006 1ST IEEE Conference on*. IEEE, 2006, pp. 1–6.

[10] S. R. Geninatti, J. I. B. Benítez, M. Calvio, N. G. Mata, and J. G. Luna, "FPGA implementation of the generalized Hough transform," in *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*. IEEE, 2009, pp. 172–177.

[11] A. Elhossini and M. Moussa, "Memory efficient FPGA implementation of Hough transform for line and circle detection," in *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*. IEEE, 2012, pp. 1–5.

[12] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[13] Z. Kulpa, "On the properties of discrete circles, rings, and disks," *Computer graphics and image processing*, vol. 10, no. 4, pp. 348–365, 1979.

[14] Xilinx Inc., *LogiCORE IP CORDIC v6.0*, 2013.

[15] ——, *7 Series FPGAs Configuration User Guide*, 2013.