Efficient Hough transform on the FPGA using DSP slices and block RAMs

Xin Zhou, Norihiro Tomagou, 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 a new FPGA architecture for the Hough transform that identifies straight lines in a binary image. Recent FPGAs have hundreds of embedded DSP slices and block RAMs. For example, Xilinx Virtex-6 Family FPGAs have a DSP48E1 slice, which is a configurable logic block equipped with fast multipliers, adders, pipeline registers, and so on. They also have a dual-port memory with 18Kbits as a block RAM. One of the most important key techniques for accelerating computation using FPGAs is an efficient usage of DSP slices and block RAMs. Our new architecture for the Hough transform uses 178 DSP48E1 slices and 180 block RAMs with 18Kbits that work in parallel. As far as we know, there is no previously published work that fully utilizes DSP slices and block RAMs for the Hough transform. Roughly speaking, a conventional sequential implementation performs $180m$ voting operations for m edge points. Our architecture performs voting operations in parallel, and outputs identified straight lines in $m+97$ clock cycles. Since $180m$ voting operations are performed using 178 DSP48E1 slices, the lower bound of the computing time is m clock cycles. Hence our implementation is close to optimal. The implementation results show that the Hough transform for a 512×512 image with 33232 edge points can be done in only 135.75μs.**

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

I. Introduction

Hough transform is a technique to find shapes in images [1]. 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 [2]. 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-dimentional image, the voting is performed along a curve $\rho = x\cos\theta + y\sin\theta$ $(0 \leq \theta < 180)$. 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. The result of voting to the parameter space is shown in Figure 2. 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)).

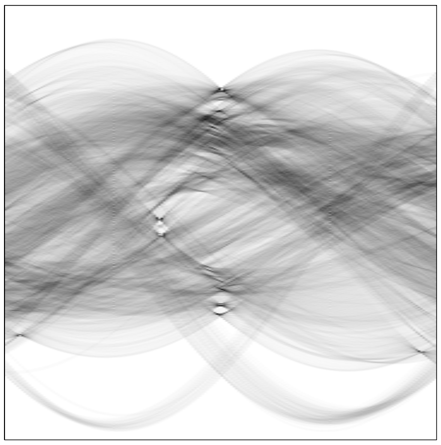
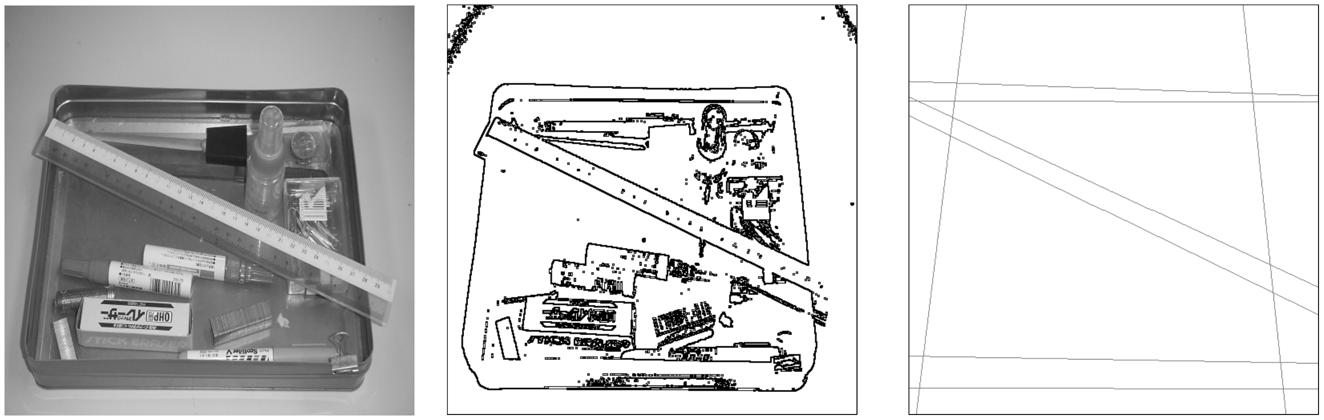


Figure 2. Hough parameter space

A Field Programmable Array (FPGA) is a programmable logic device designed to be configured by the customer or designer by hardware description language after manufacturing. The most common FPGA architecture consists of an array of logic blocks, I/O pads, block RAMs and routing channels. Furthermore, recent FPGAs have embedded DSP slices that make a higher performance and a broader application.

The Xilinx Virtex-6 series FPGAs have DSP48E1 slices that are equipped with a multiplier, adders, logic operators, etc [3]. More specifically, the DSP48E1 slice has a two-input multiplier followed by multiplexers and a three input adder/subtractor/accumulator. The DSP48E1 multiplier can



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

Figure 1. Example of straight line detection using the Hough transform

perform multiplication of an 18bit and a 25bit two's complement numbers and produces one 48bit 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-6 FPGA is an embedded memory supporting synchronized read and write operations. In the Virtex-6 FPGA, it can configured as 36Kbit dual port block RAMs, FIFOs, or two 18Kbit dual port RAMs. In our architecture, it is used as a 1K$\times$18bit dual port RAM.

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 [4], [5], [6], [7], [8], [9], [10], [11], [12].

The main contribution of this paper is to present a new FPGA architecture for the Hough transform fully utilizes embedded DSP slices and block RAMs. 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:
DSP slices are used to compute $x \cos\theta$ and $y \sin\theta$ in parallel for each edge pixel (x, y). We compute $x \cos\theta$ and $y \sin\theta$ for θ such that $0 \leq \theta < 90$ instead of computing them for θ such that $0 \leq \theta < 180$. Also, we avoid the computation of the values of $\cos\theta$ and $\sin\theta$ by pre-loading them in the DSP slices.

Fully Pipelined Architecture:
We take into account a layout of DSP slices and block RAMs in Virtex-6 FPGA architecture, and design our Hough transform architecture as a fully pipelined one. For example, in the Virtex-6 FPGA XC6VLX240T has 768 DSP48E1 slices arranged in 8 columns of 96 adjacent DSP48E1 slices. Neighboring DSP48E1 slices are connected directly through pipeline registers. Our Hough transform architecture uses 2 columns to compute $x \cos\theta$ and $y \sin\theta$ each, and uses a pipeline technique to maximize the clock frequency.

Using these ideas, our new architecture for the Hough transform uses 178 DSP48E1 slices and 180 block RAMs with 18Kbits that work in parallel. One of the most important key techniques for accelerating computation using FPGAs is an efficient usage of DSP slices and block RAMs. Nevertheless, as far as we know, there is no previously published work that fully utilizes DSP slices and block RAMs for the Hough transform. Roughly speaking, a conventional sequential implementation performs $180m$ voting operations for m edge points. Our architecture performs voting operations in parallel, and outputs identified straight lines in $m+97$ clock cycles. Since $180m$ voting operations are performed using 178 DSP48E1 slices, the lower bound of the computing time is m clock cycles. Hence our implementation is close to optimal. We have implemented our new architecture on a Virtex-6 family FPGA XC6VLX240T-1. The circuit runs in 245.519MHz and outputs identified straight lines in $m+97$ cycles. For example, Figure 1 includes 33232 edge points. Therefore, the circuit can perform the Hough transform in $135.75\mu s$.

Many hardware algorithms for FPGA implementation of the Hough transform for lines have been proposed in past. As far as we know, however, there is no published hardware algorithm using embedded DSP slices or multipliers in the

FPGA. In the existing researches, instead of circuits of multiplication with DSP slices or multipliers, they introduced incremental Hough transform [13], [14], [15], CORDIC [16], [17], and hybrid-log arithmetic [18] to the computation of the Hough transform. Since most of recent FPGAs produced by principal vendors equip embedded DSP blocks [19], [20], [21], one of the most important key techniques for accelerating computation using FPGAs is an efficient usage of DSP slices and block RAMs.

This paper is organized as follows. Section II introduces the Hough transform algorithms for lines. We show the FPGA architecture for the Hough transform in Section III. Section IV shows the experimental results. Finally, Section V concludes the paper.

II. Hough Transform

The main purpose of this section is to review the Hough transform algorithms for straight lines. 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 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 a 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 for every pixel in the xy-space. Let $(x_0, y_0), (x_1, y_1), \ldots, (x_{k-1}, y_{k-1})$ be the k pixels in xy-space. The Hough transform is spelled out as follows:

[Straight Forward Hough Transform]

```
for $i \leftarrow 0$ to $k-1$
    for $\theta \leftarrow 0$ to 179
        begin
            $\rho \leftarrow x_k\cos\theta + y_k\sin\theta$
            $v[\theta][\rho] \leftarrow v[\theta][\rho] + 1$
        end
for $\theta \leftarrow 0$ to 179 do in parallel
    for $\rho \leftarrow -\frac{n}{\sqrt{2}}$ to $\frac{n}{\sqrt{2}}$ do in parallel
        output $(\theta, \rho)$ if $v[\theta][\rho] \geq$ *threshold*
```

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

We will show that, it is sufficient to compute these values for $\theta = 0, 1, \ldots, 90$. From the addition theorem of trigonometric functions, we have

$$\begin{aligned} \rho &= x_k\cos(180-\theta) + y_k\sin(180-\theta) \\ &= -x_k\cos(\theta) + y_k\sin(\theta). \end{aligned} \qquad (2)$$

Using Formula (2), the Hough transform can also be done by partitioning the range $[0, 179]$ of θ into two ranges $[0, 89]$ and $[90, 179]$. Also, we avoid going through array v for finding elements larger than a threshold. Thus, our new Hough transform, called the Circuit-oriented Hough Transform is be spelled out as follows:

[Circuit-oriented Hough Transform]

```
for $i \leftarrow 0$ to $k-1$ do
    begin
        for $\theta \leftarrow 0$ to 89 do
            begin
                $\rho \leftarrow x_k\cos\theta + y_k\sin\theta$
                $v[\theta][\rho] \leftarrow v[\theta][\rho] + 1$
                output $(\theta, \rho)$ if $v[\theta][\rho] =$ *threshold*
            end
        for $\theta \leftarrow 1$ to 90 do
            begin
                $\rho \leftarrow -x\cos(\theta) + y\sin(\theta)$
                $v[180-\theta][\rho] \leftarrow v[180-\theta][\rho] + 1$
                output $(\theta, \rho)$ if $v[\theta][\rho] =$ *threshold*
            end
    end
```

In the following section, we show an efficient implementation of the Circuit-oriented Hough Transform.

III. Our FPGA architecture for the Hough transform

This section describes our FPGA architecture for the Hough transform using DSP slices and block RAMs in Xilinx Virtex-6 FPGA. We use Xilinx Virtex-6 Family FPGA XC6VLX240T-1 as the target device [22].

A. Structure of our architecture for the Hough transform

Figure 4 illustrates our architecture for the Hough transform. We use 178 DSP slices $X_1, X_2, \ldots X_{89}$ and $Y_1, Y_2, \ldots, Y_{89}$. For each θ $(0 \leq \theta \leq 90)$ X_θ and Y_θ compute $x_k\cos\theta$ and $y_k\cos\theta$ for given x_k and y_k, respectively. Since $x_k\cos 0 = x_k$, $x_k\cos 90 = 0$, $y_k\sin 0 = 0$, and $y_k\cos 90 = y_k$, DSP slices X_0, X_{90}, Y_0, and Y_{90} are not necessary. Using an adder and a subtractor for each pair X_θ and Y_θ, $\rho_\theta = x_k\cos\theta + y_k\cos\theta$ and $\rho_{180-\theta} = -x_k\cos\theta + y_k\cos\theta$ are computed. We also use 180 block RAMs $V_0, V_1, \ldots V_{179}$ to store the voting value. Address ρ

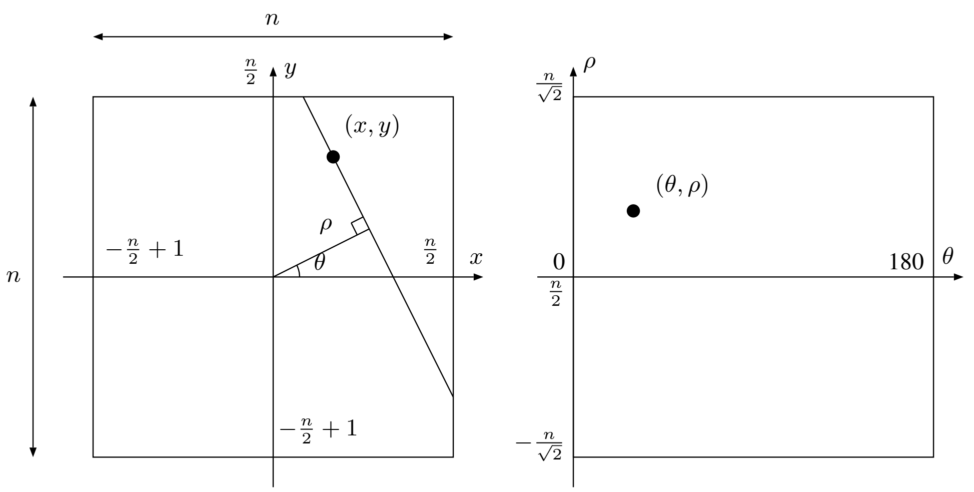


Figure 3. Two dimensional Spaces xy and $\theta\rho$ used in the Hough transform

of each V_θ ($0 \le \theta \le 179$) is used to store the value of $v[\theta][\rho]$.

To minimize the delay between registers, DSP slices $X_1, \ldots, X_{90}$ are connected in a pipeline fashion as illustrated in Figure 4. Each X_θ has a register to store the value of x_k. In every clock cycle, the value is transferred from X_θ to $X_{\theta+1}$. Similarly, DSP slices $Y_0, Y_1, \ldots, Y_{90}$ are connected in a pipeline fashion.

Figure 5 illustrates two DSP slices X_θ and Y_θ with an adder and subtractor to compute ρ. In X_θ, the value of x_k is loaded in an internal register. Also, the value of $\cos\theta$ is pre-computed. Note that the value of $\cos\theta$ used in X_θ is a fixed value. The product of x_k and $\cos\theta$ is computed in a multiplier of the DSP slice X_θ. Similarly, the value of $\sin\theta$ used in Y_θ is a fixed value and the product of y_k and $\sin\theta$ is computed in a multiplier of the DSP slice Y_θ.

In the Virtex-6 FPGA XC6VLX240T, that is our target device, DSP48E1 slices are arranged in 8 columns of 96 adjacent DSP48E1 slices. Neighboring DSP48E1 slices are connected directly through pipeline registers. Our Hough transform architecture uses 2 columns to compute $x_k \cos\theta$ and $y_k \sin\theta$ each, and uses a pipeline technique to maximize the clock frequency (Figure 6).

Figure 7 illustrates the architecture of V_θ using a block RAM. A block RAM in the FPGA is dual port architecture. Xilinx Virtex-6 Family has 18Kbit 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

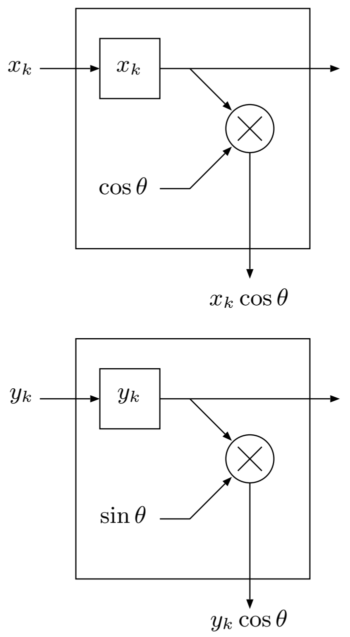


Figure 5. Two DSP slices X_θ and Y_θ with an adder and subtractor to compute ρ

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

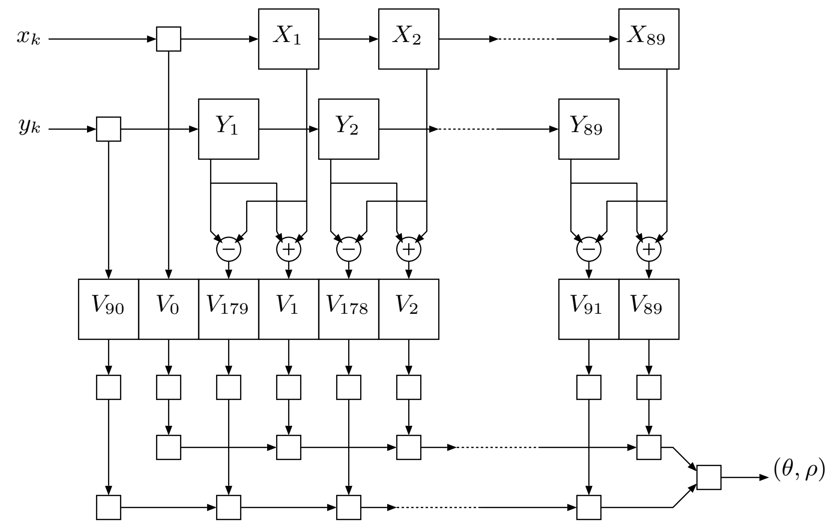


Figure 4. The outline of our FPGA architecture for the Hough transform

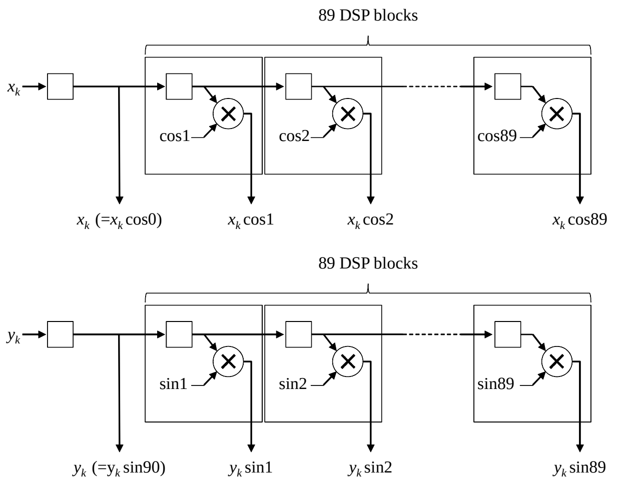


Figure 6. Pipeline architecture to compute $x_k \cos\theta$ and $y_k \sin\theta$ with DSP slices

reading 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 operations to the same address are performed for each port, the setting of block RAMs must be RF [23].

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 the data of address i in block RAM V_θ. Since ρ is given to it ADDRA, $v_\theta[\rho]$ is output from *DOA* after the rising clock edge as illustrated in Figure 7. After that, $v_\theta[\rho]+1$ is computed and it is given to *DOB*. Since ρ is given to *ADDB*, $v_\theta[\rho]+1$ is written in $v_\theta[\rho]$. In other words, $v_\theta[\rho] \leftarrow v_\theta[\rho]+1$ 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.

In the same time, a comparator is used to determine if $v_\theta[\rho]+1 = \mathit{threshold}$. If so, the value of ρ is written in a register. After that, a pair (θ, ρ) is written into a next register. The pair (θ, ρ) represents a probable line. It moves toward the output of the circuit using series of shift registers one by one shown in Figure 4. In order to reduce the number of clock cycles necessary to move data to the output, we use two series of shift registers. One is used for output data of $V_0, \ldots, V_{89}$. The other is used for output data of $V_{90}, \ldots, V_{179}$. Therefore, the number of clock cycles necessary to move data to the output is reduced to at most 90 clock cycles.

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

In our work, in order to minimize chip space and computation time, short fixed point representation of numbers

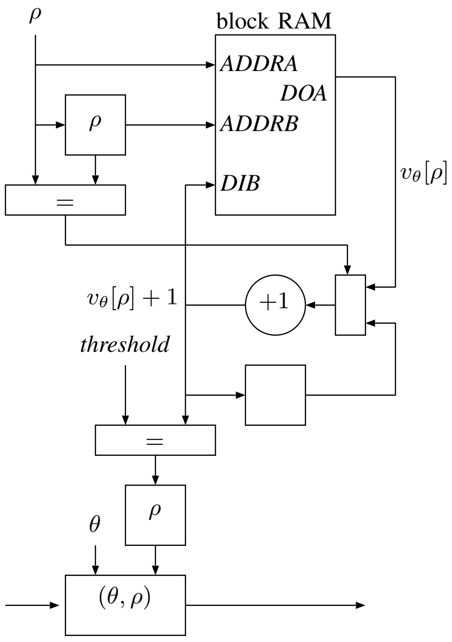


Figure 7. A block RAM V_θ to store $v[\theta][\rho]$

are used. Considering the structure of DSP slices and block RAMs, we choose the data presentation in our implementation, as follows. The data format of inputs that are pairs of coordinates x_k and y_k are 10bit two's complement integer each. Also, the data format of $\cos\theta$ and $\sin\theta$ is 16bit fixed point number, which consists of 1bit sign, 1bit integer and 14bit fraction based on two's complement. On the other hand, the data format of ρ is 10bit two's complement integer. The data format of the voted value is 18bit integer. Namely, the number of the vote is at most $2^{18}-1$. Since the range of the value of θ is 0 to 180, the data format of θ is 8bit integer.

IV. Experimental Results

We have implemented and evaluated our proposed methods of the Hough transform on the FPGA. For the purpose of estimating the speed up of our implementations, we have also implemented a conventional software approach of the 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 Hough transform. For the image shown in Figure 1(b) that includes 33232 edge points, the software implementation can perform the Hough transform in $37.10ms$. If the input image is worst case in terms of the computing time, that is, if all the points of an image of size $512\times512(=262144)$ are edge points, it takes $359.27ms$ to complete to output the results.

In the evaluation, we have used the Xilinx Virtex-6 FPGA XC6VLX240T-1. Table I shows the experimental results using Xilinx ISE 13.1. In the implementation, to reduce the delay of the circuit, some pipeline registers are inserted into between circuit elements. It takes 3 clock cycles to compute the values of ρ for given x_k and y_k. Also, 4 clock cycles are necessary to output a pair (θ,ρ) that represents a probable line. Moreover, the number of clock cycles necessary to move data to the output is reduced to at most 90 clock cycles. Therefore, this circuit can output identified straight lines represented by (θ,ρ) in $m+97$ cycles, i.e., $\frac{m+97}{245.519}\mu s$. For example, Figure 1(b) includes 33232 edge points. Therefore, the circuit can perform the Hough transform in $135.75\mu s$. Also, if all the points of an image of size $512\times512(=262144)$ are edge points, it takes $1068.11\mu s$ to complete to output the results. Of course, it is not possible that all points are edge points, however, this fact guarantees that our Hough transform implementation for any 512×512 image terminates in less than $1068.11\mu s$.

Table I
Performance evaluation of the proposed architecture for the Hough transform

DSP48E1 slices (out of 768)	178 (23.1%)
18Kbit block RAMs (out of 832)	180 (21.6%)
Slices (out of 301440)	14493 (4.81%)
Clock frequency [MHz]	245.519

Table II shows the computing time of the Hough transform for Figures 1(b), 8(a), and 9(a). According to the table, we can find that the computing time for both the CPU implementation and the FPGA implementation almost depends on the number of the edge points, not the size of the image.

There are a number of literatures reported to implement the Hough transform for lines using the FPGA shown in Section I. Performances such as device, logic blocks, DSP slices, frequency and throughput are compared in Table III. It is difficult to directly compare to other works because 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.

Table III
Comparison with related works for the Hough transform using FPGAs

	Karabernou [16]	Deng [17]
Device	XC4010EPC84	XC4010XL
Logic blocks	205 CLBs	333 CLBs
DSP slices	—	—
Frequency	23.166MHz	40MHz
Throughput	10.368Mpixel/s	0.623Mpixel/s

	Lee [18]	This work
Device	Virtex 4	XC6VLX240T-1
Logic blocks	314 CLBs	14493 Slices
DSP slices	—	178 DSP48E1s
Frequency	132MHz	245.519MHz
Throughput	32.768Mpixel/s	245.428Mpixel/s

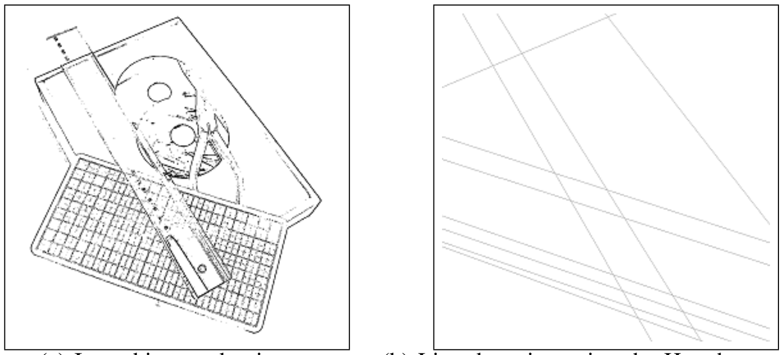

(a) Input binary edge image (b) Line detection using the Hough transform

Figure 8. Example of straight line detection using the Hough transform (1024×1024, 23293 edge points)

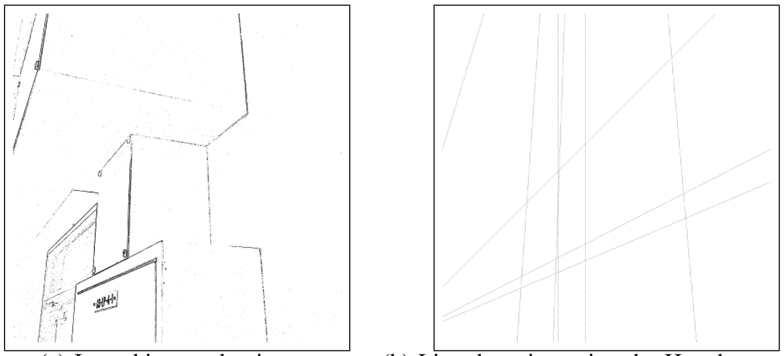

(a) Input binary edge image (b) Line detection using the Hough transform

Figure 9. Example of straight line detection using the Hough transform (4096×4096, 80092 edge points)

Table II
COMPUTING TIME OF THE HOUGH TRANSFORM

Image	Size	# edge points	Time (FPGA)	Time (CPU)	Speed-up
Figure 1(b)	512×512	33232	135.75μs	37.10ms	273.3
Figure 8(a)	1024×1024	23293	95.27μs	27.47ms	288.3
Figure 9(a)	4096×4096	80092	326.61μs	121.64ms	372.4

V. CONCLUSIONS

We have presented a new FPGA implementation for the Hough transform that identifies straight lines in a binary image. Our basic idea is to partition the voting space and the voting operation is performed in parallel. In our implementation, we utilize DSP slices and block RAMs on the Virtex-6 Family FPGA. Partitioning the parameter space to vote, the 180 voting operations are performed in parallel with 178 DSP48E1 slices and 180 18Kbit block RAMs. We have implemented our architecture on the Virtex-6 Family FPGA XC6VLX240T-1. The experimental results show that this implementation runs in 245.519MHz and given m coordinates of edge points, it can output identified straight lines in $m+97$ cycles, i.e., $\frac{m+97}{245.519}\mu s$. The implementation results show that the Hough transform for a 512×512 image with 33232 edge points can be done in $135.75\mu s$ on the FPGA.

REFERENCES

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

[2] 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.

[3] Xilinx Inc., *Virtex-6 FPGA DSP48E1 Slice User Guide (v1.3)*, 2011.

[4] 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.

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

[6] Y. Ito and K. Nakano, "Low-latency connected component labeling using an FPGA," *International Journal on Foundations of Computer Science*, pp. 405–426, 2010.

[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*, pp. 1–13, 2012.

[8] Y. Ito and K. Nakano, "A new FM screening method to generate cluster-dot binary images using the local exhaustive search with FPGA acceleration," *International Journal on Foundations of Computer Science*, pp. 1373–1386, 2008.

[9] ——, "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.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

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

[22] Xilinx Inc., *Virtex-6 Family Overview(v2.4)*, 2012.

[23] ——, *Virtex-6 FPGA Memory Resources User Guide (v1.6)*, 2011.