Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm

Kazuhiro Negi, Keisuke Dohi, Yuichiro Shibata, Kiyoshi Oguri

Graduate School of Science and Thechnology, Nagasaki University

{negi,dohi,shibata,oguri}@pca.cis.nagasaki-u.ac.jp

***Abstract*—In this paper, deep pipelined FPGA implementation of a real-time image-based human detection algorithm is presented. By using binary patterned HOG features, AdaBoost classifiers generated by offline training, and some approximation arithmetic strategies, our architecture can be efficiently fitted on a low-end FPGA without any external memory modules. Empirical evaluation reveals that our system achieves 62.5 fps of the detection throughput, showing 96.6% and 20.7% of the detection rate and the false positive rate, respectively. Moreover, if a high-speed camera device is available, the maximum throughput of 112 fps is expected to be accomplished, which is 7.5 times faster than software implementation.**

I. Introduction

Real-time image-based human detection plays a key role in a wide range of application domains including robotics, security, surveillance, nursing, and entertainment. Also, the importance of compact and power-efficient implementation of the algorithm with embedded hardware is rising along with this expansion of application domains. FPGA implementation of such advanced video processing is a promising approach, but external memory is often required to store a video frame and data commutation between the external memory and the FPGA tends to harm the performance as well as energy-efficiency.

In this paper, we present external memory-free FPGA implementation of a real-time image-based human detection system. A process flow of image-based human detection generally consists of two stages; calculation of feature amount of given images and pattern classification based on a machine learning technology. In this implementation, histograms of oriented gradients (HOG)[1] is used as feature amount of images while AdaBoost classifiers[2] are employed for pattern recognition. Making the best use of deep pipelined arithmetic structure configured on an FPGA and a high bandwidth provided by on-chip RAMs, our streamed processing approach achieves real-time human detection for input video frames without any external memory modules. This external memory-free architecture is also suitable for implementation with a reconfigurable fabric on an SoC.

So far, hardware implementation of HOG-based object detection has been actively investigated. In order to reduce the HOG feature amount and hardware size, an effective binarization scheme of HOG feature was proposed [3]. In [4], efficient FPGA implementation of real-time HOG feature extraction was presented and application for a stop sign detection system was shown in [5]. Our contribution is to present a unified pipelined architecture of not only the HOG feature extraction but also AdaBoost for real-time human detection on a single FPGA without external memory.

The rest of this paper is organized as follows. Section II explains fundamentals of the HOG feature extraction and Section III shows reduction techniques of the calculation amount for efficient implementation of human detection on an FPGA. Then, Section IV explains FPGA implementation technique with on-chip Block RAMs and shift registers. After evaluation of the proposed architecture is presented in Section V, finally the paper is summarized in Section VI.

II. HOG feature

The histograms of oriented gradients (HOG) uses local histograms of oriented gradients of pixel luminance to characterize a given image. HOG description of a local region of the image roughly expresses object shape, thus the method is widely used for various object recognition such as pedestrian detection and car detection [1], [6], [7], [8]. The process of HOG feature extraction consists of three stages:

1) Luminance is calculated from given color signals, and the gradient strengths and the gradient directions are calculated from the luminance map. (Section II-A)
2) Histograms of oriented gradients are calculated by summarizing the gradient strength and direction for each group of pixels called a cell. (Section II-B)
3) The histograms are normalized for each group of cells called a block. (Section II-C)

In the followings, details of each process stage are explained.

A. Calculation of luminance gradients

The luminance value L for each pixel is calculated from the corresponding RGB values according to Eq. (1).

$$\begin{aligned} MAX &= \max(R,G,B) \\ MIN &= \min(R,G,B) \\ L &= \frac{(MAX+MIN)}{2} \end{aligned} \tag{1}$$

where R, G, B, and L take the value from 0 to 255.

Let $L(x,y)$ be the luminance of the pixel at the coordinate (x,y). The vertical and horizontal differences of luminance

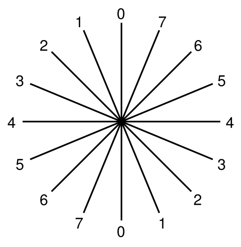

Fig. 1 Quantization of gradient direction θ

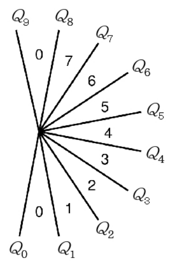

Fig. 2 Quantization of divided angles

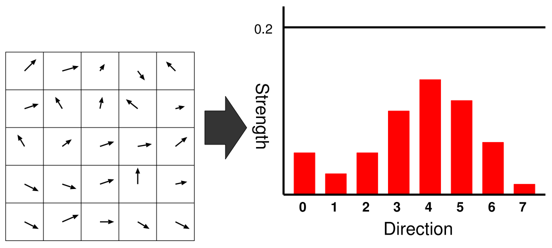

Fig. 3 The example of histogram generation

$\Delta L_x(x,y)$ and $\Delta L_y(x,y)$ are defined as shown in Eq. (2). Using these values, the gradient strength $m(x,y)$ and gradient direction $\theta(x,y)$ are calculated according to Eq. (3) and Eq. (4), respectively.

$$\begin{cases} \Delta L_x(x,y) = L(x+1,y) - L(x-1,y) \\ \Delta L_y(x,y) = L(x,y+1) - L(x,y-1) \end{cases} \quad (2)$$

$$m(x,y) = \sqrt{\Delta L_x(x,y)^2 + \Delta L_y(x,y)^2} \quad (3)$$

$$\theta(x,y) = \tan^{-1}\frac{\Delta L_y(x,y)}{\Delta L_x(x,y)} \quad (4)$$

B. Histogram generation

A histogram of the gradient orientations is generated for each cell, which consists of 5×5 luminance values, using the corresponding gradient strengths m and directions θ. The gradient directions are quantized to make histograms as shown in Figure 1.

Here, the gradient directions are labeled by eight quantized orientations according to the threshold angles illustrated in Figure 2. For example, when θ is within the range of $Q_2 \leq \theta < Q_3$, the corresponding orientation label becomes 2. Note that we only take account of the range from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ for θ, since the HOG method does not focus on gradient directions but orientations; the opposite direction means the same orientation. The threshold values $Q_0, Q_1, \ldots Q_9$ are given by Eq. (5).

$$Q_d = \tan^{-1}((7-2d) \times \frac{\pi}{16}) \quad (d = 0, 1, ..., 9) \quad (5)$$

Then the gradient strengths m are voted according to the corresponding quantized orientation label for every luminance value in a cell to make the histograms, as shown in Figure 3. Since we quantize the gradient orientation into eight labels, 8-dimension feature vectors are eventually generated. For a $(w \times h)$ input image, a total of $\frac{w}{5} \times \frac{h}{5}$ histograms are obtained since each cell consists of 5×5 luminance values.

C. Block normalization

The histogram of the gradient orientations obtained from cells are normalized in terms of a block, which consists of 3×3 cells. Let:

$$F_{i,j} = [f^0{}_{i,j}, f^1{}_{i,j}, f^2{}_{i,j}, f^3{}_{i,j}, f^4{}_{i,j}, f^5{}_{i,j}, f^6{}_{i,j}, f^7{}_{i,j}] \quad (6)$$

be the feature vector for the cell located at the i-th row and j-th column.

Since one block contains feature vectors for nine cells, the feature vector of the block whose upper-left corner cell is at the i-th row and j-th column can be expressed as:

$$V_{i,j} = [F_{i,j}, F_{i+1,j}, F_{i+2,j}, F_{i,j+1}, F_{i+1,j+1}, F_{i+2,j+1}, F_{i,j+2}, F_{i+1,j+2}, F_{i+2,j+2}] \quad (7)$$

which has $8 \times 9 = 72$ dimensions.

Let $f^m{}_{k,l}$ denote a one-dimension feature amount in $V_{i,j}$ where $i \leq k \leq i+2$ and $j \leq l \leq j+2$. Then, the normalized feature amount $v^m{}_{k,l}$ is calculated by Eq. (8).

$$\begin{aligned} v^m{}_{k,l} &= \frac{f^m{}_{k,l}}{\sqrt{\|V_{i,j}\|^2 + \varepsilon^2}} \quad (\varepsilon = 1) \\ \|V_{i,j}\| &= \|F_{i,j}\| + \|F_{i+1,j}\| + \cdots + \|F_{i+2,j+2}\| \\ \|F_{i,j}\| &= |f^0{}_{i,j}| + |f^1{}_{i,j}| + \cdots + |f^7{}_{i,j}| \end{aligned} \quad (8)$$

When the HOG features are extracted from an image of $w \times h$ luminance values, we have a total of $\frac{w}{5} \times \frac{h}{5}$ cells as described in the previous subsection. A block consists of 3×3 cells and scans the entire image area in a cell-by-cell manner, so that we have a total of $(\frac{w}{5}-2)\times(\frac{h}{5}-2)$ blocks. This means a feature vector with $(\frac{w}{5}-2)\times(\frac{h}{5}-2)\times 72$ dimensions is obtained from a single image data.

D. Binarization of features

If we express the HOG feature with the 8-byte double-precision floating point format for each dimension, approximately $(\frac{w}{5}-2)\times(\frac{h}{5}-2)\times 72 \times$ 8-byte memory capacity will be required to store the whole HOG features for a $(w \times h)$ single image, making compact implementation with embedded hardware difficult. To reduce the size of the features, we employed a binarized HOG scheme proposed in [3].

In this scheme, each one-dimension feature amount is binarized using a threshold value so that each gradient orientation can be expressed in a single bit. Since the gradient directions are quantized into eight orientations, a feature amount obtained from a cell can be described with eight bits, that is, 0 to 255 in decimal.

Figure 4 depicts an example of the HOG feature binarization. In this case, the feature amounts for three of the eight orientations exceed the threshold value, generating the 8-bit binary pattern of 00111000 (56 in decimal) as the feature vector for the cell. With this reduction scheme, the memory capacity required to store the HOG features for an image of $w \times h$ is reduced to $(\frac{w}{5}-2)\times(\frac{h}{5}-2)\times 72$-bits. Compared to the

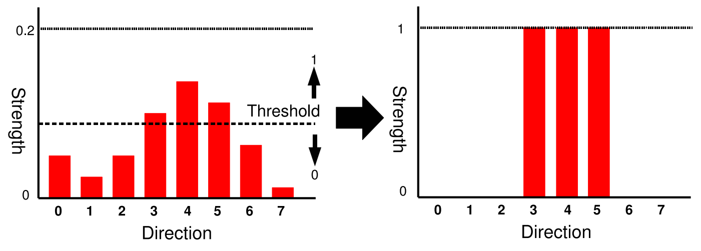

Fig. 4 The example of binarization

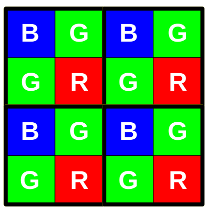

Fig. 5 bayer-pattern

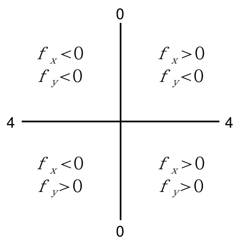

Fig. 6 Orientations and sings of $\Delta L_x(x, y)$ and $\Delta L_y(x, y)$

original size of $(\frac{w}{5}-2)\times(\frac{h}{5}-2)\times 72\times 8$-bytes, $\frac{1}{64}$ of reduction is achieved.

III. Simplification of the calculation process for hardware implementation

The calculation process of the HOG features described in the previous section is still too complicated to directly implement as a simple FPGA circuit. Thus, we have further modified the process to reduce the calculation complexity.

A. *Calculation of luminance gradient*

The calculation of a luminance value L is executed with integer arithmetic by rounding a result value of Eq. (1). For this calculation step, a gradient strength m and a gradient orientation θ are obtained after calculating pixel luminance. Thus, L, $\Delta L_x(x, y)$ and $\Delta L_y(x, y)$ are expressed as 8-bit integer values.

In Eq. (3), in which the gradient strength m is obtained, both the values of $\Delta L_x(x, y)$ and $\Delta L_y(x, y)$ are squared and thus unsigned arithmetic can be carried out. By rounding down the fraction part of the result, m takes a value in the range of 0 to 361 and can be expressed as 9-bit integer value.

Since the values of $\Delta L_x(x, y)$ and $\Delta L_y(x, y)$ have already been calculated in the previous steps, the value of $\tan\theta$ can be obtained by Eq. (9). On the other hand, the tangent values of the quantization threshold angles $Q_d(d = 1, 2, \ldots, 8)$, which are described in Section II (Figure 2), can be precomputed since they are constant values. Therefore, we get a quantized gradient orientation by the conditional expression with $\tan Q_d$ and $\tan Q_{d+1}$ as shown in Eq. (10).

$$\tan\theta = \frac{\Delta L_y(x, y)}{\Delta L_x(x, y)} \tag{9}$$

$$\tan Q_{d+1} \le \tan\theta < \tan Q_d \ (d = 1, 2, \ldots, 8) \tag{10}$$

Here, by substituting Eq. (9) into Eq. (10), we we obtain Eq. (11), and additional expansion of the equation produces Eq. (12).

$$\tan Q_{d+1} \le \frac{\Delta L_y(x, y)}{\Delta L_x(x, y)} < \tan Q_d \tag{11}$$

$$\Delta L_x(x, y)\times\tan Q_{d+1} \le \Delta L_y(x, y) < \Delta L_x(x, y)\times\tan Q_d \tag{12}$$

In order to simplify the implementation, we employed fixed-point arithmetic with a 10-bit fraction part. This arithmetic system is equivalent to integer arithmetic in which all the values are multiplied by 1,024 (left-shifted by 10 bits). For example, in the case of $d = 3$ ($\tan Q_3 = 0.6681$ and $\tan Q_4 = 0.1989$), the condition of Eq. (12) is implemented as Eq. (13).

$$\Delta L_x(x, y)\times 204 \le \Delta L_y(x, y)\times 1024 < \Delta L_x(x, y)\times 684 \tag{13}$$

Here, we can observe the relationship between the gradient orientation and the signs of $\Delta L_x(x, y)$ and $\Delta L_y(x, y)$ as illustrated in Figure 6. When $\Delta L_x(x, y)$ and $\Delta L_y(x, y)$ have the same sign, the quantized gradient orientation fits in the range of 0 to 4. Otherwise, the quantized gradient orientation task the value of 4 to 7, and 0.

Based on this observation of Figure 6, we have derived the conditional expression described in Figure 7. Using this conditional expression, a quantized gradient orientation θ_d can be obtained from $\Delta L_x(x, y)$ and $\Delta L_y(x, y)$ without calculating arc tangent. Since a quantized gradient orientation is labeled by a value from 0 to 7, the orientation θ_d can be eventually coded in 3-bit data.

B. *Histogram generation for cells*

Since quantization has been made in the calculation step of a gradient orientation θ_d as mentioned above, any additional quantization process is not required when histograms are made.

We have 320×240 values of luminance gradient, which gives us a cell array with the width of $\frac{320}{5} = 64$ cells and the height of $\frac{240}{5} = 48$ cells. Since the maximum value of the gradient strength $F_{i,j}$ is $361\times 25 = 9{,}025$ for a histogram of a single cell, data size for one cell becomes 14 bits ×8. Therefore, total histograms of luminance gradient for the entire single image are expressed with $(14\times 8)\times(64\times 48) = 344{,}064$ bits.

C. *Normalization in a block*

The normalization process described in Eq. (8) needs calculation of a square root and division, making compact FPGA implantation difficult. Therefore, we take an approximation approach which expands the method proposed in [4].

If the denominator of Eq. (8) ($\sqrt{\|V_{i,j}\|^2+\varepsilon^2}$) is approximated by 2^α as $2^{\alpha-1} < \sqrt{\|V_{i,j}\|^2+\varepsilon^2} \le 2^\alpha$, the division for the normalization can be replaced by a shift operation. However,

```
if ($\Delta L_x(x,y)$ and $\Delta L_y(x,y)$ have the same sign) then
    if ($5148 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)|$) then
        $\theta_d = 0$
    else if ($1533 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)| < 5148 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 1$
    else if ($684 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)| < 1533 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 2$
    else if ($204 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)| < 684 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 3$
    else if ($0 \leq 1024 \times |\Delta L_y(x,y)| < 204 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 4$
    end if
else if (Signs of $\Delta L_x(x,y)$ and $\Delta L_y(x,y)$ are different) then
    if ($5148 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)|$) then
        $\theta_d = 0$
    else if ($1533 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)| < 5148 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 7$
    else if ($684 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)| < 1533 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 6$
    else if ($204 \times |\Delta L_x(x,y)| \leq 1024 \times |\Delta L_y(x,y)| < 684 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 5$
    else if ($0 \leq 1024 \times |\Delta L_y(x,y)| < 204 \times |\Delta L_x(x,y)|$) then
        $\theta_d = 4$
    end if
end if
```

Fig. 7 Conditional expression to obtain the value of quantized θ_d

naive approximation to the nearest power-of-two value increases the normalization error. To mitigate the approximation error, we divided the interval between $2^{\alpha-1}$ and 2^{α} into four sub-intervals.

The range of $2^{\alpha-1}$ to 2^{α} can be divided into four intervals; $(2^{\alpha-1}, 2^{\alpha-1} + \frac{2^{\alpha-1}}{4}]$, $(2^{\alpha-1} + \frac{2^{\alpha-1}}{4}, 2^{\alpha-1} + 2 \times \frac{2^{\alpha-1}}{4}]$, $(2^{\alpha-1} + 2 \times \frac{2^{\alpha-1}}{4}, 2^{\alpha-1} + 3 \times \frac{2^{\alpha-1}}{4}]$, and $(2^{\alpha-1} + 3 \times \frac{2^{\alpha-1}}{4}, 2^{\alpha}]$. Here, $\varepsilon = 1$ and this is significantly smaller than $||V_{i,j}||^2$. Therefore, the conditional statements in Figure 8 can be derived when we think: $\sqrt{||V_{i,j}||^2 + \varepsilon^2} \fallingdotseq ||V_{i,j}||$.

Figure 9 shows comparison results of approximation errors for our quadrisection approach and the naive power-of-two approach, in the case of ${f^m}_{k,l} = 361$. The results show that the normalization errors are effectively reduced with the relatively simple calculation process.

Since the maximum value of $||V_{i,j}||$ is 81,225, the maximum number of shift operations for the division is 19. Thus, as a fraction part, 19 bits of 0s are appended to the LSB side of ${f^m}_{k,l}$ in advance of shifting, and obtained ${v^m}_{k,l}$ is also expressed with a fixed point arithmetic number with 14 + 19 = 33 bits.

D. Binarization of features

We set the value of the binarization threshold as 0.08 based on results of our preliminary experimentation. Since one block

```
if ($2^{\alpha-1} < ||V_{i,j}|| \leq 2^{\alpha-1} + \frac{2^{\alpha-1}}{4}$) then
    $v = \frac{f}{2^{\alpha}} + \frac{f}{2^{\alpha+1}} + \frac{f}{2^{\alpha+2}}$
else if ($2^{\alpha-1} + \frac{2^{\alpha-1}}{4} < ||V_{i,j}|| \leq 2^{\alpha-1} + 2 \times \frac{2^{\alpha-1}}{4}$) then
    $v = \frac{f}{2^{\alpha}} + \frac{f}{2^{\alpha+1}}$
else if ($2^{\alpha-1} + 2 \times \frac{2^{\alpha-1}}{4} < ||V_{i,j}|| \leq 2^{\alpha-1} + 3 \times \frac{2^{\alpha-1}}{4}$) then
    $v = \frac{f}{2^{\alpha}} + \frac{f}{2^{\alpha+2}}$
else if ($2^{\alpha-1} + 3 \times \frac{2^{\alpha-1}}{4} < ||V_{i,j}|| \leq 2^{\alpha}$) then
    $v = \frac{f}{2^{\alpha}}$
end if
```

Fig. 8 Normalization conditional statement to range of $2^{\alpha-1} < ||V_{i,j}|| \leq 2^{\alpha}$

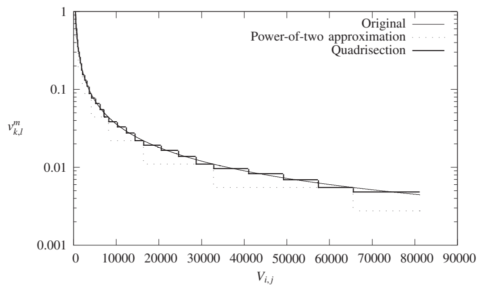

Fig. 9 Evaluation of the HOG normalization errors

consists of 9 cells, feature amount extracted from one block has $8 \times 9 = 72$ dimensions and is expressed as 72-bit data due to the binarization. As described in Section II-C, the number of blocks are less by two than those of cells for both the horizontal and vertical orientations. Therefore, the total number of blocks to be processed for a single image is $62 \times 46 = 2,852$.

IV. FPGA implementation of human detection

A. Input image

In this implementation, an OmniVision Technologies OV9620 CMOS camera was used as an input device. Since this device produces a raw Bayer pattern image consisting of 640×480 pixels as shown in Figure 5, a 2×2-pixel filter was implemented to extract RGB values, resulting in gray scale images of 320×240 luminance value. So, (w, h) is (320,240).

B. Calculation of luminance gradient

As described in Section IV-A, a luminance value L is obtained from 2×2 raw pixel values in the Bayer pattern. Therefore, we provide a 640-pixel line buffer with Block RAM. While pixel values of even lines are stored in the line buffer, L is calculated when pixels of odd lines are given. As shown in Figure 10, L can be calculated from three pixel values in one clock cycle.

The calculation of luminance gradient needs the luminance values of adjacent four luminance values. Therefore, we use

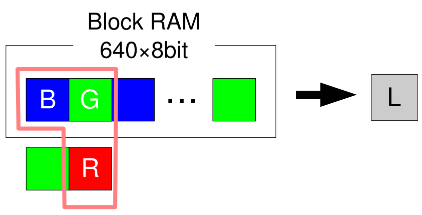

Fig. 10 Calculation of pixel luminance L

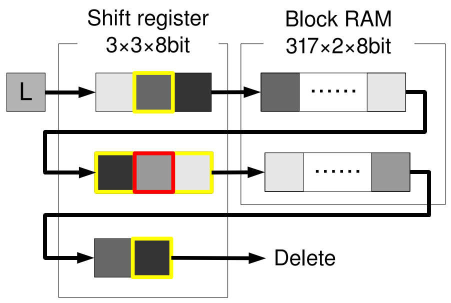

Fig. 11 Streamed structure for the calculation of luminance gradient

three lines of 3-stage shift registers and two lines of Block RAM buffers for 317 luminance values as illustrated in Figure 11.

Arithmetic units of the square and square root operations required for gradient strength m were synthesized by Xilinx CORE Generator so that the both units have the execution latency of one clock cycle, respectively. For the square unit, Block RAM was used for a look-up table. The gradient orientation θ_d is also calculated in one clock cycle in the way described in Section III-A.

C. Histogram generation for cells

We also take a stream processing approach for the histogram generation as shown in Figure 12. A histogram for a cell is obtained from a total of 25 (5×5) values of luminance gradient (m and θ_d). Partial histogram of gradient orientations for five consecutive luminance values is made every five luminance values. Then the stream of the partial histograms goes through Block RAM line buffers so that partial histograms for five lines are eventually summed up to make the full histogram for the cell.

Since the maximum value for the gradient strength is 361, each orientation of a partial histogram of five luminance values can be expressed with 11 bits. Thus, the required capacity of the Block RAM buffers corresponds to 11 bits × 8 orientations × 63 cells × 4 lines. Since the luminance values are streamed in every two clock cycles when the Bayer filter is active, full histograms are streamed out every 10 clock cycles.

D. Normalization in a block

The normalization process is carried out for a moving 3×3 window of cell histograms. Again, we can exploit streamed structure as show in Figure 13. For this process, three lines of 3-stage shift registers and 2 lines of 61-stage Block RAM buffers to store cell histograms.

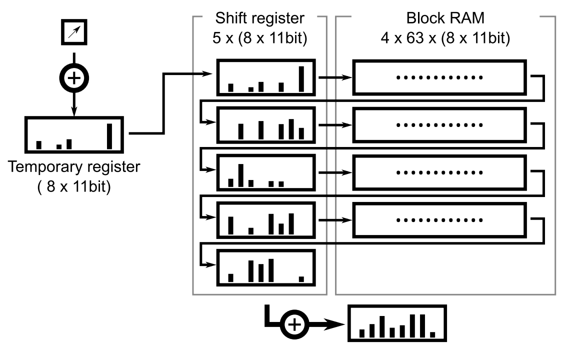

Fig. 12 Streamed structure for the histogram generation

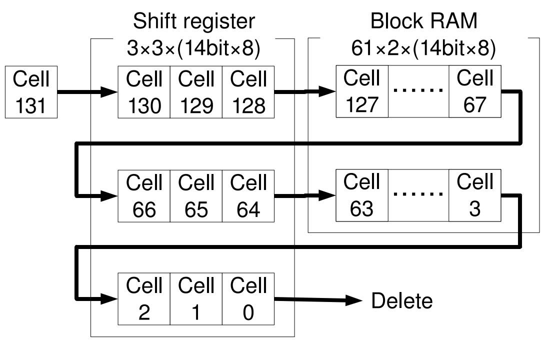

Fig. 13 Streamed structure for the histogram normalization

Every time a new cell histogram is streamed in, 72 dimensions of histogram elements in the 3x3-cell window are summed up to obtain a value of $||V_{i,j}||$. This addition is done in two clock cycles to avoid degradation of the clock frequency.

In the next clock cycle, right-shift amounts for the normalization are derived from the value of $||V_{i,j}||$. As shown in Figure 8, up to three shift amounts are required for our quadrisection approximation approach. These shift amounts (S_1, S_2, and S_3) are easily obtained by the conditional statements described in Figure 14.

Finally, the normalized HOG features are calculated by shift

```
if $(2^{\alpha-1} < ||V_{i,j}|| \leq 2^{\alpha-1} + \frac{2^{\alpha-1}}{4})$ then
    $S_1 = \alpha + 1,\ S_2 = \alpha + 2,\ S_3 = \alpha + 3$
else if $(2^{\alpha-1} + \frac{2^{\alpha-1}}{4} < ||V_{i,j}|| \leq 2^{\alpha-1} + 2 \times \frac{2^{\alpha-1}}{4})$ then
    $S_1 = \alpha + 1,\ S_2 = \alpha + 2,\ S_3 = 0$
else if $(2^{\alpha-1} + 2 \times \frac{2^{\alpha-1}}{4} < ||V_{i,j}|| \leq 2^{\alpha-1} + 3 \times \frac{2^{\alpha-1}}{4})$ then
    $S_1 = \alpha + 1,\ S_2 = 0,\ S_3 = \alpha + 3$
else if $(2^{\alpha-1} + 3 \times \frac{2^{\alpha-1}}{4} < ||V_{i,j}|| \leq 2^{\alpha})$ then
    $S_1 = \alpha + 1,\ S_2 = 0,\ S_3 = 0$
end if
```

Fig. 14 Conditional statements to obtain shift amounts

```
if $((S_1 != 0) \&\& (S_2 != 0) \&\& (S_3 != 0))$ then
  $v = (f >> S_1) + (f >> S_2) + (f >> S_3)$
else if $((S_1 != 0) \&\& (S_2 != 0) \&\& (S_3 == 0))$ then
  $v = (f >> S_1) + (f >> S_2)$
else if $((S_1 != 0) \&\& (S_2 == 0) \&\& (S_3 != 0))$ then
  $v = (f >> S_1) + (f >> S_3)$
else if $((S_1 != 0) \&\& (S_2 == 0) \&\& (S_3 == 0))$ then
  $v = (f >> S_1)$
else
  $v = 0$
end if
```

Fig. 15 Conditional shift statements for the HOG normalization

operations as show in the Figure 15. Therefore, the normalization of 9 cell histograms in the window is accomplished in 4 clock cycles.

E. Binarization of features

As described in Section III-D, the HOG binarization process is relatively easy, and thus we implemented in a combinational circuit.

F. Data stream of HOG feature extraction

As summarized in Figure 16, 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 Block RAM. The Block RAM can hold all the normalized HOG histograms of 62×46 blocks, which are obtained from a single frame image.

G. Human detection using AdaBoost classifiers

The HOG features of a frame stored in the Block RAM are examined for human detection by AdaBoost classifiers which are generated by an offline learning in advance. The detection process is repeatedly executed moving a detection window on a given frame image. This means the detection process does not need every HOG features in the frame at the same time. Thus, the process can start the execution once the HOG features in the detection window are stored in the Block RAM. Since dual-port Block RAMs are provided in Xilinx FPGAs, the two processes for the HOG feature extraction and the detection can be partially overlapped.

In the followings, the AdaBoost method, generation of classifiers and the detection scheme are explained.

1) AdaBoost: AdaBoost is a machine learning method that combines multiple weak classifieds, each of which only returns a true or false, so that an effective strong classifier is constructed [2]. In the training phase, positive sample images and negative sample images are repeatedly used by changing their weights, to select weak classifiers.

2) Classifier generation: Since generation of classifiers using sample images is an offline process, we implemented with software. In this implementation, HOG features that frequently appear in human sample images (positive samples),

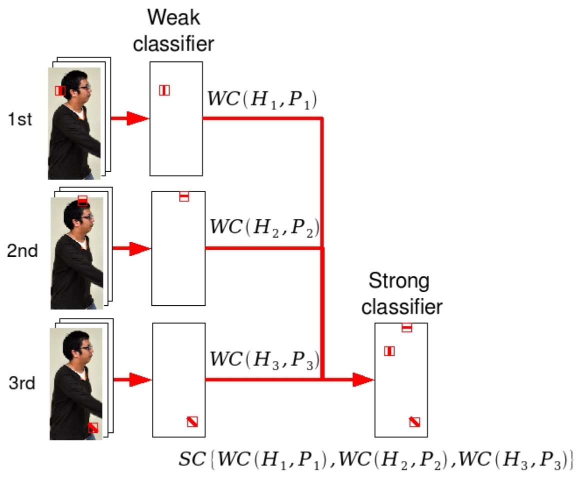

Fig. 17 Strong classifier generated by three times training

while are 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. Let H_N and P_N denote the HOG feature amount and its block coordinate obtained in the N-th training phase, respectively. Then, the strong classifier constructed through X times of the training can be expressed as: $SC\{WC(H_1, P_1), WC(H_2, P_2), WC(H_3, P_3), \cdots, WC(H_X, P_X)\}$.

An example of a strong classifier constructed with three training phases is shown in Figure 17.

The HOG features and their block coordinates selected by the offline AdaBoost training are stored in two ROMs called ROM_HOG and ROM_POSI, respectively, while these ROMs are actually implemented with Block RAMs. What the circuit for strong classifier needs to do is simply to compare the contents of the ROMs with HOG features extracted from input images.

In this implementation, we set the size of the detection window to 50×105 luminance values (8×19 blocks), which is the same as that of training samples. Both the HOG features and block coordinates are coded in 8-bit data. 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.

3) Human detection with classifiers: Using the strong classifier constructed in the way shown in Section IV-G2, image-based human detection is carried out.

The detection flow is summarized in Figure 18. In contrast to the streamed approach exploited in the HOG feature extraction, a random access approach is needed since every weak classifier corresponds to a different block coordinate.

Figure 19 shows an example of the human detection process.

The detection window moves the entire image from the upper left corner in a cell-by-cell raster scan manner. Since the detection window of 8×19 blocks is used, the required

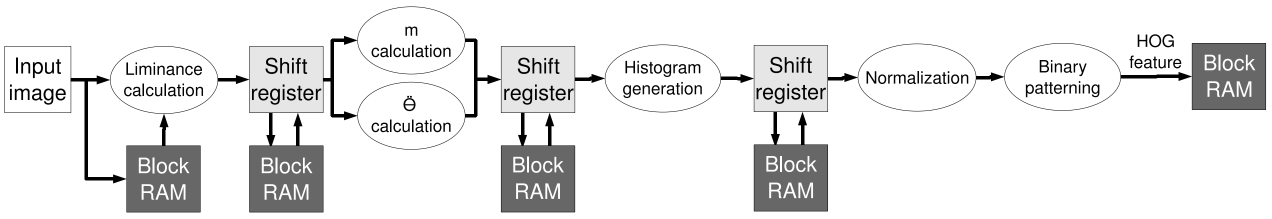

Fig. 16 Pipelined structure for HOG feature extraction

1) The block coordinate for the first weak classifier is obtained by accessing the head of ROM_POSI.
2) The block coordinate is translated to an memory address of the HOG Block RAM and the corresponding HOG feature is read out. The access to the HOG Block RAM is a kind of random access, but the accessed address never go out of the current detection window.
3) The feature read out from the HOG Block RAM is compared with that of ROM_HOG. If they are matched, the weak classifier returns a true. This process is executed in one clock cycle.
4) The address for ROM_HOG and ROM_POSI is incremented and the steps of (2), (3), and (4) are repeated.
5) When all the data stored in ROM_HOG and ROM_POSI are examined, the detection window is moved by one cell and the steps of (1), (2), (3), and (4) are repeated again.
6) When the detection window finishes scanning the entire image, the window location, where the largest number of weak classifiers returned a true and the number exceeds a threshold value, is detected as a human image region.

Fig. 18 Process flow of human detection

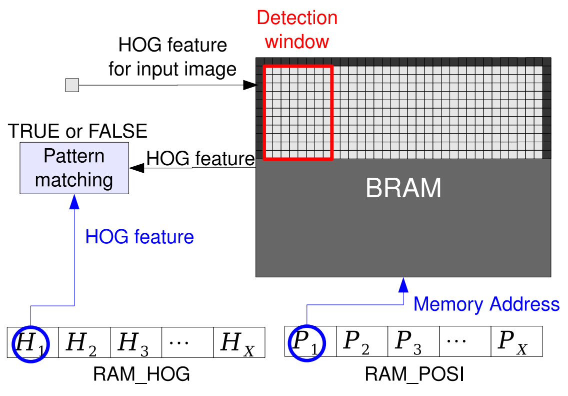

Fig. 19 An example of human detection process

number of widow scans for a single image is $(62-8+1)\times(46-19+1) = 1,540$. Therefore, the total number of matching processes required for the 84 weak classifiers requires is $1540\times 84 = 129,360$.

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 from one frame data, while the AdaBoost detection process takes 129,360. If we just executed these two processes sequentially, it would require 514,812 clock cycles and would not fit in the frame time. However, the dual-port HOG Block RAM allows us to execute these processes in a partially overlapped manner as described above. In our design, the whole process for one frame finishes in 383,840 clock cycles including the HOG feature extraction and AdaBoost detection, enabling in-frame real-time processing.

TABLE I The results of FPGA mapping

	Used	Available	%
Number of Slice Registers	2,181	28,800	7.6
Number of Slice LUTs	17,383	28,800	60.4
Number of fully used LUT-FF pairs	2,070	23,109	9.0
Number of Block RAM/FIFO	36	48	75.0
MAX Frequency(MHz)	44.85		

V. Implementation results and evaluation

The human detection processing described in Section IV was implemented on a Xilinx ML501 board equipped with a Virtex-5 VLX-50 FPGA with Verilog-HDL. The circuit was synthesized and mapped on the FPGA using Xilinx ISE 12.1 tool-set.

The implementation results of the circuit are summarized in Table I. The resource usage of this rather small FPGA supports the effectiveness of our compact implementation approach.

While the maximum operating frequency of the circuits achieved 44.85 MHz, the camera device used in our experimentation system restrict the system clock to 25MHz. In spite of the restricted frequency, our system achieved the throughput of 62.5 fps for VGA frames and execution latency was also fitted in a single frame time, that is, the real-time performance was accomplished. Furthermore, if a high-speed camera device were used and the FPGA circuits operated with the maximum operation frequency of 44.85 MHz, the execution throughput would be improved up to 112 fps. Figure 20 illustrates some of our experimentation results of the human detection system.

Next, in order to evaluate the impact on the quality of results of simplification of implementation described in Section IV such as the use of fixed-point arithmetic, we compared the simplified implementation (HOG_HARD) to the original one (HOG_SOFT). We implemented two software simulators in

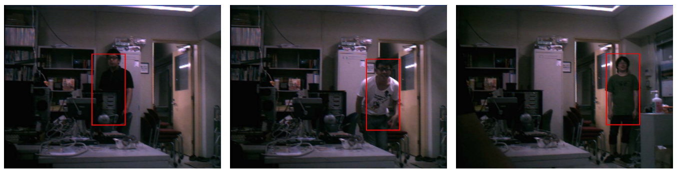

Fig. 20 Examples of human detection process

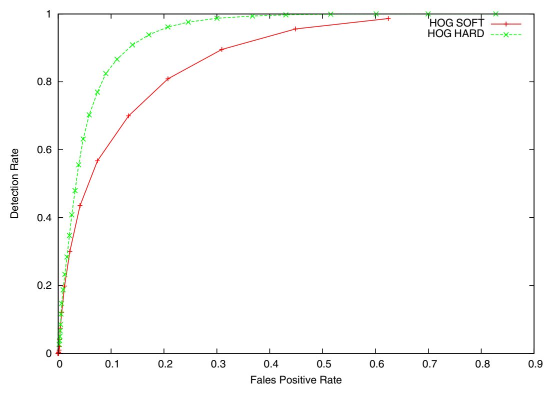

Fig. 21 ROC curves for the NICTA Pedestrian database

C language for both of the implementation and evaluated accuracy of human detection.

In this evaluation, NICTA Pedestrian database [9] was used for benchmarking. From the database, 2,000 images were used for offline machine learning, while 1,000 images were used as the evaluation data. The size of each image is 64×80 pixels. AdaBoost classifiers were generated after 500 times of training. The threshold value for the HOG binarization was set to 0.08.

The comparison results are summarized as a receiver operator characteristics (ROC) curve and shown in Figure 21. This 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 an ROC curve goes, the better quality of results the system shows. The plots were made by changing the threshold number of weak classifiers for detection from 0 to 30.

As a result, the algorithm simplified for hardware implementation (HOG_HARD) shows 96.6% of the detection rate with 20.7% of the false positive rate, while the original one (HOG_SOFT) shows 80.9% of the detection rate with 20.8% of the false positive rate. As far as this evaluation results are concerned, the simplified implementation showed rather better detection results. Although the detection results for HOG_SOFT might be improved by tuning parameters such as the threshold value, the evaluation results suggest the negative impact of algorithm simplification for hardware implementation is limited.

Finally, we compared the throughput of the FPGA impelmetation to software implementation compiled by gcc version 4.3.1. We used a PC with 2.67-GHz Intel Core i7 920 and 6-GB DDR3 operated by openSUSE 11.2. As a result, the software system achieved about 15 fps. This means the FPGA implementation is 4.2 times faster than the software implementation. Moreover, 7.5 times faster throughput is expected if the camera device operates at the maximum frequency.

VI. Conclusion

In this paper, compact FPGA implementation of real-time human detection using the HOG feature extraction and AdaBoost has been presented. As a result of empirical evaluation with an experimental setup equipped with an FPGA and camera device, 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 simplification was made to alleviate hardware complexity, the evaluation with ROC curves showed that the simplification did not harm the quality of detection severely.

Our future work includes to introduce more advanced techniques such as Joint-HOG [10] which takes account of association between features and EHOG [11] in which feature amounts are further reduced. Detailed analysis of the detection quality and implementation alternatives is also important.

References

[1] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *Proceedings of the 2005 International Conference on Computer Vision and Pattern Recognition*, vol. 2. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893.

[2] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” in *Proceedings of the 2nd European Conference on Computational Learning Theory*. London, UK: Springer-Verlag, 1995, pp. 23–37.

[3] W. Sun and K. Kise, “Speeding up the Detection of Line Drawings Using a Hash Table,” in *Proceedings of The 1st China Japan Korea Joint Workshop on Pattern Recognition*, vol. 2, nov 2009, pp. 896–900.

[4] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, “Hardware Architecture for HOG Feature Extraction,” in *Proceedings of the 2009 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1330–1333.

[5] T. P. Cao and G. Deng, “Real-Time Vision-Based Stop Sign Detection System on FPGA,” in *Proceeding of Digital Image Computing: Techniques and Applications*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 465–471.

[6] N. Dalal, B. Triggs, and C. Schmid, “Human Detection Using Oriented Histograms of Flow and Appearance,” in *Proceedings of European Conference on Computer Vision*. Graz, Autriche: Springer-Verlag, 2006, pp. 428–441.

[7] P. Dollar, C. Wojek, B. Schiele, and P. Perona, “Pedestrian detection: A benchmark,” in *Proceedings of the 2009 International Conference on Computer Vision and Pattern Recognition*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 304–311.

[8] C.-H. Kuo and R. Nevatia, “Robust multi-view car detection using unsupervised sub-categorization,” in *Proceedings of Workshop on Applications of Computer Vision*, 2009, pp. 1–8.

[9] NICTA:Computer Vision Datasets. [Online]. Available: http://www.nicta.com.au/research/projects/AutoMap/computer_vision_datasets

[10] T. Mitsui and H. Fujiyoshi, “Object detection by joint features based on two-stage boosting,” in *Proceedings of the 2009 International Conference on Computer Vision Workshops*. IEEE Computer Society, 2009, pp. 1169–1176.

[11] C. Hou, H. Ai, and S. Lao, “Multiview pedestrian detection based on vector boosting,” in *Proceedings of the 8th Asian conference on Computer vision*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 210–219.