IEICE TRANS. INF. & SYST., VOL.E96–D, NO.8 AUGUST 2013

PAPER Special Section on Reconfigurable Systems

FPGA Implementation of Human Detection by HOG Features with AdaBoost*

Keisuke DOHI[†a)], **Kazuhiro NEGI**[†], **Yuichiro SHIBATA**[††], ***and*** **Kiyoshi OGURI**[††], ***Members***

SUMMARY We implement external memory-free deep pipelined FPGA implementation including HOG feature extraction and AdaBoost classification. To construct our design by compact FPGA, we introduce some simplifications of the algorithm and aggressive use of stream oriented architectures. We present comparison results between our simplified fixed-point scheme and an original floating-point scheme in terms of quality of results, and the results suggest the negative impact of the simplified scheme for hardware implementation is limited. We empirically show that, our system is able to detect human from 640 × 480 VGA images at up to 112 FPS on a Xilinx Virtex-5 XC5VLX50 FPGA.

key words: *histogram of oriented gradients, AdaBoost, human detection, FPGA*

1. Introduction

In this paper, we present external memory-free FPGA implementation of a real-time image-based human detection system. The image-based human detection generally consists of two stages; calculation of feature amount of given images and pattern classification based on machine learning. In this implementation, histograms of oriented gradients (HOG) [2] and AdaBoost classifiers [3] are used as feature amount and pattern classifiers, respectively. The HOG feature roughly describes object shape of local regions of given images and this is widely used for various object recognition such as pedestrian and car detection [4]–[6]. High-performance and compact implementation is achieved by making deep pipelined arithmetic structure and a high bandwidth on-chip RAMs. Our streamed processing approach illustrated by Fig. 1 achieves real-time human detection for input video frames without any external memory. The streamed architecture consists of three main parts: registers, a FIFO, and pipeline(s). The registers hold the data for pipeline parts and shift to next registers or a FIFO. The FIFO part holds the data that is not required by pipeline parts and pass the data to the next line of registers. The pipeline part does actual computation and outputs results for next image processing. One advantage of this architecture is any huge memory to store whole input frame data is not used; instead, FIFO to store only a few lines of data is needed. This is a preferable character also in term of energy efficiency. Many previous researches were reported which used this external memory-free architecture, especially for image processing, including our previous works [7]–[9].

So far, hardware implementation of HOG-based object detection has been actively investigated. Cao *et al.* [10] presented FPGA implementation of a stop sign detection system using HOG features. By using a simplified 4-bin HOG method, their architecture achieves a processing throughput of 60 frames per second (FPS) for 752 × 480 images on a Virtex-4 SX35 FPGA. However, this simple detection method is not directly applicable for human detection. Kadota *et al.* [11] presented a novel simplification technique of the HOG feature extraction for efficient FPGA implementation. Their architecture can process 640 × 480 image at 30 FPS with operating frequency 127.49 MHz on Stratix II FPGA, but detection part is not implemented. After we presented a preliminary version of this paper [1], Komorkiewicz *et al.* [12] presented fully-pipelined HOG and SVM implementation without using any external memory. To achieve superior accuracy, they used single-precision floating-point arithmetic for all stages of processing on a Virtex-6 XC6VLX240T FPGA. Their architecture needs multiple clock domains: 25 MHz clock for the HOG feature extraction and up to 237 MHz clock for SVM classifiers. Their architecture is able to process 640×480 images at 60 FPS in real-time. While their system shares some architectural concepts with ours in terms of streamed processing, an aspect of low-cost and low-energy implementation is more emphasized in our approach. Mizuno *et al.* [13] presented HOG and SVM implementation for HDTV resolution video images, which is able to process 1920 × 1080

Manuscript received November 10, 2012.
Manuscript revised March 13, 2013.

[†]The authors are with the Department of Science and Technology, Graduate School of Engineering, Nagasaki University, Nagasaki-shi, 852–8521 Japan.
[††]The authors are with the Division of Electrical Engineering and Computer Science, Graduate School of Engineering, Nagasaki University, Nagasaki-shi, 852–8521 Japan.

*A preliminary version of this paper appeared in the Proceedings of the 2011 International Conference on Field-Programmable Technology, New Delhi, India, December 12–14, 2011 [1].

a) E-mail: dohi@pca.cis.nagasaki-u.ac.jp

DOI: 10.1587/transinf.E96.D.1676

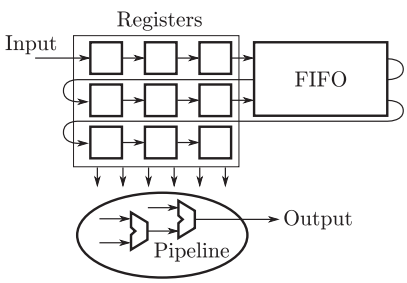


Fig. 1 Streamed architecture.

images at 30 FPS with a Cyclone IV EP4CE115 FPGA operating at 76.2 MHz. Their architecture is based on an SoC style, in which an HOG feature extraction module is connected to a soft-core processor with on-chip buses on the FPGA. Contrasting to our approach, two types of external memories with SDRAM and SRAM are attached to the FPGA and aggressively used to process image data.

The contributions of this paper are as follows: a) presenting a unified pipelined architecture of HOG feature extraction; b) using AdaBoost classifiers for real-time human detection; c) only single clock frequency is needed for HOG feature extraction and AdaBoost classifying; and d) our architecture is constructed without any external memory. Section 2 explains fundamentals of the HOG feature extraction. Section 3 shows the reduction techniques of the calculation amount for efficient implementation of human detection on an FPGA. Then, Sect. 4 details FPGA implementation with on-chip Block RAM and shift registers. Section 5 presents evaluation of the proposed architecture. Finally, Sect. 6 summarizes the paper.

2. Algorithms

In this section we explain two algorithms that we have implemented, the HOG algorithm for feature extraction from an input image and AdaBoost classifiers for real-time human detection. Along with the original HOG described in [2], we used some extended schemes for compact FPGA implementation [11], [14].

2.1 HOG Features

The histograms of oriented gradients (HOG) use local histograms of oriented gradients of pixel luminance for feature extraction from a given image. In our implementation, the process of HOG feature extraction roughly consists of the following four stages:

1. Luminance gradients calculation
2. Histogram generation
3. Histogram normalization
4. Feature binarization

The first step is to calculate luminance values from a given image. We used lightness in the HSL color model as luminance for ease of luminance extraction from RGB full color images. In this scheme, the luminance L for each pixel is given by the following equation:

$$L = \frac{\max(R, G, B) + \min(R, G, B)}{2}, \tag{1}$$

where R, G and B mean values of each color channel of given image. All the values are presented as 8-bit unsigned integers, i.e., the value from 0 to 255.

Using the luminance, 1st-order central-differences in both x and y direction, g_x and g_y, are given by:

$$\begin{aligned} g_x(x, y) &= L(x + 1, y) - L(x - 1, y) \\ g_y(x, y) &= L(x, y + 1) - L(x, y - 1), \end{aligned} \tag{2}$$

where $L(x, y)$, $g_x(x, y)$ and $g_y(x, y)$ mean values of luminance and central-differences at the coordinate (x, y), respectively. Then a magnitude m as well as an orientation θ of the gradient are computed by:

$$\begin{aligned} m(x, y) &= \sqrt{g_x(x, y)^2 + g_y(x, y)^2} \\ \theta(x, y) &= \tan^{-1} \frac{g_y(x, y)}{g_x(x, y)}, \end{aligned} \tag{3}$$

respectively.

After computing gradient magnitudes and orientations for each coordinate, the second step, histogram generation, is started. A histogram is generated for each *cell*, a square region of $p \times p$ pixels, by accumulating the magnitude values according to each orientation of all pixels in a cell. Note that cells have no overlap with neighbors which means that a total of $\frac{w}{p} \times \frac{h}{p}$ cells are defined for $w \times h$ luminances of the given image. In this implementation, we used $p = 5$.

To make histograms, the gradient magnitudes are voted into 8 bins according to their orientations as shown in Fig. 2. When an orientation θ meets the following condition:

$$\frac{n}{8}\pi - \frac{\pi}{16} \le \theta < \frac{n}{8}\pi + \frac{\pi}{16}, \tag{4}$$

the corresponding magnitude is voted to the bin b_n. Note that since the HOG does not focus on gradient directions but orientations, the opposite direction locates in the same bin. Since we vote gradients into eight bins, eight-dimension feature vector is eventually generated for each cell as shown in Fig. 3. Then the feature vector for the cell is described as:

$$\boldsymbol{f} = (f_0, f_1, \ldots, f_7) \tag{5}$$

where f_n means the sum of voted gradient magnitudes for bin b_n.

The third step, histogram normalization, is one of the most complex process. A histogram $\boldsymbol{v}$ for the *block* at (i, j), the larger spatial region which consists of 3×3 cells in this implementation, is defined as:

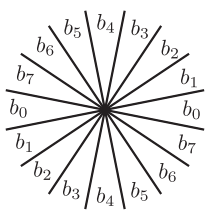


Fig. 2 Orientation spacing of 8-bin.

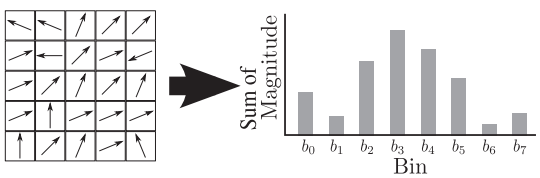


Fig. 3 Voting magnitude for bins.

$$\begin{aligned}\boldsymbol{v} = (&\\ &\boldsymbol{f}(i,j), \boldsymbol{f}(i+1,j), \boldsymbol{f}(i+2,j),\\ &\boldsymbol{f}(i,j+1), \boldsymbol{f}(i+1,j+1), \boldsymbol{f}(i+2,j+1),\\ &\boldsymbol{f}(i,j+2), \boldsymbol{f}(i+1,j+2), \boldsymbol{f}(i+2,j+2)\\ &),\end{aligned} \tag{6}$$

where $\boldsymbol{f}(i,j)$ means the feature vector for the cell located at the i-th row and j-th column. Since $\boldsymbol{f}$ is eight-dimension vector, $\boldsymbol{v}$ has $8 \times 9 = 72$ dimensions. Note that the block scans the entire image in a cell-by-cell manner (stride 1) and thus the number of blocks is equal to that of cells.

The values of histograms in a block are normalized using the L1-norm scheme described in [2]. The normalized histogram $\boldsymbol{v}_n$ is computed as:

$$\begin{aligned}\boldsymbol{v}_n &= \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_1 + \varepsilon}\\ \|\boldsymbol{v}\|_1 &= \sum \|\boldsymbol{f}\|_1,\end{aligned} \tag{7}$$

where ε means regularization constants (here, $\varepsilon = 1$) to support empty histograms and it is known to have little impact on final results over a large range [2].

The final step is feature binarization. As a result of normalization, the normalized histogram has 72 real numbers. So the HOG features for $w \times h$ luminance image occupies approximately $\frac{w}{5} \times \frac{h}{5} \times 72 \times 8$ byte of memory capacity to store. This corresponds about 6.75 MB for VGA image, if we use the 8-byte double-precision floating point format for each value of histograms. This makes compact implementation with embedded hardware difficult. Therefore, we employed a binarized HOG scheme described in [14] to reduce the size of the features. In this scheme, each value of normalized histograms is binarized as:

$$v_b = \begin{cases} 1 \text{ if } v_n \geq t_b \\ 0 \textit{ otherwise}, \end{cases} \tag{8}$$

where v_b means a binarized result of a value v_n in the normalized histogram and t_b means a scalar threshold value. The binarized histogram $\boldsymbol{v}_b$ is generated by applying the binarization to all the values within the normalized histogram $\boldsymbol{v}_n$. With this reduction scheme, the memory capacity required to store the HOG features for luminance of $w \times h$ is reduced to $\frac{w}{5} \times \frac{h}{5} \times 72$ bits. Compared with the original size of $\frac{w}{5} \times \frac{h}{5} \times 72 \times 8$ bytes, $\frac{1}{64}$ of reduction is achieved.

In the boundaries of a given image, we have to treat special cases for generation of both cells and blocks. In this implementation, we ignored cells and blocks which cover the outside of the image. Therefore the numbers of cells and blocks are $\left(\frac{w}{5} - 2\right) \times \left(\frac{h}{5} - 2\right)$.

2.2 AdaBoost Classifiers

AdaBoost is a machine learning method that combines multiple weak classifiers, each of which only returns a true or false, so that an effective strong classifier is constructed [3]. In the training phase, positive sample images and negative sample images are repeatedly used by changing their weights, to select appropriate weak classifiers.

Since generation of classifiers using sample images is an offline process, we implemented AdaBoost classifier generator with software. In this implementation, HOG features that frequently appear in human sample images (positive samples) and rarely observed in other images (negative samples), were employed for weak classifiers. In addition, the block coordinates of such HOG features exist were also utilized. In AdaBoost method, one training phase generates one weak classifier as:

$$C_w = \{\boldsymbol{H}, \boldsymbol{P}\}, \tag{9}$$

where $\boldsymbol{H}$ means the feature pattern and $\boldsymbol{P}$ means coordinate of the classifier. Given a binarized histogram $\boldsymbol{v}_b$ as a HOG feature, our weak classifier C_w returns true if any one of the nine binarized feature vectors for the cells within a block is exactly matched with the feature $\boldsymbol{H}$ of the classifier. The strong classifier C_s constructed through N_c times of the training can be expressed as:

$$C_s = \{C_{w1}, C_{w2}, \ldots, C_{wi}\}. \tag{10}$$

Note that since duplicated weak classifiers are eliminated, the number of weak classifiers in one strong classifier is not always equal with the number of trainings. An example of a strong classifier constructed with three training phases is shown in Fig. 4.

The strong classifier counts up the number of HOG features which weak classifiers return true in a $w_c \times h_c$ detection window. After that the region that surrounded by the detection window is identified as a human image region if enough number of weak classifiers return true ($\geq t_c$).

The detection window moves the entire image from the upper left corner in a block-by-block raster scan manner. The required number of window scans for $w_b \times h_b$ blocks using $w_c \times h_c$ window is $(w_b - w_c + 1) \times (h_b - h_c + 1)$. Therefore,

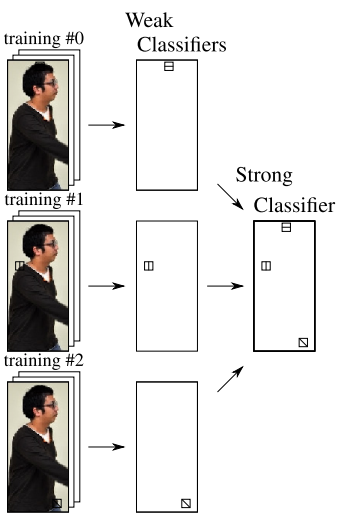


Fig. 4 Strong classifier generation by three times training.

the total number of matching processes required by the N_c weak classifiers is $(w_b - w_c + 1) \times (h_b - h_c + 1) \times N_c$.

3. Simplification of the HOG Feature Process for Hardware

Some of the processes of the HOG feature extraction described in the previous section consist of mathematical functions such as a trigonometric function and division. These functions make our design difficult to fit in small FPGAs. Thus, we introduce approximation schemes to reduce the calculation complexity.

3.1 Approximation for Gradient Orientations

Our first approximation is for choosing the best bin b_n according to a given gradient orientation θ as described in [10]. A naive scheme requires computation of the arc tangent function as shown in Eq. (3). Since we only need to choose the best bin b_n to vote from eight bins, we can introduce a more compact scheme to compute Eq. (3) and Eq. (4). The best bin b_n for given orientation θ can be defined as:

$$g_x \cdot \tan\left(\frac{n}{8}\pi - \frac{\pi}{16}\right) \le g_y < g_x \cdot \tan\left(\frac{n}{8}\pi + \frac{\pi}{16}\right). \qquad (11)$$

Equation (11) allows us to choose the bin using simpler functions. Figure 5 shows pseudo code for simplified computation of the bin selecting. This scheme only requires four multiplication with constants, four comparisons, comparison of signs, subtraction and modulo operation.

3.2 Approximation for Normalization of Histogram

The second approximation is for normalization process in Eq. (7). Kadota *et al*. introduced an approximation scheme for L2-norm normalization in [11]. In this scheme, divisors for the normalization are approximated to power-of-two values, so that the division is replaced by a shift operation. We enhanced this approximation and applied it in Eq. (7).

If the denominator of Eq. (7) is approximated to 2^α such that $2^{\alpha-1} < (\|\boldsymbol{v}\|_1 + \varepsilon) \le 2^\alpha$, the division for the normalization can be replaced by a shift operation. However, this naive approximation to the nearest power-of-two value increases the normalization error. To mitigate the error, we used sums of the number of the form $1/2^k$, like $1/2^k + 1/2^l$.

The original interval $(2^{\alpha-1}, 2^\alpha]$ is divided into n sub-intervals; $\left(2^{\alpha-1}, (1+\frac{1}{n})2^{\alpha-1}\right]$, $\left((1+\frac{1}{n})2^{\alpha-1}, (1+\frac{2}{n})2^{\alpha-1}\right]$, ..., $\left((1+\frac{n-1}{n})2^{\alpha-1}, 2^\alpha\right]$, and shift amounts for each interval are precomputed. Figure 6 shows pseudo code for the approximation we used. In the approximation, the minimum α which meets the condition $(\|\boldsymbol{v}\|_1 + \varepsilon) \le 2^\alpha$ is first computed. Then an appropriate interval is chosen from n sub-intervals, and finally each value of normalized histogram is computed by shift and addition. We divided the original interval into four sub-intervals ($n = 4$) and used three kinds of power of two values in this implementation.

Figure 7 shows comparison results of approximation errors between our scheme and the naive power-of-two scheme, in the case that a numerator is 361 in Eq. (7). The results show that the normalization errors are effectively reduced with the relatively simple additional computation processes.

```
if (|g_x| · tan(7/16 π) ≤ |g_y|) then
    n ← 4
else if (|g_x| · tan(5/16 π) ≤ |g_y|) then
    n ← 3
else if (|g_x| · tan(3/16 π) ≤ |g_y|) then
    n ← 2
else if (|g_x| · tan(1/16 π) ≤ |g_y|) then
    n ← 1
else
    n ← 0
end if
if (sign(g_x) ≠ sign(g_y)) then
    n ← (8 − n) (mod 8)
end if
```

Fig. 5 Simplified bin selection.

```
α ← ⌈log_2 (||v||_1 + ε)⌉
if ((1 + 3/4)2^{α−1} < ||v||_1 + ε) then
    v_n ← f/2^α
else if ((1 + 2/4)2^{α−1} < ||v||_1 + ε) then
    v_n ← f/2^α + f/2^{α+2}
else if ((1 + 1/4)2^{α−1} < ||v||_1 + ε) then
    v_n ← f/2^α + f/2^{α+1}
else
    v_n ← f/2^α + f/2^{α+1} + f/2^{α+2}
end if
```

Fig. 6 Simplified histogram normalization.

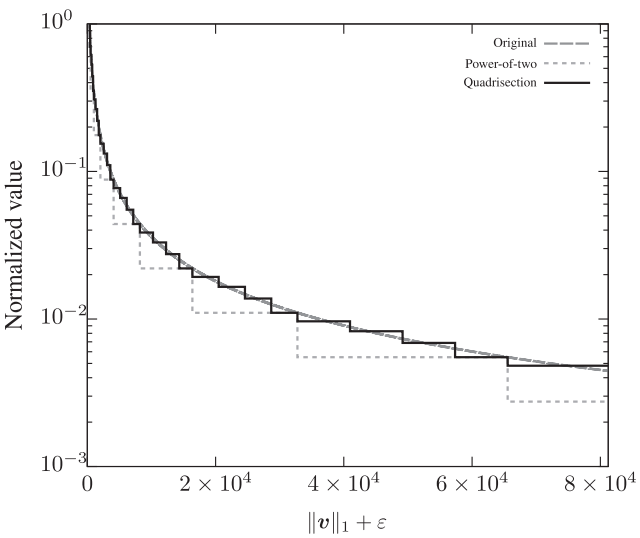


Fig. 7 Normalization errors.

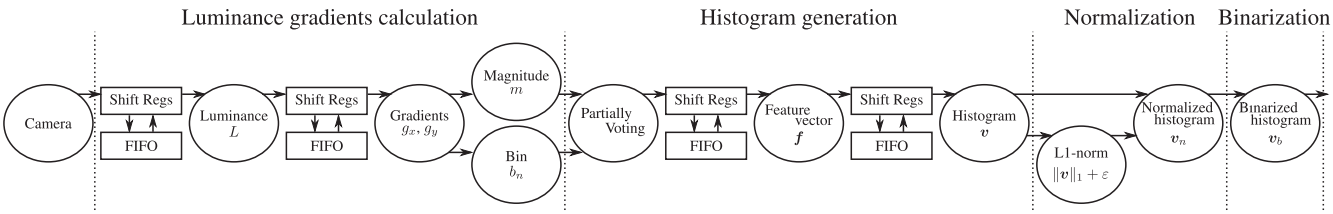


Fig. 9 Overview of HOG feature extraction pipeline.

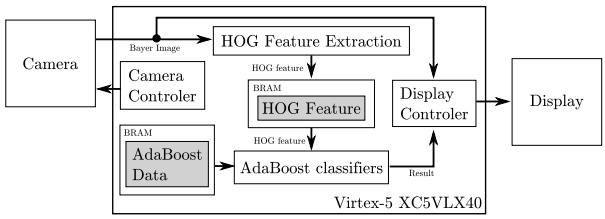


Fig. 8 Overview of our architecture for human detection. Gray boxes show what data are stored on BRAM.

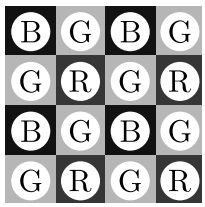


Fig. 10 Bayer pattern of camera images. Each alphabet means valid color channel at each location.

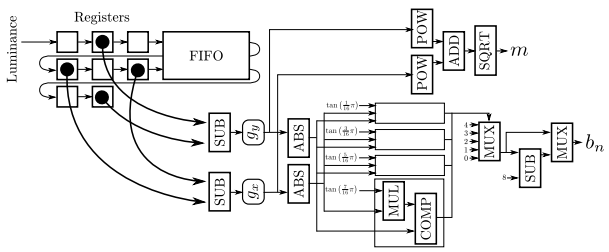


Fig. 11 Streamed architecture for computation of the gradient magnitude and the bin.

4. Implementation

Figure 8 shows an overview of our human detection system and Fig. 9 shows the pipeline structure for HOG feature extraction. The camera module outputs bayer-patterned image data sequentially in a pixel-by-pixel manner, and these data are directly passed to the HOG feature extraction pipeline. Then, extracted HOG features are stored in on-chip BRAMs and are used by the human detection module consisting of AdaBoost classifiers. Decision criteria for each weak classifier (AdaBoost data) are provided by on-chip ROM also implemented with BRAMs. Finally, detection results are indicated with markers on output images and outputted to an external display.

4.1 Luminance

We used an OmniVision Technologies OV9620 CMOS camera as an input device. Since this device produces a raw 8-bit Bayer pattern image consisting of 640×480 valid pixels as shown in Fig. 10, we implemented 2×2-pixel filter to convert a bayer-patterned image to a full color image. Then, luminance values are calculated from the full color image using Eq. (1). This filter can be implemented by the streamed architecture shown in Fig. 1 with 2×2 of 8-bit registers and a FIFO to store 638 of 8-bit pixels. As a result, a gray scale image of 320×240 of 8-bit luminance values is produced from an input image.

4.2 Luminance Gradient

As shown in Fig. 11, calculation of central-differences of luminance g_x and g_y in Eq. (2) requires the streamed architecture with 3×3 of 8 bit-registers and a FIFO to store two lines. In a pipeline part, two subtractors for 8-bit integers is needed for computing both g_x and g_y. Computation of the gradient magnitude m in Eq. (3) requires two multiplier for 8-bit unsigned integers ($|g_x|$ and $|g_y|$), an adder for 17-bit unsigned integers and a square root operator for 17-bit unsigned integers. Since the maximum value of the gradient magnitude is 361 ($= \lceil \sqrt{255^2 + 255^2} \rceil$), m is expressed as a 9-bit unsigned integer. The bin b_n to be vote can be chosen with the algorithm shown in Fig. 5. In order to simplify the implementation, we employed fixed-point arithmetic with a 10-bit fraction part for $|g_x|$, $|g_y|$ and results of tangent functions. As a result, the computation for the bin b_n requires four multipliers for $|g_x|$ and 10-bit constant unsigned integers, four comparators for 18-bit unsigned integers and other small operators. The bin b_n is expressed as a 3-bit unsigned integer.

Multipliers and a square root operator were generated by Xilinx CORE Generator. Note that all the module is fully-pipelined to compute luminance gradients in the same rate with the camera interface.

4.3 Histogram Generation for Cells

We also took a stream processing approach for the histogram generation. Since each cell is not overlapped with others, histograms do not have to be computed every clock cycle. Thus we need not to handle 25 gradients at the same time.

As shown in Fig. 12, the first partial histogram of gradient histograms for five consecutive luminance gradients

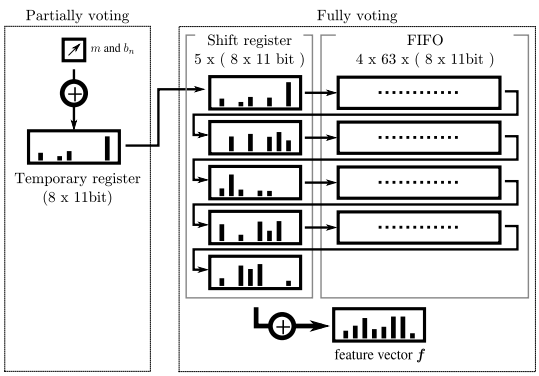


Fig. 12 Histogram generation for cells.

in horizontal direction is computed using temporary register in partially voating process. Then the stream of the partial histograms goes through FIFO so that partial histograms for five lines are eventually summed up to make the full histogram for the cell in fully voating process.

Since the gradient magnitude m is expressed with a 9-bit value, each orientation of a partial histogram can be expressed as a 11-bit unsigned integer. Thus, required resources for streaming are registers corresponds to 11 bits × 8 orientations × 5 lines and FIFOs corresponds to 11 bits × 8 orientations × 63 cells per line × 4 lines. The voter requires eight comparators for 3-bit unsigned integers and eight accumulators for 11-bit unsigned integers. Finally, full histogram consists of eight 14-bit unsigned integers.

4.4 Histogram Normalization in a Block

The normalization process is carried out for a moving 3×3 windows of cell histograms. Again, we can exploit the streamed structure. For this process, three lines of 3-stage shift registers and 2 lines of 61-stage FIFOs are used to store cell histograms. The histogram $\boldsymbol{v}$ is generated by concatenating all the cell histograms in a window.

Every time a new cell histogram is streamed in, all the 72 values of nine histograms in the 3×3-cell window are summed up to obtain a value of $\|\boldsymbol{v}\|_1$. This addition is done in two clock cycles to avoid degradation of the clock frequency. Since the maximum value of $\|\boldsymbol{v}\|_1$ is $81,225$, $\|\boldsymbol{v}\|_1 + \varepsilon$ can be expressed as a 17-bit unsigned integer when $\varepsilon \leq 49,847$, and the value of ε is '1' in this implementation.

In the next clock cycle, shift amounts for normalization are computed using the approximation scheme shown in Sect. 3.2. Figure 13 shows more hardware-oriented pseudo code for the approximation scheme. Lines from 1 to 4 of Fig. 13 can be implemented as a Look-Up-Table. Since a value of $\|\boldsymbol{v}\|_1 + \varepsilon$ is expressed with a 17-bit unsigned integer, α is expressed with a 5-bit unsigned integer. In lines 5-16, the shift amounts are computed with small shift operation and comparisons. Lines 6-7 are needed for intervals which are too narrow to divide.

In the 4th clock cycle, values of the normalized histogram are computed by three shift operations and two additions in line 17. Therefore, the normalization in a block is accomplished in four clock cycles.

```
1: α ← 0
2: while 2^α < (||v||_1 + ε) do
3:     α ← α + 1
4: end while
5: i ← (||v||_1 + ε) >> (α − 3)
6: if (α ≤ 2) then
7:     S_0 ← α, S_1 ← ∞, S_2 ← ∞
8: else if (i = 8) then
9:     S_0 ← α, S_1 ← ∞, S_2 ← ∞
10: else if (i = 7) then
11:    S_0 ← α, S_1 ← ∞, S_2 ← α + 2
12: else if (i = 6) then
13:    S_0 ← α, S_1 ← α + 1, S_2 ← ∞
14: else
15:    S_0 ← α, S_1 ← α + 1, S_2 ← α + 2
16: end if
17: v_n ← Σ_{i=0}^{2} (f >> S_i)
```

Fig. 13 Hardware oriented histogram normalization.

Since the maximum shift amount is 19, each value of histograms are temporarily expanded to a 33-bit fixed-point number with a 19-bit fraction part. As a result of the normalization, each value of histograms is expressed as a 19-bit fixed-point number with a 19-bit fraction part. Which means the normalized histogram for a block is expressed as 72 of 19-bit fixed-point numbers.

4.5 Histogram Binarization

As described in Sect. 2, the binarization process is relatively simple. The process requires 72 comparators for 19-bit fixed-pointer numbers, and thus we implemented in a combinational circuit. As a result of the binarization, 72-bit HOG feature for a block is extracted.

4.6 Data Stream of HOG Feature Extraction

As summarized in Fig. 9, the whole process flow of the HOG feature extraction is fully pipelined. All the HOG features obtained in this process flow are serially stored in on-chip RAMs. The on-chip RAMs can hold all the normalized HOG histograms of 62×46 blocks, which are obtained from a single frame image. Since each normalized HOG histograms is expressed as a 72-bit value, whole HOG feature for a single frame image occupies about 25 kBytes of the on-ship RAMs.

4.7 Human Detection using AdaBoost Classifiers

Figure 14 shows an overview of the human detection module using AdaBoost classifiers. The strong AdaBoost classifier C_s is stored in ROM which actually implemented with BRAM. Since each weak classifier is expressed as two 8-bit values for a feature pattern $\boldsymbol{H}$ and a block coordinates $\boldsymbol{P}$, N_c weak classifiers occupies approximately $2N_c$ byte of

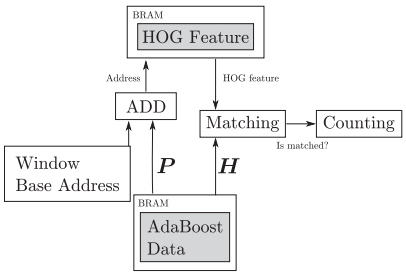


Fig. 14 Overview of human detection module using AdaBoost classifiers.

memory capacity to store. What the circuit for the strong classifier needs to do is simply to compare the weak classifiers with HOG features extracted from input images and to count the number of active classifiers.

Since each weak classifier corresponds to a difference block coordinate, random access to on-chip RAMs which stores HOG feature is needed in contrast to the streamed approach exploited in the HOG feature extraction.

In this implementation, we set the size of the detection window ($w_c \times h_c$) to 8×19 according to the size of training samples. While we executed 500 times training trials to construct a strong classifier, a total of 84 weak classifiers were eventually generated since we obtained a lot of duplications. Thus 168 bytes of memory capacity was required to store the strong classifier.

The total number of matching processes required for the 84 weak classifiers is $(62-8+1)\times(46-19+1)\times 84 = 129,360$. The camera device we used generates one frame image data in 400,000 clock cycles including synchronization intervals. Our implementation requires 385,452 clock cycles to extract whole HOG features for one frame data, while the AdaBoost detection process takes 129,360. To finish whole the detection process within 400,000 clock cycles, we overlapped the feature extraction and the human detection processes. As a result, our implementation processes whole computation for one frame in 387,820 clock cycles to enable in-frame real-time processing. Due to the remainder can process additional 7 weak classifiers, $\lfloor 12,180/(55\times 28)\rfloor = 7$, this result suggests the strong classifier can be constructed of up to 91 weak classifiers.

5. Implementation Results and Evaluation

The human detection process described in Sect. 4 was implemented on a Xilinx ML501 board equipped with a Virtex-5 XC5VLX50 FPGA. The design was described in Verilog HDL and a bitstream file was generated using Xilinx ISE design tools 13.4.

Table 1 shows implementation results of the design. While the maximum operating frequency of the design achieved 45 MHz, the camera device in our system restricts the system clock to 25 MHz. In spite of the restricted relatively-low clock frequency, our FPGA implementation achieved the throughput of 62.5 FPS for VGA frames and execution latency was also fitted in a single frame time,

Table 1 Resource utilization.

Resource	Used	Available	Percentage (%)
SLICE	6,607	7,200	91.8
FF	2,255	28,800	7.8
LUT	17,121	28,800	59.4
BRAM/FIFO	36	48	75.0
DSP48E	2	48	4.2

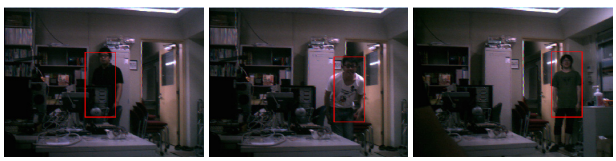


Fig. 15 Example results of human detection process.

that is, the real-time performance was accomplished. Furthermore, if a high-speed camera device were used and the FPGA design was operated with the maximum clock frequency of 45 MHz, the execution throughput would be improved up to 112.5 FPS. An adder tree for computing L1-norm in Eq. (7) lies on a critical path. Figure 15 illustrates examples of our experimentation results. These images were obtained by a monitoring mechanism we also implemented on the FPGA board, which allows us to transmit actual result image data to the host PC. The red frame markers were also generated by the FPGA circuit to display detected regions.

We evaluated the quality of results of our approximation scheme by comparing to the original floating-point arithmetic algorithm. We implemented two software simulators in C for both of the schemes and evaluated accuracy of human detection. In this evaluation, NICTA Pedestrian database [15] was used for benchmarking. To generate AdaBoost classifiers, 2,000 images from the database were used for offline machine learning, while other 1,000 images were used as the evaluation data. The size of each image is 64×80 pixels. As a result of 500 times of training, a strong AdaBoost classifier which consists of 84 weak classifiers were eventually generated. The threshold value for the histogram binarization was set to 0.04.

Figure 16 summarizes the results of comparison as a chart of receiver operator characteristics (ROC) curve. The chart shows the relationship between the false positive rate (x-axis) and the detection rate (y-axis) of the system. The closer to the upper left area of the chart means better quality of results. The plots were made by changing the threshold number of weak classifiers for detection from 0 to 30. As a result, the simplified scheme for hardware implementation shows 94.5 % of the detection rate with 15.7 % of the false positive rate, while the original one shows 95.6 % of the detection rate with 14.5 % of the false positive rate. Although the detection results for original scheme might be improved by tuning parameters such as the threshold value, the evaluation results suggest the negative impact of simplified scheme for hardware implementation is limited in terms of detection

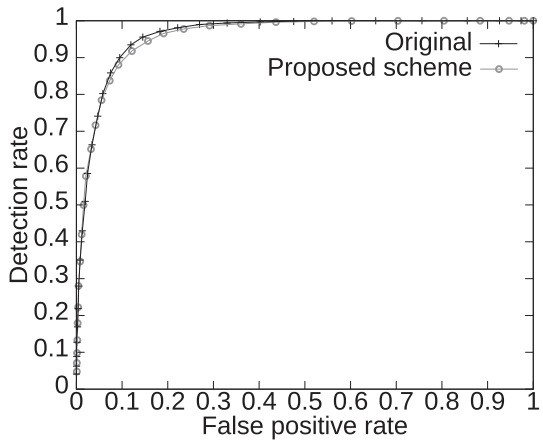


Fig. 16 ROC curves for the NICTA Pedestrian database.

rate.

Finally, we compared the throughput of our FPGA implementation to software implementation. The software implementation was compiled by gcc 4.3.1 and run on 2.67 GHz Intel Core i7 920 with 6 GB DDRIII operated by openSUSE 11.2. As a result, the software implementation achieve about 15 FPS which means the FPGA design is 4.2 times faster than the software implementation. Although another software implementation using SIMD instructions is reported to achieve 20 FPS [12], our FPGA design is still 3.13 times faster. Moreover, 7.5 times faster throughput to our software implementation is expected if the camera device operates at the maximum frequency of the design.

Although our external memory free architecture was shown to be efficient for the HOG-based human detection algorithm, this architecture will not be versatile for every image processing application. For example, use of an external frame buffer enables random access to image data and makes it easy to use a soft-core processor to execute a part of tasks. Frame buffers are also useful for introducing multiple clock domains in designs and absorbing differences in throughputs between the domains. On the other hand, applications that have a relatively simple control flow and regular data access patterns, especially a class of algorithms that use moving widow operators are good candidates for our architecture. Avoiding the use of frame buffers, we can reduce energy consumption for the external memories and off-chip data communications as well as implementation size, which is advantageous especially for embedded systems.

6. Conclusion

In this paper, compact FPGA implementation of real-time human detection using the HOG feature and AdaBoost classifier has been presented. As a result of evaluation, the throughput of 62.5 FPS was achieved without using any external memory modules. If a high-speed camera device was available, the maximum throughput of 112 FPS was expected to be accomplished. While some simplifications were introduced to reduce hardware complexity, the evaluation with ROC curves showed that the negative impact of the simplifications is limited.

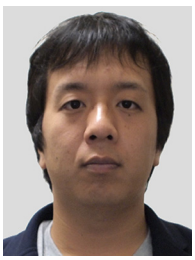
Our future work includes to implement more compact design to fit smaller and cheaper FPGA devices rather than Virtex families and construct human abnormal behavior detector combining with another feature detection like CHLAC approach [16].

References

[1] K. Negi, K. Dohi, Y. Shibata, and K. Oguri, "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm," Proc. Int. Conf. Field-Programmable Technology, pp.1–8, Dec. 2011.
[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, pp.886–893, 2005.
[3] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," Computational Learning Theory, pp.23–37, Springer, 1995.
[4] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," Computer Vision–ECCV 2006, pp.428–441, 2006.
[5] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pp.304–311, 2009.
[6] C. Kuo and R. Nevatia, "Robust multi-view car detection using unsupervised sub-categorization," Applications of Computer Vision (WACV), 2009 Workshop on, pp.1–8, 2009.
[7] H. Matsubayashi, S. Nino, T. Aramaki, Y. Shibata, and K. Oguri, "Retrieving 3-d information with FPGA-based stream processing," Proc. 16th ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp.261–261, Feb. 2008.
[8] K. Dohi, Y. Yorita, Y. Shibata, and K. Oguri, "Pattern compression of FAST corner detection for efficient hardware implementation," Proc. IEEE 21st Int. Conf. Field Programmable Logic and Applications, pp.478–481, Sept. 2011.
[9] K. Dohi, Y. Hatanaka, K. Negi, Y. Shibata, and K. Oguri, "Deep-pipelined fpga implementation of ellipse estimation for eye tracking," Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on, pp.458–463, 2012.
[10] T. Cao and G. Deng, "Real-time vision-based stop sign detection system on fpga," Computing: Techniques and Applications, 2008. DICTA'08. Digital Image, pp.465–471, 2008.
[11] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, "Hardware architecture for hog feature extraction," Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP'09. Fifth International Conference on, pp.1330–1333, 2009.
[12] M. Komorkiewicz, M. Kluczewski, and M. Gorgon, "Floating point hog implementation for real-time multiple object detection," Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on, pp.711–714, 2012.
[13] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural study of HOG feature extraction processor for real-time object detection," Signal Processing Systems (SiPS), 2012 IEEE Workshop on, pp.197–202, 2012.
[14] Y. Yamauchi, C. Matsushima, T. Yamashita, and H. Fujiyoshi, "Relational hog feature with wild-card for object detection," Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, pp.1785–1792, 2011.
[15] G. Overett, L. Petersson, N. Brewer, L. Andersson, and N. Pettersson, "A new pedestrian dataset for supervised learning," Intelligent Vehicles Symposium, 2008 IEEE, pp.373–378, 2008.
[16] N. Otsu, "Chlac approach to flexible and intelligent vision systems,"

Bio-inspired Learning and Intelligent Systems for Security, 2008. BLISS'08. ECSIS Symposium on, pp.23–33, 2008.

Keisuke Dohi received B.E. and M.E. degrees from Nagasaki University, Japan, in 2009 and 2011, respectively. His research interests include reconfigurable system and GPGPU computing.



Kazuhiro Negi received B.E. and M.E. degrees from Nagasaki University, Japan, in 2010 and 2012, respectively. His research interests include image enhancement and electronic control systems.

Yuichiro Shibata received the B.E. degree in electrical engineering, the M.E. and Ph.D. degrees in computer science from Keio University, Japan, in 1996, 1998 and 2001, respectively. Currently, he is an associate professor at Department of Computer and Information Sciences, Nagasaki University. He was a Visiting Scholar at University of South Carolina in 2006. His research interests include reconfigurable systems and parallel processing. He won the Best Paper Award of IEICE in 2004.

Kiyoshi Oguri received B.S. and M.S. degrees in physics from Kyushu University, Japan, in 1974 and 1976, respectively. He also received the Ph.D. degree in information engineering from the same university in 1997. Since joining NTT in 1976, he have been engaged in the research, design and development of high-end general purpose computer, high-level logic synthesis system and a wired logic-based dynamic computing architecture. Currently, he is a professor of Nagasaki University, Japan. His research interests are in hardware modeling, high-level synthesis, FPGA-related systems, and Plastic Cell Architecture. Prof. Oguri received the Motooka Prize in 1987, the Best Paper Award of IPSJ in 1990, the Okochi Memorial Technology Prize in 1992, the Achievement Award of IEICE in 2000, and the ACM Gordon Bell Prize in 2009.