A low power, VLSI object recognition processor using Sparse FIND feature for 60 fps HDTV resolution video

Go Matsukawa[1a)], **Taisuke Kodama**[1], **Yuri Nishizumi**[1], **Koichi Kajihara**[1], **Chikako Nakanishi**[2], **Shintaro Izumi**[1], **Hiroshi Kawaguchi**[1], **Toshio Goto**[3], **Takeo Kato**[4], **and Masahiko Yoshimoto**[1]

[1] *Graduate School of System Informatics, Kobe University, 4 Rokkodai, Nada-ku, Kobe, Hyogo 657–8501, Japan*

[2] *Osaka Institute of Technology, 10–1 Wakamiya, Morinosato, Atsugi, Kanagawa 243–0197, Japan*

[3] *Electronics Advanced Development Department, Toyota Motor Corporation, Toyota 471–8571, Japan*

[4] *Toyota Central R&D Labs., Inc, Nagakute 480–1192, Japan*

a) *matsukawa@cs28.cs.kobe-u.ac.jp*

Abstract: This paper describes a low-power object recognition processor VLSI for HDTV resolution video at 60 frames per second (fps) using an object recognition algorithm with Sparse FIND features. The VLSI processor features two-stage feature extraction processing by HOG and Sparse FIND, a highly parallel classification in the support vector machine (SVM), and a block-parallel processing for RAM access cycle reduction. Compared to the accuracy by the original Sparse FIND algorithm, the two-stage object detection demonstrates insignificant accuracy degradation. Using this architectural design, a 60 fps performance for object recognition of HDTV resolution video was attained at an operating frequency of 130 MHz. This $3.35 \times 3.35\,\mathrm{mm}^2$ chip, designed with 40 nm CMOS technology, contains 8.22 M gates and 5 Mb SRAM in the chip of $3.35 \times 3.35\,\mathrm{mm}^2$. The simulated power consumption at 133 MHz were 528 mW and 702 mW at the slow process condition (SS, 0.81 V, −40°C) and typical process condition (TT, 0.9 V, 25°C), respectively.

Keywords: Sparse FIND, object recognition, HDTV, low-power, VLSI

Classification: Integrated circuits

References

[1] N. Dalal and B. Triggs: "Histograms of oriented gradients for human detection," IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2005) 886 (DOI: 10.1109/CVPR.2005.177).

[2] K. Takagi, *et al.*: "A sub-100-milliwatt dual-core HOG accelerator VLSI for

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

real-time multiple object detection," IEEE Conf. on Acoustics, Speech and Signal Processing (ICASSP) (2013) 2533 (DOI: 10.1109/ICASSP.2013.6638112).

[3] T. Watanabe, *et al.*: "Co-occurrence histograms of oriented gradients for pedestrian detection," Third Pacific-Rim Symposium on Image and Video Technology (2009) 34 (DOI: 10.1007/978-3-540-92957-4_4).

[4] A. Suleiman, *et al.*: "A 58.6 mW real-time programmable object detector with multi-scale multi-object support using deformable parts model on 1920 × 1080 video at 30 fps," IEEE Symp. on VLSI-Circuits (2016) 184 (DOI: 10.1109/VLSIC.2016.7573528).

[5] K. He, *et al.*: "Deep residual learning for image recognition," IEEE Conf. on Computer Vision and Pattern Recognition, CVPR (2016) 770 (DOI: 10.1109/CVPR.2016.90).

[6] T. Kato, *et al.*: "SpaFIND: An effective and low-cost feature descriptor for pedestrian protection systems in economy cars", to be published on IEEE Trans. on Intelligent Vehicles (2017).

[7] W. Liu, *et al.*: "SSD: Single shot MultiBox detector," Third European Conf. on Computer Vision (ECCV) (2016).

[8] T. Kato, *et al.*: "Sparse find: A novel low computational cost feature for object detection," FASTZero (2013).

[9] H. Cao, *et al.*: "Feature interaction descriptor for pedestrian detection," IEICE Trans. Inf. & Syst. **E93-D** (2010) 2656 (DOI: 10.1587/transinf.E93.D.2656).

[10] Caltech Pedestrian Detection Benchmark: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/.

[11] P. Dollar, *et al.*: "Pedestrian detection: A benchmark," IEEE Conf. on Computer Vision and Pattern Recognition, CVPR (2009) (DOI: 10.1109/CVPR.2009.5206631).

1 Introduction

Real-time object recognition is an important task for many computer vision applications such as surveillance cameras, automobile systems, and robots. The histogram of oriented gradients (HOG) [1, 2], which is widely known as a feature descriptor, is robust to changes in luminance and scale. Moreover feature descriptors obtained by computing the correlation between histogram elements have been proposed. They have achieved accurate object recognition. Among them, the Sparse FIND feature has higher capability of object recognition than HOG and CoHOG [3]. Advances in high-performance general-purpose processors in recent years make it possible to recognize real-time objects using these features. However, it is difficult to apply these processors to applications for mobile systems that require low power consumption because of battery restrictions and in-vehicle systems that have constraints on severe thermal design. For that reason, low-power and high-performance object recognition processor VLSI is desired for use in widely various applications. Particularly, in-vehicle systems require detection of distant objects while traveling at high speed. Under such circumstances, it is necessary to recognize objects with a high frame rate at high resolution, such as that of HDTV (1920 × 1080) [2, 4].

High-resolution video offers the capability of capturing detailed shapes of objects with a wide field angle, but with the important difficulty that large amounts

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

of computation are necessary for real-time processing. Currently, object recognition using a neural network yields excellent results in terms of recognition accuracy [5, 7]. Nevertheless, the computational cost is very high, making it difficult to produce a high-resolution and high-speed processing system with low power consumption. For example, the multiscale object recognition algorithm SSD using deep learning provides excellent accuracy for object recognition, but even when processing an input image of 300×300 pixels at 46 fps, GPGPU dissipates several hundred watts because the throughput exceeds 1 TOPs [7]. Therefore, a VLSI accelerator that can achieve real-time object recognition for HDTV resolution video is strongly requested.

As described in this paper, we present the Sparse FIND algorithm introduced in VLSI design in Chapter 2 and the proposed design technique in Chapter 3. Chapter 4 presents a description of the entire architecture and explains its performance evaluation. These are followed by an explanation of VLSI implementation in Chapter 5 and presentation of the conclusion in Chapter 6.

2 Sparse FIND algorithm and design issue for VLSI implementation

Sparse FIND [8] is a feature descriptor which reduces the number of dimensions of the FIND [9] feature created using the HOG feature. The FIND feature is obtained by computing the correlation among all histogram elements of the HOG feature and performing normalization of the correlation. The object shape can be expressed more finely. Therefore, the accuracy of detection achieved with the FIND feature is superior to that achieved with HOG feature. However, high computational costs are incurred. To overcome this issue, among the elements of the HOG feature, only elements with high validity in identification are used for feature extraction (sparsified) and correlation is taken to reduce the number of dimensions. Consequently, compared with the FIND feature, computational costs are reduced while maintaining accuracy of detection.

The Sparse FIND feature extraction method is described sequentially as follows. First, the gradient magnitude and direction are calculated using the luminance value of each pixel of the input image. Using these, the gradient histogram in the d direction in a cell of $p \times p$ pixels is created. A group of $q \times q$ cells is defined as one block. The histograms for a block are arranged in a row to create HOG feature vectors:

$$H = (h_1, h_2, \cdots, h_m) \tag{1}$$

as depicted in Fig. 1. The number of dimensions per block is $m(= d \times q \times q)$.

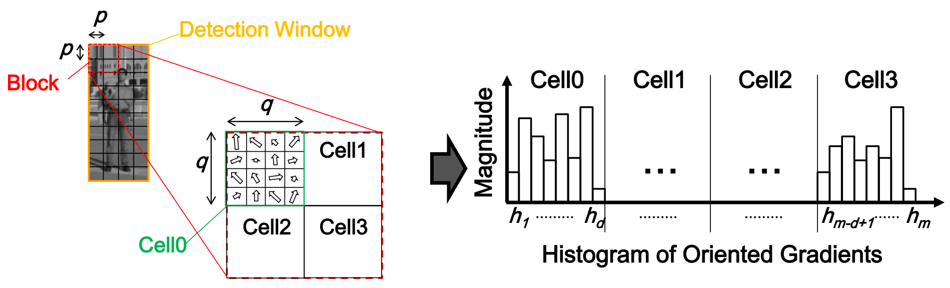

Fig. 1. HOG feature extraction

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

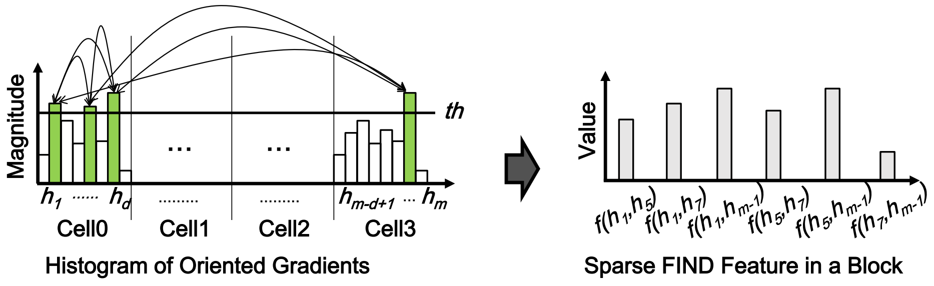
Magnitude
th
h_1 …… h_d
……… ………
h_{m-d+1} … h_m
Cell0 Cell1 Cell2 Cell3
Histogram of Oriented Gradients
Value
$f(h_1,h_5)$ $f(h_1,h_7)$ $f(h_1,h_{m-1})$ $f(h_5,h_7)$ $f(h_5,h_{m-1})$ $f(h_7,h_{m-1})$
Sparse FIND Feature in a Block

Fig. 2. Sparse FIND Feature extraction

Next, the sparsification threshold *th* and the dimensionless coefficient *a* are calculated from the block-level histogram. Using the sparsification threshold *th*, elements with high validity for identification are selected from the HOG features (Fig. 2). The sparsification threshold *th* and the dimensionless coefficient *a* are calculated using the following equations.

$$th = k \cdot \frac{1}{m} \sum_{h_i \in H} h_i \tag{2}$$

$$a = 1 \Big/ \sum_{h_i \in H} |h_i|^2 \tag{3}$$

Here, the coefficient *k* in equation (2) is a sparsification rate that determines the number of dimensions of Sparse FIND feature. The dimensionless coefficient in equation (3) makes normalization processing unnecessary. The Sparse FIND feature is calculated (4) by taking correlation with only elements which exceeds *th* as illustrated in Fig. 2.

$$f(h_i, h_j) = a \cdot h_i \cdot h_j | h_i \in H_D \wedge h_j \in H_D \wedge i \neq j \tag{4}$$

H_D is defined as $H_D = \{h_i | h_i > th\}$.

In the VLSI implementation, the processor was designed with $p = 4$, $d = 8$, $q = 2$ and $k = 1.0$.

Fig. 3 presents data demonstrating the extraction accuracy of HOG, FIND and Sparse FIND with sparsification rate k from 0.5 to 2.0 in 0.5 increments. The Sparse FIND exhibits better characteristics than HOG.

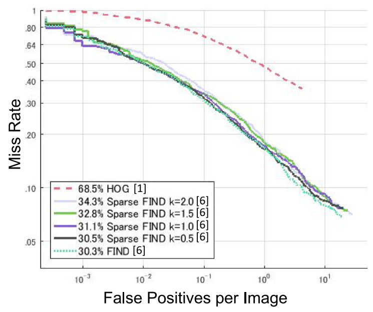

Fig. 3. Recognition accuracy cited from reference [1, 6]. FIND [6] is a special case of Sparse FIND with k = 0.0, which computes the full feature interaction for each pair of all possible combinations of *H*.

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

Table I. The number of features per a detection window

	HOG	Sparse FIND	FIND
# of features	2400	3300	76800

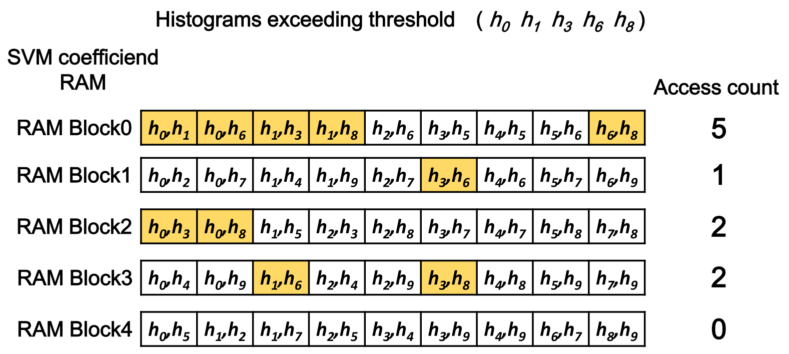
Histograms exceeding threshold (h_0 h_1 h_3 h_6 h_8)
SVM coefficiend RAM
Access count
RAM Block0 | h_0h_1 | h_0h_6 | h_1h_3 | h_1h_8 | h_2h_6 | h_3h_5 | h_4h_5 | h_5h_6 | h_6h_8 | 5
RAM Block1 | h_0h_2 | h_0h_7 | h_1h_4 | h_1h_9 | h_2h_7 | h_3h_6 | h_4h_6 | h_5h_7 | h_6h_9 | 1
RAM Block2 | h_0h_3 | h_0h_8 | h_1h_5 | h_2h_3 | h_2h_8 | h_3h_7 | h_4h_7 | h_5h_8 | h_7h_8 | 2
RAM Block3 | h_0h_4 | h_0h_9 | h_1h_6 | h_2h_4 | h_2h_9 | h_3h_8 | h_4h_8 | h_5h_9 | h_7h_9 | 2
RAM Block4 | h_0h_5 | h_1h_2 | h_1h_7 | h_2h_5 | h_3h_4 | h_3h_9 | h_4h_9 | h_6h_7 | h_8h_9 | 0

Fig. 4. Example of RAM access when classification using Sparse FIND feature

The number of features of Sparse FIND per detection window is about 1.4 times as numerous as the number of features of HOG (Table I). The number of features is a factor determining the RAM size for the SVM coefficient. It is also related to the magnitude of the workload. It is fewer than that of FIND, but the shortcomings remain: a workload of 322 GOPS is required. The number of processing cycles is also large. Solutions to these issues are proposed in Section 3.

Although the differences between the numbers of Sparse FIND features and the HOG features are not large, a difficulty exists in hardware design to attain high speed processing. Fig. 4 presents an example of RAM block access randomness when acquiring SVM coefficients. For simplicity of explanation, let the number of histograms in the block and the number of RAM blocks, respectively be 10 and 5 blocks. Each box in the RAM block stores SVM coefficients corresponding to the Sparse FIND feature calculated from the combination of the histograms written in the boxes. In this example, because $(h_0, h_1, h_3, h_6, h_8)$ are histograms exceeding the threshold, yellow boxes contain the SVM coefficient corresponding to the Sparse FIND features calculated using the histogram pair among the above. Consequently, it is necessary to access all yellow boxes to read all SVM coefficients required for block processing. For that reason, the access count in each RAM block fluctuates dynamically. In the example portrayed in Fig. 4, five cycles are necessary for processing this block. This factor impedes the speeding up of object recognition using the Sparse FIND feature. Increasing the number of RAM blocks can reduces dispersion but it entails area overhead.

3 Proposed technique

To resolve the difficulty of hardware implementation, three design techniques are newly introduced in the proposed architecture.

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

3.1 Block-based two-stage object recognition algorithm

The first technique is two-stage processing by HOG and Sparse FIND. Using classification results by the HOG features, the detection windows that are highly likely to be a target object (e.g. Pedestrian) are narrowed down; the windows unlikely to be a target object are rejected. When the SVM score in the HOG classification for a detection window exceeds a threshold α, the window is rejected. The threshold α is defined as the rejection threshold in this paper. Then, Sparse FIND features are extracted only in the block in squeezed detection windows; the second classification is performed. The HOG features are calculable in the process of extracting Sparse FIND features. In the proposed architecture, HOG and Sparse FIND stage are performed as block-based which means they are performed for each block in order to adapt to the highly parallel processing of the SVM calculation described in Section 3.2. Fig. 5 presents an example of two-stage processing block-based. In this example, it is assumed that there is no block overlap and the detection window comprises of 4 vertical blocks and 3 horizontal blocks. At the HOG stage, blocks are processed in the vertical direction in order from the left column. When classification for detection windows at the HOG stage is completed and some windows are not rejected, processing at Sparse FIND stage is performed sequentially in the vertical direction for blocks belonging to the windows. Two-stage pipeline processing reduces the total computation amount by 19.7% without degrading the accuracy in comparison to detection by Sparse FIND alone.

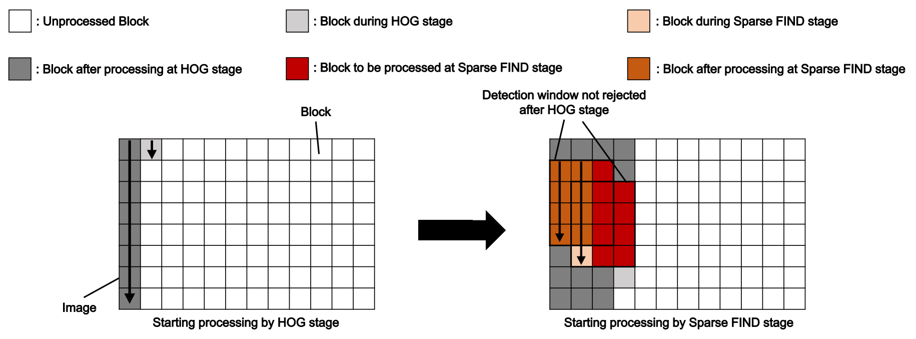
: Unprocessed Block
: Block during HOG stage
: Block during Sparse FIND stage
: Block after processing at HOG stage
: Block to be processed at Sparse FIND stage
: Block after processing at Sparse FIND stage
Block
Detection window not rejected after HOG stage
Image
Starting processing by HOG stage
Starting processing by Sparse FIND stage

Fig. 5. Flow of block-based two-stage processing for VLSI implementation

3.2 High parallelization of computing for SVM

The second technique is highly parallel processing of SVM operation. The SVM operation is performed on a block-level. Since the detection window consists of 75 blocks. One block belongs to a maximum of 75 detection windows. The SVM classification circuit presented in Fig. 6 comprises of five SVM calculation cores, a detector, an SRAM for intermediate classification results, and an SRAM for SVM coefficients. The SVM calculation core comprises of 15 MAC modules, and carries out multiply-accumulate operations for 15 blocks: the MAC module in the SVM classification circuit has 75 parallel configurations (5 cores × 15). This number is equal to the total number of blocks per detection window. Each time the SVM calculation of the block is completed, the SVM score obtained from the block is

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

cumulatively added to the SVM score sent from the MAC module at the preceding row and transferred to the MAC module at the next row. At that time, the SVM score output from the MAC module at the last row in the "SVM calculation core" is written to "SVM calculation Intermediate result RAM", and when calculating the block of the next column, the SVM score is read out to the MAC module at the first row of the next column. Finally, when the SVM operation of the block 74 of a certain detection window is performed, the processing of the detection window is ended. The intermediate result of the SVM operation of the window is stored in the RAM and is invoked from the RAM at any time when the data of the block belonging to the window is inputted if the calculation of all the feature in the window is not completed and data of a new block belonging to the window is not input. In this way, high-speed operation is achieved by executing the SVM operation at maximum 75 parallels. However, in the classification in Sparse FIND stage, each MAC module is controlled by enabling the signal to reduce power consumption because all detection windows to which a block belongs are not always classified as described in section 3.2. This method reduced the processing cycle of the SVM operation by 98.6%.

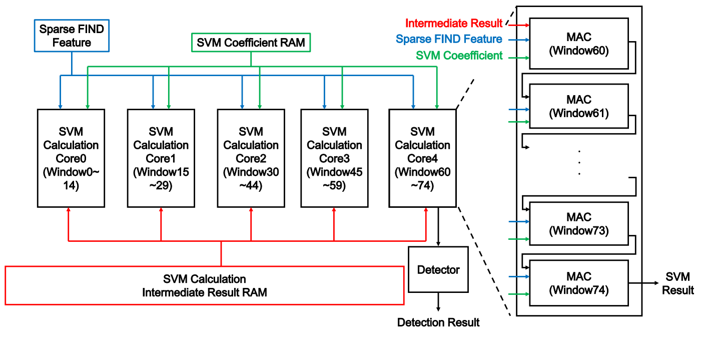
Sparse FIND Feature
SVM Coefficient RAM
Intermediate Result
Sparse FIND Feature
SVM Coeefficient
MAC (Window60)
MAC (Window61)
MAC (Window73)
MAC (Window74)
SVM Result
SVM Calculation Core0 (Window0~14)
SVM Calculation Core1 (Window15~29)
SVM Calculation Core2 (Window30~44)
SVM Calculation Core3 (Window45~59)
SVM Calculation Core4 (Window60~74)
SVM Calculation Intermediate Result RAM
Detector
Detection Result

Fig. 6. Block diagram of SVM classification circuits

3.3 Block-parallel processing for RAM access cycle reduction

The third technique is a block-parallel processing for RAM access cycle reduction at the Sparse FIND stage. In the SVM operation, to calculate the sum of products with the Sparse FIND feature, the SVM coefficient is accessed and read out from the RAM. The tendency of features excluded from computation by sparsifying process is random for each block and has no regularity. In the feature calculation and the SVM operation for each block, the randomness causes dispersion in the number of accesses for each RAM block, so that efficient computation can be not performed. Fig. 7 presents the proposed block-parallel processing scheme. First, addresses for the SVM coefficient RAM and selection signals are generated by the sparsifying processing using the histogram of the block A and B. The selection signal determines which histogram is used to generate the Sparse FIND feature. After Sparse FIND features are generated, SVM operation is performed. In this way, the proposed block-parallel processing requires no duplication of feature extraction and classification circuits. An example is shown in Fig. 8. When two blocks A and B are processed consecutively, the access cycle is the sum of the

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

maximum number of RAM accesses of each block. If the maximum access number of the block A is 6 and the maximum number of accesses of the block B is 5, then 11 cycles in all are required. When two blocks A and B are executed at the same time, the access cycle is the maximum sum of the number of RAM accesses: 9 cycles are required in the case shown in right side of Fig. 8. Consequently, the maximum access cycle for two blocks is changed from the sum of the maximum access cycle of two blocks to the maximum sum of access cycle of 2 block processing. Using this method, the average access cycle of SVM coefficient to RAM in the SVM operation was reduced by 28.5%.

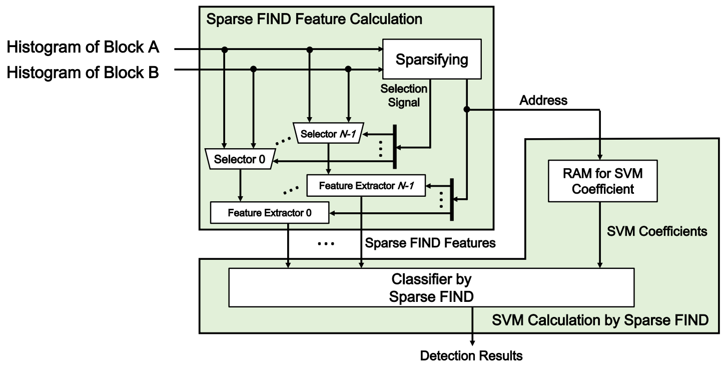

Fig. 7. Block-parallel processing scheme: block A and block B are processed in parallel. Here, *N* is the number of RAM blocks.

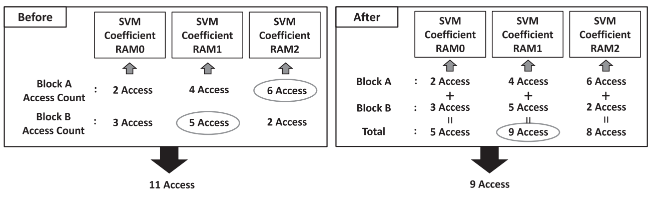
Before
SVM Coefficient RAM0
SVM Coefficient RAM1
SVM Coefficient RAM2
Block A Access Count : 2 Access 4 Access 6 Access
Block B Access Count : 3 Access 5 Access 2 Access
11 Access
After
SVM Coefficient RAM0
SVM Coefficient RAM1
SVM Coefficient RAM2
Block A : 2 Access 4 Access 6 Access
Block B : 3 Access 5 Access 2 Access
Total : 5 Access 9 Access 8 Access
9 Access

Fig. 8. Comparison of RAM access-cycle count

4 Architecture

4.1 Architectural design

Fig. 9 portrays a block diagram of the entire object recognition processor core, which contains function blocks for common stage, HOG stage and Sparse FIND stage. Magnitude and orientation of luminance gradient are calculated in the

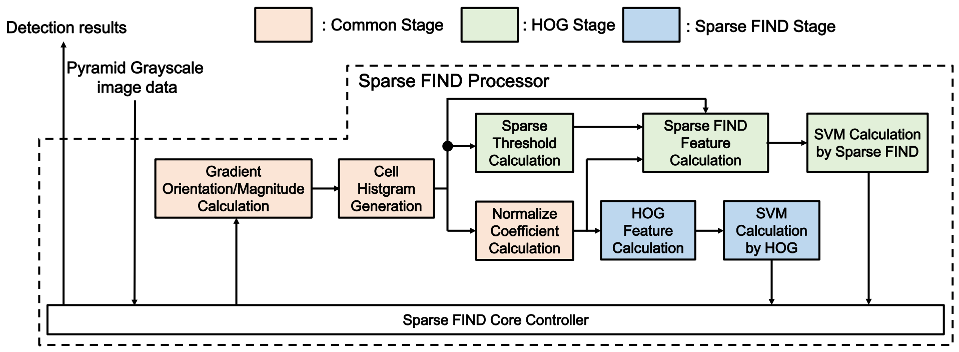
Detection results
: Common Stage
: HOG Stage
: Sparse FIND Stage
Pyramid Grayscale image data
Sparse FIND Processor
Sparse Threshold Calculation
Sparse FIND Feature Calculation
SVM Calculation by Sparse FIND
Gradient Orientation/Magnitude Calculation
Cell Histgram Generation
Normalize Coefficient Calculation
HOG Feature Calculation
SVM Calculation by HOG
Sparse FIND Core Controller

Fig. 9. Architectural block diagram of Sparse FIND processor core

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

common pre-processing stage, which are followed by the computation of a cell histogram and a dimensionless coefficient. In the HOG stage, HOG features are calculated and classified. Generation of Sparse FIND features and SVM classification are performed for blocks in the window exceeding the rejection threshold α among the columns where the HOG classification is completed. The HOG classification and Sparse FIND classification can be performed concurrently. Also, the cell histogram generation, the cell histogram addition, the dimensionless coefficient calculation, and the HOG classification are executed in a pipeline manner. The Sparse FIND core controller unit identifies blocks for which the Sparse FIND feature extraction process is performed. When the Sparse FIND classification process is delayed, the process for function block for common stage and HOG stage in Fig. 9 are stopped, where the stop signal is sent from the Sparse FIND core controller.

4.2 Performance evaluation

The FPGA emulation for pedestrian detection was performed using test images (1920×1080 pixels) to evaluate the accuracy degradation and processing speed. These test images are obtained by resizing the image of 4024 sheets of Caltech Pedestrian Dataset [10, 11] from VGA resolution to HDTV resolution and performing grayscale conversion. The pyramid grayscale image data, which is transferred into the core, is created by shrinking HDTV image sequentially by $2^{-1/6}$ ($\approx$0.891) times. The pyramid images are sequentially processed while the detection window of 24×64 pixels is shifted every 4 pixels, and the pedestrian was detected using the classifier with SVM coefficients learned beforehand.

In the VLSI design, we adopted the CORDIC method to calculate arctangent and square root at histogram generation, and the NEWTON method for square root division in calculating the dimensionless coefficient. The accuracy was improved by the appropriate number of steps of CORDIC and NEWTON method. The smallest number of steps of CORDIC and NEWTON methods maintaining the accuracy was chosen. As a result, the number of CORDIC steps was 11, and the number of NEWTON was 4.

The rejection rate, which is the ratio of rejected windows to all windows, is strongly related to processing speed. As the number of blocks rejected after the HOG classification increases, the number of blocks to be processed by Sparse FIND can be reduced, allowing speed performance enhancement. On the contrary,

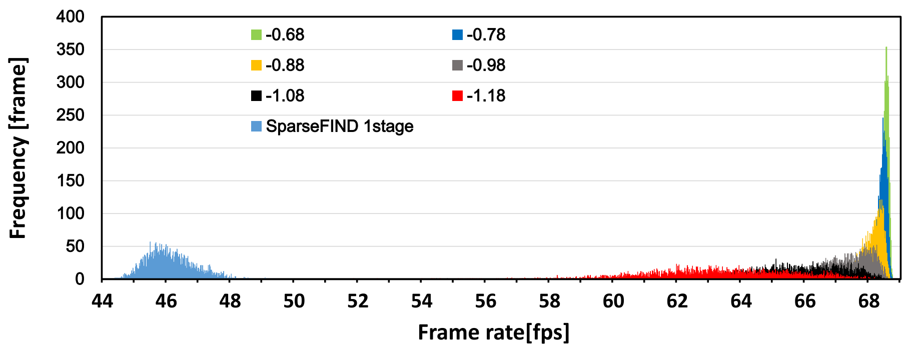

Fig. 10. Processing speed distribution versus rejection threshold

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

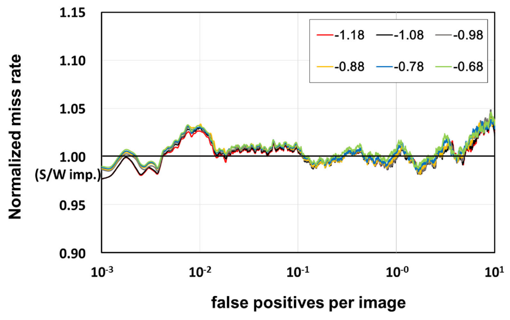

Fig. 11. Detection accuracy versus rejection threshold

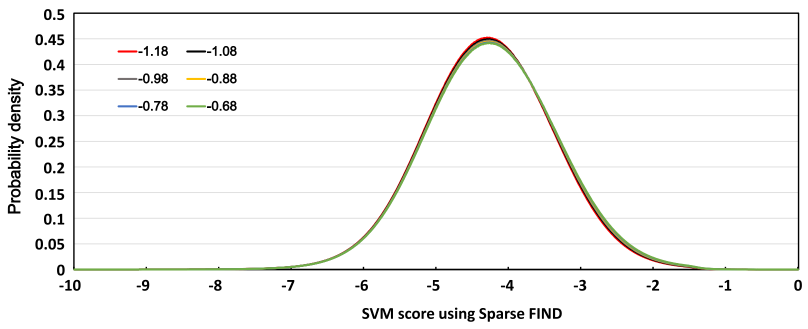

Fig. 12. Probability density function of SVM score for the window rejected versus SVM score using Sparse FIND

if rejected too much, the accuracy can be degraded. The rejection rate depends on the rejection threshold α. Table II shows the accuracy (average miss rate), the average processing speed, and the worst processing speed when the threshold is incremented by 0.1 from the -1.18 to -0.68. The initial value of -1.18 is a value determined when learning is executed by the original C++ source code. It also contains the results of a one stage processing of Sparse FIND only.

Fig. 10 represents the distribution of the processing speed at 130 MHz when changing the rejection threshold α. The processing speed exceeds 60 fps even in the worst case if the rejection threshold α is -0.88 or more. As the rejection threshold α is increased, the distribution narrows with less variation. The average frame rate when α is -0.68 is increased by 48.4% as compared with the one-stage processing of Sparse FIND only. How does the rejection threshold α effect on the detection accuracy was investigated. Fig. 11 shows the normalized miss rate at each false positives per window when the rejection threshold α is changed. Here the miss rate obtained by the FPGA emulation was normalized by that in the software implementation. The degradation of miss rate is less than 0.5%, which shows that the hardware implementation using the proposed architecture provides a sufficient accuracy. It is understood that the precision varies depending on the rejection threshold α, but there is no great difference. Therefore, we investigated the SVM score obtained when the Sparse FIND classification was performed for windows rejected after the HOG classification.

Fig. 12 portrays the probability density function of SVM scores estimated for the window rejected. The horizontal axis shows the SVM score obtained in the Sparse FIND classification and the vertical axis shows the probability of the detection window having the SVM score on the horizontal axis. As the rejection

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

threshold α is higher, the probability of rejecting the window in the range of the SVM score from −4.2 to −1.0 rises slightly. However, in this study, the detection window with −0.3 or more of SVM score in the Sparse FIND classification is not rejected with −0.68 or less of the reject threshold α. That is, if the threshold in the Sparse FIND classification is set to be equal to or greater than −0.3, the detection result will not be affected.

Fig. 13 presents results of the pedestrian recognition using Caltech Pedestrian Dataset. Pedestrians can be recognized correctly from recognition results. In addition, Fig. 13(b) demonstrates possibility of recognizing a pedestrian at a distance with benefits of HDTV resolution images.

Table II. Processing speed for Rejection threshold

Rejection threshold	Normalized average miss rate	Average processing speed [fps] @130 MHz	Worst processing speed [fps] @130 MHz	Frequency required for 60 fps processing in worst case [MHz]
Only Sparse FIND	1	46.36	44.26	177
−1.18	1.0257	63.56	52.11	150
−1.08	1.0259	65.84	54.71	143
−0.98	1.0270	67.43	57.62	136
−0.88	1.0261	68.28	60.73	129
−0.78	1.0270	68.65	64.05	122
−0.68	1.0286	68.79	66.65	118

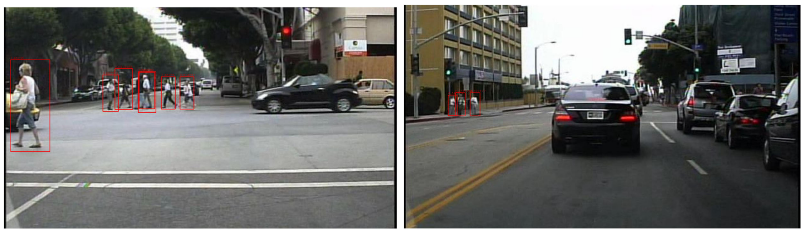
(a) (b)

Fig. 13. Results of pedestrian recognition

5 VLSI implementation

The VLSI object recognition processor using the Sparse FIND feature has been designed using 40 nm CMOS technology. It contains 8.22 M gates and 5.00 Mbit on-chip SRAM in a size of 3.35 × 3.35 mm 2. A chip plot is depicted in Fig. 14. The static timing analysis after back annotation shows that 133 MHz operation is attained at the condition of 0.81 V, SS corner and 125°C. Table III shows the specifications of the chip. Commercial evaluation tools (Primetime by Synopsys Inc.) were utilized to estimate the power consumption of the entire core with the physical parameters extracted after placement and routing. Fig. 15 presents power consumption obtained at two kinds of operating conditions. It is seen that the 133 MHz VLSI processor dissipates 528 mW and 702 mW, respectively at the slow

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017

Table III. Chip specification

Technology	40 nm CMOS
Chip size	$3.35 \times 3.35\,\text{mm}^2$
Core size	$2.95 \times 2.95\,\text{mm}^2$
Power supply	0.81 V (minimum)
Max frequency	133 MHz
Gate count	8.22 M gates
Memory size	$0.80\,\text{mm}^2$ (5.00 Mbit)
Image resolution	HDTV (1920×1080 pixels) @60 fps
Power	528 mW @ 0.81 V 133 MHz

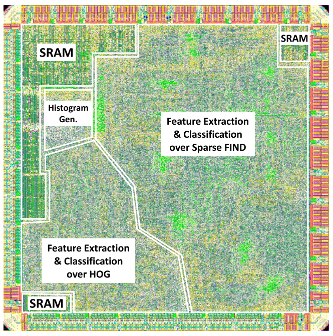
SRAM
SRAM
Histogram Gen.
Feature Extraction & Classification over Sparse FIND
Feature Extraction & Classification over HOG
SRAM

Fig. 14. Chip layout

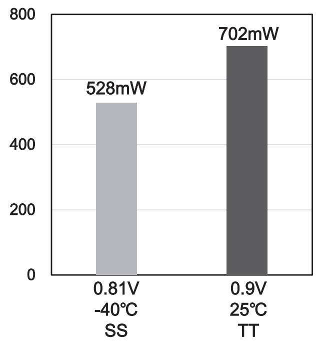

Fig. 15. Power consumption for power supply voltage

process condition (SS, 0.81 V, −40°C) and the typical process condition (TT, 0.9 V, 25°C) to realize real-time processing for HDTV resolution video at 60 fps.

6 Conclusion

In this research, VLSI implementation of an object detection algorithm using Sparse FIND feature for HDTV resolution video was studied. The performance evaluation result shows that when the rejection threshold at the HOG classification is −0.68 and the operating frequency is 130 MHz, the lowest frame rate 66.65 fps is attained, allowing 60 fps operation at HDTV resolution. The power simulation using the actual layout data results in 528 mW and 702 mW power consumption at 133 MHz, respectively at 0.81 V and 0.9 V. The accuracy degradation from the original Sparse FIND algorithm implemented on software is only 0.5%, which means it has a sufficient accuracy for practical use.

© IEICE 2017
DOI: 10.1587/elex.14.20170668
Received June 25, 2017
Accepted July 4, 2017
Publicized July 24, 2017
Copyedited August 10, 2017