An Efficient Implementation of the Gradient-based Hough Transform using DSP slices and block RAMs 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 gradient-based Hough transform is an improvement of the original Hough transform. It is utilized to reduce substantially the computation quantity and make the detection more accurate using gradient information.**

The main contribution of this paper is to present an efficient implementation of the gradient-based Hough transform for straight lines detection using a Xilinx Virtex-7 FPGA with embedded DSP slices and block RAMs. We implemented the circuit using 13 DSP48E1 slices, 180 block RAMs with 36Kbits and 8 block RAMs with 18Kbits. The experimental results show that the architecture runs in 260.061MHz and for an $n \times n$ gray-scale image, our circuit can perform in $n^2 + (\sqrt{2}+2)n + 232$ clock cycles including the computation of gradient information.

***Keywords*-Image processing, Line detection, Hough transform, FPGA, Embedded DSP slices, Embedded block RAMs**

I. Introduction

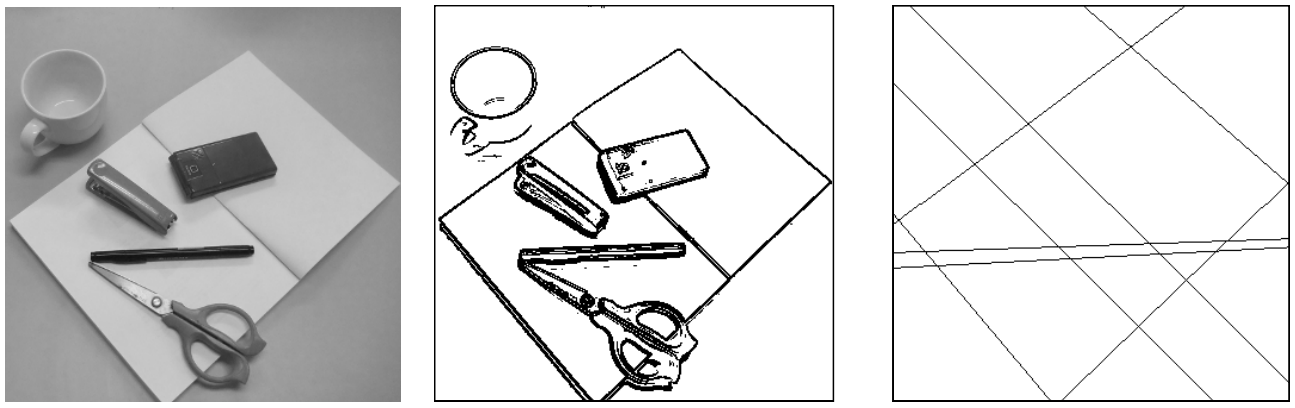
Recent FPGAs (Field Programmable Gate Arrays) have embedded DSP slices that make a higher performance and a broader application. The Xilinx Virtex-7 series FPGAs have DSP48E1 blocks that are equipped with a multiplier, adders, logic operators, etc [1]. More specifically, the DSP48E1 block has a two-input 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 produces one 48-bit two's complement production. Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput and improves frequency. The DSP48E1 also has pipeline registers between operators to reduce the delay. The block RAM in the Virtex-7 FPGA is an embedded memory supporting synchronized read and write operations. In the Virtex-7 FPGA, it can be configured as a 36Kbit dual port block RAMs, FIFOs, or two 18Kbit dual port RAMs. Since FPGA chips maintain relatively low 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. They are widely used in consumer and industrial products for accelerating processor intensive algorithms [2], [3], [4], [5], [6], [7], [8], [9], [10].

The Hough transform is a technique to find shapes in images [11]. In particular, it has been utilized to extract lines, circles, ellipses and arbitrary shapes. The Hough transform defines a mapping from an image into a parameter space represented by an accumulate array. The parameter space is defined by parameterizing detected shapes. Based on each edge point of the image, the mapping adds a vote to corresponding elements in the accumulate array. The elements that are increased represent associated parameters based on detected shapes. Therefore, the elements that are voted intensively correspond to the parameters of shapes in the image space.

The Hough transform can be used to extract straight lines in a binary image [12]. The idea of this method is to exploit the duality between points of a line and parameters of that line. A point in the image is represented by a curve in the parameter space and lines of collinear points intersect in the parameter space at one point. These intersections are counted in an array of accumulators that quantizes the parameter space appropriately. In the followings, we call this counting to the accumulators *voting*. More specifically, for each edge point (x, y) in a 2-dimensional image, the voting is performed along a curve $\rho = x\cos\theta + y\sin\theta$ $(0 \le \theta < 180)$. Possible lines can be detected by searching points that are voted intensively. Figure 1 shows an example of straight line detection using the Hough transform. For an input image (Figure 1(a)), the binary edge image (Figure 1(b)) is obtained by the edge detector such as Sobel filter. We can see that the normal Hough transform performs well basing on the pure edge image. The result of voting to the parameter space is shown in Figure 2(a). In this figure, darker points show points that are voted intensively, that is, represent probable lines. According to the result of voting, the principal lines are detected (Figure 1(c)).

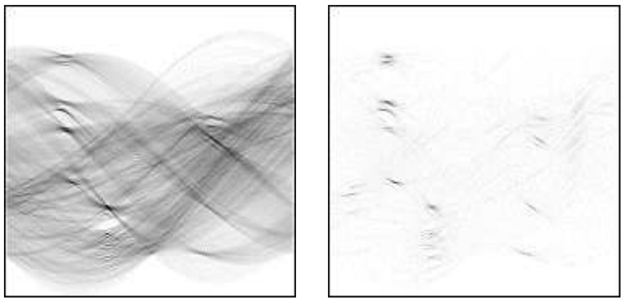
There are many improvements to the Hough transform for line detection [13]. One of the efficient improvements is using gradient information [14]. The idea of the method is to utilize gradient direction and magnitude. It is based on the fact that if a given point happens to indeed be on a line,

- The local direction of the gradient gives approximately the same direction of the actual line.
- The gradient magnitude at the pixel is higher than that



(a) Input image (b) Binary edge image by Sobel filter (c) Line detection using the Hough transform

Figure 1. Example of straight lines detection using Hough transform



(a) Conventional (b) Gradient-based

Figure 2. Hough parameter spaces of the conventional Hough transform and gradient-based Hough transform

of other points not lying on lines.

Using these ideas, we reduce the number of useless votes by limiting the range of votes with the local gradient direction, and weight voted values proportional to the local gradient magnitude to enhance the votes of pixels on lines. In the following, the original Hough transform is called *conventional Hough transform*, and the Hough transform using gradient information is called *gradient-based Hough transform* to distinguish them easily. In our implementation, we use the Sobel filter, which is used in edge detection algorithms [15] to obtain the gradient information. Figure 2 (b) shows the resulting Hough space based on the above ideas. Compared with that of the conventional Hough transform, we can see that votes are limited to the several parts that are darker points in the figure. Actually, these correspond to real lines in the image and it is easy to find that useless votes are reduced.

In this paper, we propose an FPGA implementation of the gradient-based Hough transform. Our new idea includes:

Voting Space Partitioning:
Polar coordinate voting space (θ, ρ) is partitioned and arranged into block RAMs. This enables us to perform voting operations in parallel. Also, the function of dual-port of block RAMs are fully used to accumulate the voting value instantly.

Efficient Usage of DSP slices and block RAMs:
DSP slices are used to compute $x\cos\theta$ and $y\sin\theta$ in parallel. Also, we avoid the computation of the values of $\cos\theta$ and $\sin\theta$ using block RAMs as look-up-tables. According to the above, we reduce the size of the circuit and increase its operating frequency.

Using these ideas, our new architecture for the Hough transform uses 13 DSP48E1 slices, 180 block RAMs with 36Kbits and 8 block RAMs with 18Kbits for the Hough transform. One of the most important key techniques for accelerating computation using FPGAs is an efficient usage of DSP slices and block RAMs. We have implemented our new architectures on a Virtex-7 XC7VX485T-2. Our proposed circuit runs in 260.061MHz, and the voting operations are performed for an $n \times n$ gray-scale image in $n^2 + 2n + 44$ clock cycles. Also, for the voted results, our circuit outputs the identified straight lines in $\sqrt{2}n + 188$ clock cycles. Therefore, our circuit can perform in $n^2 + (\sqrt{2} + 2)n + 232$ clock cycles, i.e., $\frac{n^2+(\sqrt{2}+2)n+232}{260.061}\mu s$. For example, for a gray-scale image of size 1000×1000, our circuit can perform straight lines detection in $3.859ms$.

This paper is organized as follows. In Section II, related works for FPGA implementation of the Hough transform for straight lines detection are shown. Section III reviews the conventional and gradient-based Hough transform algorithms for lines. We describe the FPGA architecture for the Hough transform in Section IV. Section V shows the performance evaluation and experimental results. Finally, Section VI concludes the paper.

II. Related Works

Many hardware algorithm for FPGA implementations of the Hough transform for lines have been proposed in past. In the existing researches, they introduced incremental Hough transform [16], [17], [18], CORDIC [19], [20], and hybrid-log arithmetic [21] to the computation in Hough transform. Since most of recent FPGAs produced by principal vendors equip embedded DSP slices [22], [23], [24], [25], [26], one of the most important key techniques for accelerating computation using FPGAs is an efficient usage of DSP slices and block RAMs. In our previous works [27], [28], we proposed two FPGA implementations of the conventional Hough transform with DSP slices and block RAMs. The DSP slices and block RAMs work in the fully pipelined architecture. Both two implementations are based on the conventional Hough transform. Karabernou *et al.* proposed an FPGA implementation of the gradient-based Hough transform [19]. However, they used CORDIC to compute the complicated calculations such as square roots and trigonometrical functions without DSP slices and block RAMs. As far as we know, our proposed implementation is the first FPGA implementation of the gradient-based Hough transform using DSP slices and block RAMs. Also, compared with our previous works, in this work, the number of utilized DSP slices is reduced considerably.

III. Hough Transform

The main purpose of this section is to review two Hough transform algorithms, the conventional Hough transform and the gradient-based Hough transform.

A. Conventional Hough transform

Suppose that we have an image of size $n \times n$. We assume that $n \times n$ pixels are arranged in two dimensional xy-space such that the origin is in the center of the image as illustrated in Figure 3. Hence, both coordinates x and y take integers in

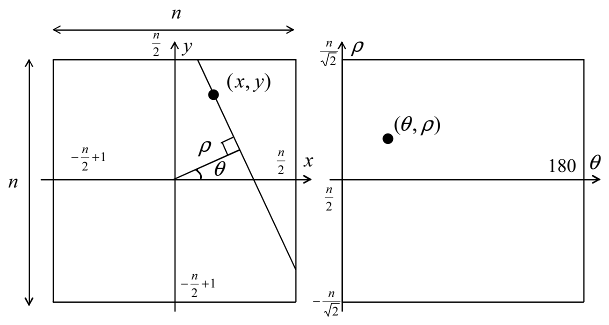


Figure 3. Two dimensional Spaces xy and $\theta\rho$ used in the Hough transform

the range $[-\frac{n}{2}+1, \frac{n}{2}]$. A pixel (x, y) $(-\frac{n}{2}+1 \leq x, y \leq \frac{n}{2})$ in the xy-space is converted to a curve in the $\theta\rho$-space by the following formula:

$$\rho \quad = \quad x\cos\theta + y\sin\theta \quad (0 \leq \theta < 180) \qquad (1)$$

Clearly, the double inequality $-\frac{n}{\sqrt{2}} < \rho \leq \frac{n}{\sqrt{2}}$ is satisfied. The values of θ and ρ can also be obtained geometrically. Suppose that we draw a line going through the origin with angle θ as illustrated in Figure 3. For such line, we can draw the orthogonal line going through a pixel (x, y). The value of ρ corresponds to the distance to the line. In other words, a point (θ, ρ) of $\theta\rho$-space corresponds to a line of xy-space.

The key idea of the Hough transform is to vote in $\theta\rho$-space only for every edge pixel in the xy-space. Let (x, y) be the pixel in xy-space, and let $p[x][y]$ be the value of the pixel. In this paper we process the image in raster scan order, the Hough transform is spelled out as follows:

[Conventional Hough Transform]

```
for $y \leftarrow -\frac{n}{2}+1$ to $\frac{n}{2}$
    for $x \leftarrow -\frac{n}{2}+1$ to $\frac{n}{2}$
        if $p[x][y] = 1$
            begin
                for $\theta \leftarrow 0$ to 179
                    $\rho \leftarrow x\cos\theta + y\sin\theta$
                    $v[\theta][\rho] \leftarrow v[\theta][\rho] + 1$
            end
```

For simplicity, we assume that the value of ρ is automatically rounded to an integer. In the conventional Hough Transform, for each point (x, y), the values of $x\cos\theta$ and $y\sin\theta$ are computed for $\theta = 0, 1, \ldots, 179$. If $v[\theta][\rho]$ is storing a large value, many edge pixels in the input pixels lie in the line in xy-space corresponds to a point (θ, ρ) in $\theta\rho$-space.

However, in many extreme cases, incremental voting operations shown in the above may cause poor performance and some error detections. For example, while the density of edge points is extremely high shown in Figure 1, some spurious lines might be detected by the conventional Hough transform. Thus, for achieving more precise lines detection, other approaches have been proposed. In the following, we describe a Hough transform algorithm based on gradient information proposed in [14]. This algorithm uses the gradient magnitude to be the weight for accumulation of the voting operations, and votes only for angles around the gradient direction.

B. Gradient-based Hough transform

In the gradient-based Hough transform, lines detection is performed for a gray-scale image, not a binary image. To obtain the gradient information for a gray-scale image, we use the Sobel filter. The Sobel filter is applied on the image for approximating the vertical and horizontal derivatives using a couple of 3×3 convolutions G_x and G_y:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \otimes I, G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \otimes I, \quad (2)$$

where I represents the input image and $\otimes$ denotes the 2-dimensional convolution operation. The two results convolved by G_x and G_y are approximations of the gradient for horizontal and vertical of the image, respectively. At each pixel in the image, the resulting gradient approximations can be combined to obtain the gradient magnitude using the formula:

$$G = \sqrt{{G_x}^2 + {G_y}^2}. \tag{3}$$

We can also compute the gradient direction θ' using

$$\theta' = \tan^{-1}(\frac{G_y}{G_x}). \tag{4}$$

Based on the gradient direction obtained by the above, we vote to the parameter space. However, there is an error between local gradient direction and the direction of actual lines due to the quantization error. Therefore, voting operation is performed not only to the angle obtained by the gradient direction, but also angles in the vicinity of it. In our implementation, we introduce weighted values w that depends on the angle as follows:

$$w(\theta - \theta') = \begin{cases} 2^{\lambda - |\theta - \theta'|} & |\theta - \theta'| \leq \lambda \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

To be suitable for the compact FPGA implementation, we use the weights as power-of-two numbers. Also, the voting range is limited to $[-\lambda, +\lambda]$ instead of the range $[0, 179]$ in the conventional Hough transform. The gradient-based Hough transform is spelled out as follows:

[Gradient-based Hough Transform]

```
for $y \leftarrow -\frac{n}{2} + 1$ to $\frac{n}{2}$
    for $x \leftarrow -\frac{n}{2} + 1$ to $\frac{n}{2}$
        Compute $G$ and $\theta'$ for $p[x][y]$
        for $\theta \leftarrow \theta' - \lambda$ to $\theta' + \lambda$ do
            begin
                if $\theta < 0$ then $\theta \leftarrow \theta + 180$
                $\rho \leftarrow x\cos\theta + y\sin\theta$
                $v[\theta][\rho] \leftarrow v[\theta][\rho] + G \cdot w(\theta - \theta')$
            end
```

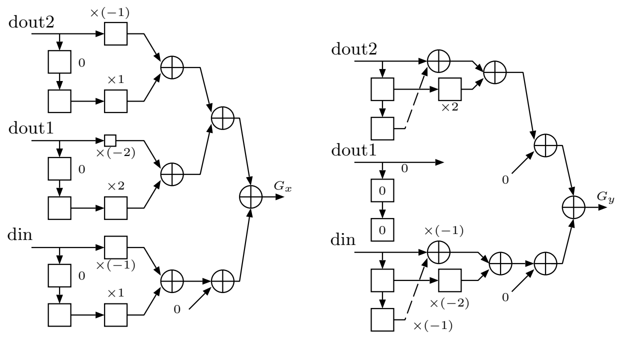
Simply speaking, the gradient-based Hough transform votes for each pixel of the gray-scale image with a weighted value $G \cdot w(\theta - \theta')$ which is proportional to the gradient magnitude. The parameter space will be sharpened by such voting operations that make the accuracy higher. Our implementation for the computation of the gradient direction and magnitude is a pipelined architecture. In the following section, we show the efficient implementations of the gradient-based Hough Transform on the FPGA.

IV. Our FPGA architecture for the Hough transform

This section describes our FPGA architecture for the gradient-based Hough transform using DSP slices and block RAMs in Xilinx Virtex-7 Family FPGA XC7VX485T-2 as the target device [26]. Figure 4 illustrates the outline of our architecture. The details are described as follows.

A. Structure for the computation of gradient information

In our architecture, we use a 3×3 Sobel filter to obtain gradient information. Since we assume that input pixels are given to the circuit in the raster scan order, we use a two-lines buffer with block RAMs to provide pixels in each 3×3 subimage to the filter. Our circuit computes the horizontal and vertical derivative approximations using combinational circuits as shown in Figure 5, where *dout2*, *dout1* and *din* represent pixel values in the three lines of the input image, respectively.



(a) G_x (b) G_y

Figure 5. Structure for the computation of G_x and G_y

As mentioned before, the algorithm needs the gradient direction and magnitude as shown in Section III. The formulae include the computation of the square root and the inverse tangent. Since it is difficult to compute them directly on the circuit, we use the CORDIC IP provided by Xilinx [29]. The CORDIC IP provides a hardware module that is fully pipelined architecture and available easily on the FPGA.

B. Structure for the computation of ρ and voting operation

Given the gradient direction θ' and magnitude G of each pixel are obtained with a pipelined architecture, the circuit computes ρ and performs voting operation. Whenever the gradient magnitude G and the gradient direction θ' of each pixel are given, the two counters for x and y increment appropriately.

We use DSP slices and block RAMs to compute $x\cos(\theta' - \lambda), \ldots, x\cos(\theta' + \lambda)$. The detail of each circuit that computes $x\cos\theta$ is shown in Figure 6. The circuit consists of one DSP slice and one block RAM. Using the block RAM as a look-up-table, $\cos\theta$ is computed and the DSP slice computes the product of x and $\cos\theta$. We note that in our implementation, since two circuits can share the two block RAMs for the look-up-table with the dual port, we use $2\lambda + 1$ DSP slices and $\lceil \frac{2\lambda+1}{2} \rceil$ block RAMs to compute $x\cos(\theta' - \lambda), \ldots, x\cos(\theta' + \lambda)$. For simplicity, the sharing is omitted in Figure 4 though it seems that every circuit that computes $x\cos\theta$ has one block RAM.

Also, to compute $y\sin\theta$ $(1 \leq \theta \leq 179)$, we use the fact that since pixel data are input in raster scan order, the value

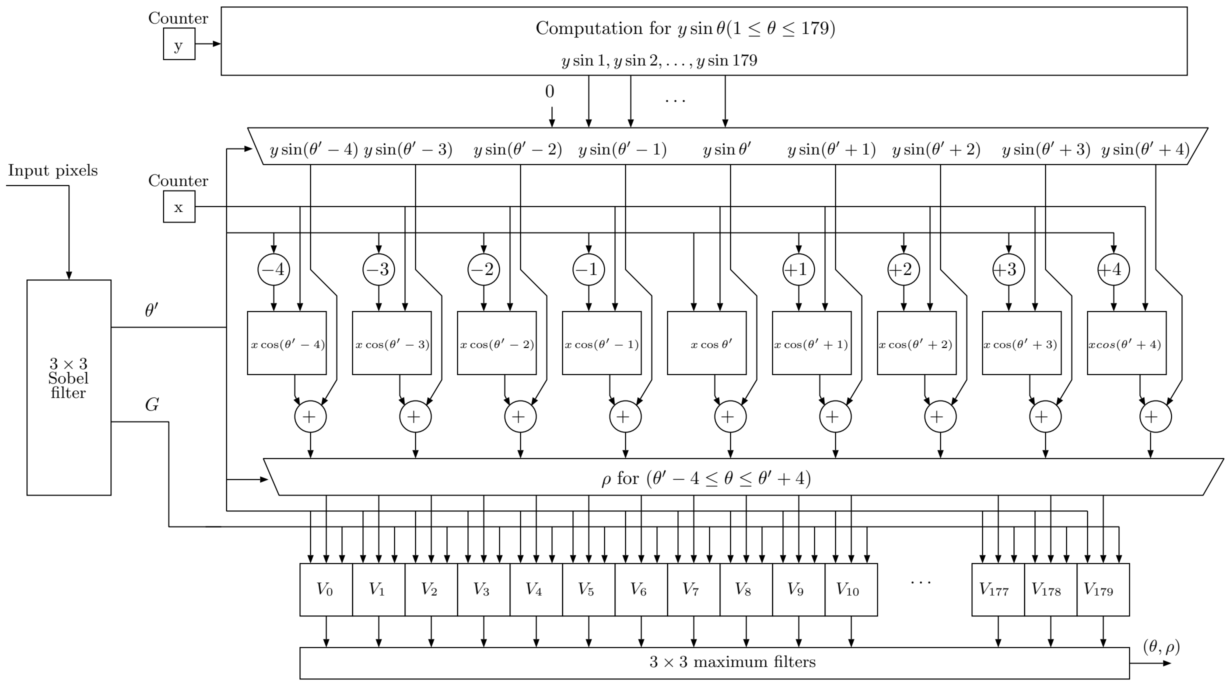

Counter
y
Computation for $y \sin\theta (1 \le \theta \le 179)$
$y\sin 1, y\sin 2, \dots, y\sin 179$
0
Input pixels
Counter
x
3×3 Sobel filter
θ'
G
$y\sin(\theta'-4)$ $y\sin(\theta'-3)$ $y\sin(\theta'-2)$ $y\sin(\theta'-1)$ $y\sin\theta'$ $y\sin(\theta'+1)$ $y\sin(\theta'+2)$ $y\sin(\theta'+3)$ $y\sin(\theta'+4)$
-4 -3 -2 -1 $+1$ $+2$ $+3$ $+4$
$x\cos(\theta'-4)$ $x\cos(\theta'-3)$ $x\cos(\theta'-2)$ $x\cos(\theta'-1)$ $x\cos\theta'$ $x\cos(\theta'+1)$ $x\cos(\theta'+2)$ $x\cos(\theta'+3)$ $x cos(\theta'+4)$
ρ for $(\theta'-4 \le \theta \le \theta'+4)$
V_0 V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8 V_9 V_{10} V_{177} V_{178} V_{179}
3×3 maximum filters
(θ, ρ)

Figure 4. The outline of our FPGA architecture for the gradient-based Hough transform ($\lambda = 4$)

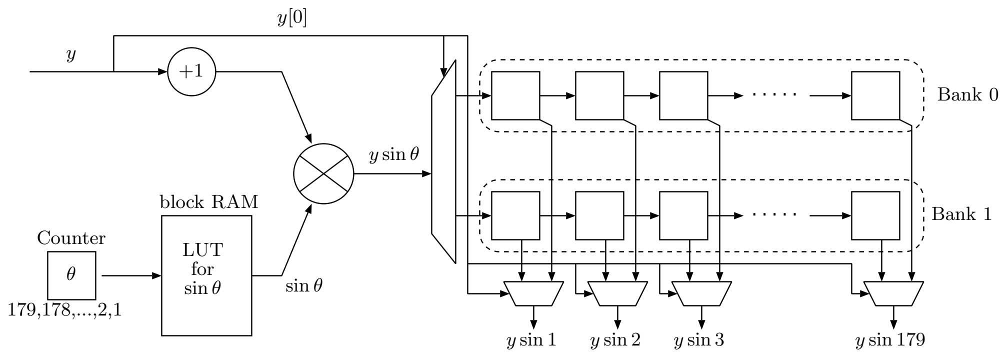


Figure 7. Architecture to compute $y_{k+1}\sin\theta$ with one DSP slice

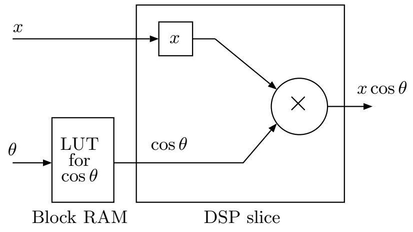


Figure 6. A DSP slice and a block RAM to compute $x\cos\theta$

of y does not change while processing pixels are in a certain row. Therefore, when pixels in a row y are processed, we pre-compute the values of $(y+1)\sin\theta$ for $(1 \le \theta \le 179)$ in the next row and store them into the registers. When pixels in the next row $y+1$ are processed, the values are used. Figure 7 illustrates our architecture to compute $ysin\theta$. We utilize a block RAM as a look-up-table to compute $sin\theta$, and one DSP slice to compute a product of y and $\sin\theta$. Also, we use two series of registers, called banks. One is used to pre-compute the values of $y\sin\theta$ for the next row. The other is used to output the already computed $y\sin\theta$ for the current processing row. To compute the values of $\sin\theta$, we successively generate the value of $\theta = 179, 178, ...2, 1$ by a counter. By inputting them to the look-up-table, the values of $\sin\theta$ are obtained. The products of $y\sin\theta$ are computed using a DSP slice. Note that the values of $y\sin\theta$ are for the next row. Therefore, the value of y is incremented in advance. The obtained values are successively input to a bank of registers. In each bank, registers are connected in cascade as shown in the figure. The values shift one by one until all the values are input to the bank. When pixels in a row are finished, the banks are switched.

Figure 8 illustrates the architecture of V_θ using a block RAM. Given gradient direction θ' and magnitude G, the voted value $G \cdot w(\theta - \theta')$ is computed, where θ is a constant value in each V_θ. Since in our implementation the value of $w(\theta - \theta')$ is power of two shown in Section III, it can be computed with a subtractor and a bit shifter. A block RAM in the FPGA is dual port architecture. Xilinx Virtex-7 Family has 36Kbit dual-port block RAMs, which have two sets of ports operated independently. Two sets of ports are:

Port Set A *ADDRA* (ADDRess A), *DOA* (Data Output A), *DIA* (Data Input A), and

Port Set B *ADDRB* (ADDRess B), *DOB* (Data Output B), *DIB* (Data Input B).

Let $M[i]$ denote a data of address i of the block RAM. In read operation of Port Set A, $M[ADDRA]$ is output from *DOA* after the rising clock edge. In write operation of Port Set A, the data given to DIA is written in $M[ADDRA]$ at the rising clock edge. Read/write operations of Port Set B are the same as Port Set A. Port Set A and Port Set B work independently. In the block RAMs in 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 RF mode, if reading and writing operations are performed to the same address, reading operation is performed before the writing operation. Hence the reading data is the data before writing data. On the other hand, in the WF mode, since the writing performed before the reading, the reading data is the updated data. However, when a dual port is used, 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 [30].

We use the block RAM to store the values of $v[\theta][\rho]$ ($-\frac{n}{\sqrt{2}} < \rho \leq \frac{n}{\sqrt{2}}$). Let $v_\theta[i]$ denote a data of address i of the block RAM V_θ. Since ρ is given to its *ADDRA*, $v_\theta[\rho]$ is output from *DOA* after the rising clock edge as illustrated in Figure 8. After that, $v_\theta[\rho] + G \cdot w(\theta - \theta')$ is computed and it is given to *DIB*. Since ρ is given to *ADDB*, $v_\theta[\rho] + G \cdot w(\theta - \theta')$ is written in $v_\theta[\rho]$. In other words, $v_\theta[\rho] \leftarrow v_\theta[\rho] + G \cdot w(\theta - \theta')$ is performed. At that time, according to the restriction stated in the above, since the same value of ρ may be input continuously, the setting of block RAMs must be RF. Namely, when the same value of ρ is input continuously, the former voted value is not read from the block RAM. To avoid this situation, we use an additional register to store the latest voted value and if the same value of ρ is input continuously, the stored value is used instead of the value read from the block RAM.

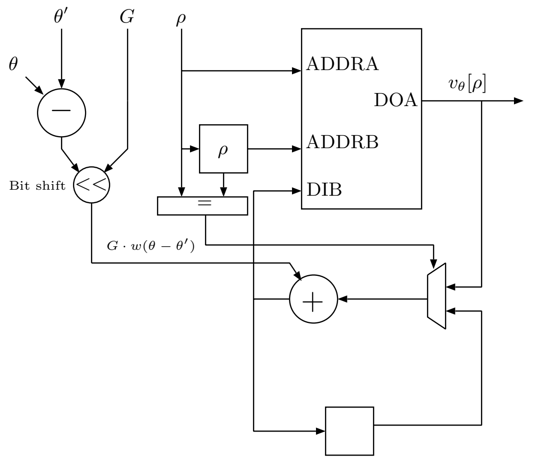
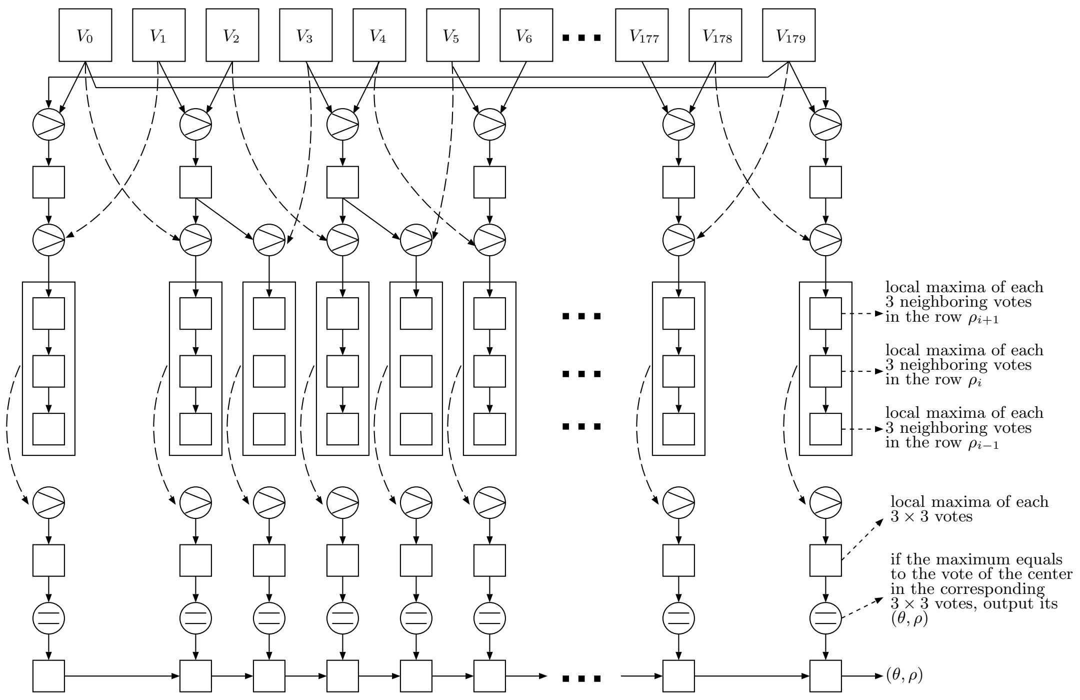


Figure 8. A block RAM V_θ to store $v[\theta][\rho]$

In the following, when all the voting operations are completed, we utilize a maximum filter to output the final correct identified straight lines. The maximum filter is defined as the maximum of all pixels within a local region of an image. In here, for each value in the voting space, this filter copies the largest value from a 3×3 region to it. Figure 9 illustrates our architecture to perform a 3×3 maximum filter to the voted results. Since the voted values in the same ρ can be obtained from $V_0,\ V_1, \ldots,\ V_{179}$, this architecture works row by row in a pipeline fashion. To perform a 3×3 maximum filter to each value in a certain row, it is concurrently read from $V_0,\ V_1, \ldots,\ V_{179}$. After that using comparators, local maxima of each 3 neighboring votes in the row are obtained. These local maxima are input to shift registers. After local maxima in the 3 rows are computed, local maxima of each 3×3 votes are obtained by computing maxima from corresponding 3 values. If the maximum equals to the original value of the center in the corresponding 3×3 votes, its (θ, ρ) that represents a probable line is input to the shift registers and output through the registers.

C. Data representation

The choice of data precision is guided by the implementation cost in terms of area, simplicity of design, speed and power consumption. Higher precision will lead to less quantization error in the final implementation. On the other hand, lower precision will produce more compaction and faster designs with less power consumption. A trade-off



V_0
V_1
V_2
V_3
V_4
V_5
V_6
V_{177}
V_{178}
V_{179}
local maxima of each 3 neighboring votes in the row ρ_{i+1}
local maxima of each 3 neighboring votes in the row ρ_i
local maxima of each 3 neighboring votes in the row ρ_{i-1}
local maxima of each 3×3 votes
if the maximum equals to the vote of the center in the corresponding 3×3 votes, output its (θ, ρ)
(θ, ρ)

Figure 9. Pipeline architecture of 3 maximum filter

choice needs to be made depending on the given application and available FPGA resources.

In this paper, the data format of inputs are 8-bit integer of all pixels in the gray-scale image and these values are input in raster scan order. The coordinates (x, y) which are necessary to compute ρ are appropriately generated by the counters as shown in Figure 4. In order to minimize chip space and computation time, short fixed point representation of numbers is used. Considering the structure of DSP slices and block RAMs, we choose the data representation in our implementation, as follows. The data format of inputs that are values of pixels $p[x][y]$ is 8-bit integer. The data format of $\cos\theta$ and $\sin\theta$ is 16-bit fixed point number, which consists of 1-bit sign, 1-bit integer and 14-bit fraction based on two's complement. On the other hand, the data formats of gradient magnitude G and gradient direction θ' are 12-bit and 8-bit integers, respectively. The data format of ρ is 10-bit two's complement integer. Since the range of the value of θ is 0 to 179, the data format of θ is 8-bit integer. The data format of the voted value is 24-bit integer.

V. Performance Evaluation and Experimental Results

We have implemented the proposed architecture for the gradient-based Hough transform and evaluated it on the Xilinx Virtex-7 FPGA XC7VX485T-2. Table I shows the experimental results using Xilinx ISE 14.1.

In our implementation, the voted range of the gradient-based Hough transform shown in Section III is set to $\lambda = 4$, that is for local gradient direction θ', we perform the voting operation to the range $\theta' - 4 \leq \theta \leq \theta' + 4$. The range was obtained by our experiments. Due to the stringent page limitation, we omit how to determine the range. However, the range is enough to extract lines because the error between the angle of lines obtained by the Sobel filter and the actual angle is small [31].

Figure 10 shows the result of lines detection for the conventional Hough transform and the gradient-based Hough transform. Compared with the result of the conventional Hough transform, we can see that the gradient-based Hough transform obtained more correct lines and exclude the inexistent lines.

Let us evaluate the performance for an $n \times n$ gray-scale image. In our implementation, the circuit can work in fully pipelined fashion. Namely, input pixels can be provided to the circuit clock by clock in raster scan order. To reduce the delay of the circuit, some pipeline registers are inserted into between circuit elements. It takes $2n + 44$ clock cycles to complete voting from the first input pixel is given to its voting is finished. Since the input image consists of n^2 pixels, the voting operations are performed in $n^2 + 2n + 44$ clock cycles, i.e., $\frac{n^2+2n+44}{260.061}\mu s$. After voting, $\sqrt{2}n + 188$ clock cycles, i.e., $\frac{\sqrt{2}n+188}{260.061}\mu s$ are necessary to output identified straight lines with 3×3 maximum

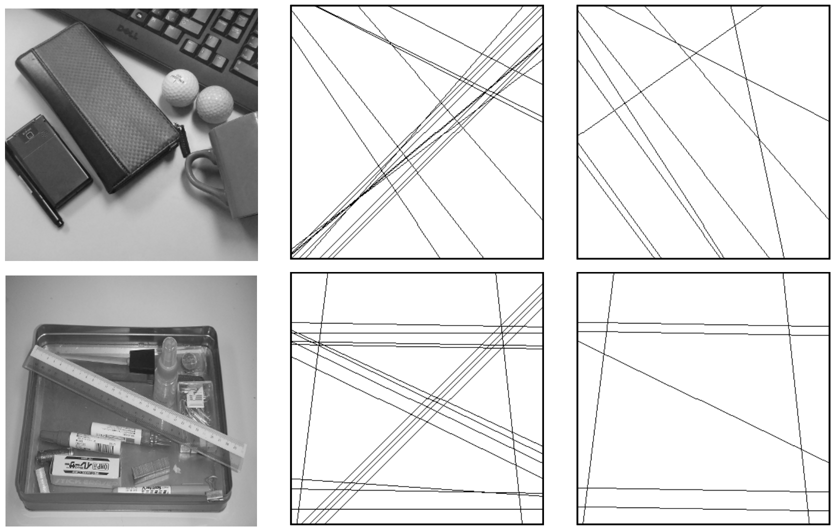

Original gray-scale image
Lines detection by the conventional Hough transform
Lines detection by the gradient-based Hough transform

Figure 10. Comparison between conventional and gradient-based Hough transform algorithms

filters. Therefore, in total, $n^2 + (\sqrt{2} + 2)n + 232$ clock cycles, i.e., $\frac{n^2+(\sqrt{2}+2)n+232}{260.061}\mu s$ are necessary to perform the gradient-based Hough transform. If an input image of size 1000×1000 is given, our circuit can detect straight lines in $3.859ms$.

Table I
PERFORMANCE EVALUATION OF THE PROPOSED ARCHITECTURE FOR THE GRADIENT-BASED HOUGH TRANSFORM

DSP48E1 slices (out of 2800)	13 (1%)
36Kbit block RAMs (out of 1030)	180 (17%)
18Kbit block RAMs (out of 2060)	8 (1%)
Slices (out of 607200)	80181 (13%)
Clock frequency [MHz]	260.061

For the purpose of estimating the speed up of our FPGA implementation, we have also implemented a software approach of the gradient-based Hough transform using GNU C. We have used Intel Xeon X7460 running in 2.66GHz and 128GB memory to run the sequential algorithm for the gradient-based Hough transform. For the image shown in Figure 1(a) whose size is 333×333, the software implementation can perform the gradient-based Hough transform in $133.519ms$. On the other hand, our circuit can perform it in $431.660\mu s$. Therefore, our FPGA implementation attains a speed-up factor of more than 309 over the sequential implementation on the CPU.

There are a number of literatures reported to implement Hough transform for lines using the FPGA shown in Section I. Algorithms, that is conventional or gradient-based, and performances such as device, logic blocks, DSP slices, frequency and throughput are compared in Table II. It is difficult to directly compare to other works because used algorithms, utilized FPGAs and supported size of images differ. Considering the throughput, however, it is clear that the performance of our FPGA implementation is better than that of other works.

VI. CONCLUSIONS

We have presented an efficient implementation of the gradient-based Hough transform for gray-scale images using DSP slices and block RAMs in the Virtex-7 Family FPGA. We have implemented the circuit using 13 DSP48E1 slices, 180 block RAMs with 36Kbits and 8 block RAMs with 18Kbits on the Virtex-7 Family FPGA XC7VX485T-2. The experimental results show that the architecture runs in 260.061MHz and for an $n \times n$ gray-scale image, our circuit can perform in $n^2 + (\sqrt{2} + 2)n + 232$ clock cycles, i.e., $\frac{n^2+(\sqrt{2}+2)n+232}{260.061}\mu s$, including the computation of gradient information.

REFERENCES

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

[2] 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.

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

Table II
COMPARISON WITH RELATED WORKS FOR HOUGH TRANSFORM

	Deng [20]	Lee [21]	Previous work [27]	Previous work [28]	Karabernou [19]	This work
Hough transform	Conventional	Conventional	Conventional	Conventional	Gradient-based	Gradient-based
Device	XC4010XL	Virtex 4	XC6VLX240T-1	XC6VLX240T-1	XC4010EPC84	XC7VX485T-2
Logic blocks	333 CLBs	314 CLBs	14493 Slices	40487 Slices	205 CLBs	82673 Slices
DSP slices	—	—	178 DSP48E1s	90 DSP48E1s	—	13 DSP48E1s
Frequency	40MHz	132MHz	245.519MHz	247.525MHz	23.166MHz	260.061MHz
Throughput	0.623Mpixel/s	32.768Mpixel/s	245.428Mpixel/s	247.430Mpixel/s	10.368Mpixel/s	263.979Mpixel/s

[4] 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. 49–62, 2011.

[5] 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. 56–78, 2012.

[6] 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.

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

[8] 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, 2012.

[9] Y. Ago, K. Nakano, and Y. Ito, "A classification processor for a support vector machine with embedded DSP slices and block RAMs in the FPGA," in *Proc. of the IEEE 7th International Symposium on Embedded Multicore SoCs*, 2013, pp. 91–96.

[10] K. Hashimoto, Y. Ito, and K. Nakano, "Tempate matching using DSP slices on the FPGA," in *Proc. of International Symposium on Computing and Networking*, 2013, pp. 338–344.

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

[12] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[13] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 1, pp. 87–116, 1988.

[14] F. O'Gorman and M. Clowes, "Finding picture edges through collinearity of feature points," *Computers, IEEE Transactions on*, vol. C-25, no. 4, pp. 449–456, 1976.

[15] M. S. Nixon and A. S. Aguado, *Feature Extraction and Image Processing*, 2nd ed. Academic Press, 2008.

[16] S. Tagzout, K. Achour, and O. Djekoune, "Hough transform algorithm for FPGA implementation," *Signal Processing*, vol. 81, no. 6, pp. 1295–1301, 2001.

[17] H. Bessalah, S. Seddiki, F. Alim, and M. Bencherif, "On line mode incremental Hough transform implementation on Xilinx fpga's," in *Proc. of the 8th conference on Signal, Speech and image processing*, 2008, pp. 176–179.

[18] O. Djekoune and K. Achour, "Incremental Hough transform: an improved algorithm for digital device implementation," *Real-Time Imaging*, vol. 10, no. 6, pp. 351–363, 2004.

[19] S. M. Karabernou and F. Terranti, "Real-time FPGA implementation of Hough transform using gradient and CORDIC algorithm," *Image and Vision Computing*, vol. 23, no. 11, pp. 1009–1017, 2005.

[20] D. D. S. Deng and H. ElGindy, "High-speed parameterisable Hough transform using reconfigurable hardware," in *Proc. of the Pan-Sydeny area workshop on Visual information processing*, vol. 11, 2001, pp. 51–57.

[21] P. Lee and A. Evagelos, "An implementation of a multiplierless Hough transform on an FPGA platform using hybrid-log arithmetic," in *Proc. of Real-Time Image Processing 2008*, vol. 6811, 2008, pp. 68 110G–1.

[22] Xilinx Inc., *Virtex-4 FPGA User Guide(v2.6)*, 2008.

[23] ——, *Virtex-5 FPGA User Guide(v5.2)*, 2009.

[24] ——, *Virtex-6 Family Overview(v2.4)*, 2012.

[25] Altera Corp., *Stratix V Device Handbook*, 2012.

[26] Xilinx Inc., *7 Series FPGAs Overview (v1.14)*, 2013.

[27] X. Zhou, N. Tomagou, Y. Ito, and K. Nakano, "Implementations of the Hough transform on the embedded multicore processors," *International Journal of Networking and Computing*, vol. 4, no. 1, pp. 174–188, 2014.

[28] X. Zhou, Y. Ito, and K. Nakano, "An efficient implementation of the Hough transform using DSP slices and block RAMs on the FPGA," in *IEEE 7th International Symposium on Embedded Multicore SoCs*, 2013, pp. 85–90.

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

[30] ——, *7 Series FPGAs Memory Resources (v1.10)*, 2014.

[31] E. R. Davies, "Circularity – a new principle underlying the design of accurate edge orientation operators," *Image and Vision Computing*, vol. 2, no. 3, pp. 134–142, 1984.