IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

VOL. E101-A NO. 9
SEPTEMBER 2018

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. FUNDAMENTALS, VOL.E101–A, NO.9 SEPTEMBER 2018

PAPER Special Section on Intelligent Transport Systems

Hardware Architecture for High-Speed Object Detection Using Decision Tree Ensemble

Koichi MITSUNARI[†a)], ***Student Member***, **Jaehoon YU**[†], **Takao ONOYE**[†], ***and*** **Masanori HASHIMOTO**[†], ***Members***

SUMMARY Visual object detection on embedded systems involves a multi-objective optimization problem in the presence of trade-offs between power consumption, processing performance, and detection accuracy. For a new Pareto solution with high processing performance and low power consumption, this paper proposes a hardware architecture for decision tree ensemble using multiple channels of features. For efficient detection, the proposed architecture utilizes the dimensionality of feature channels in addition to parallelism in image space and adopts task scheduling to attain random memory access without conflict. Evaluation results show that an FPGA implementation of the proposed architecture with an aggregated channel features pedestrian detector can process 229 million samples per second at 100 MHz operation frequency while it requires a relatively small amount of resources. Consequently, the proposed architecture achieves 350 fps processing performance for 1080P Full HD images and outperforms conventional object detection hardware architectures developed for embedded systems.

key words: *decision tree ensemble, task scheduling, object detection, machine learning, embedded systems*

1. Introduction

Object detection is now an indispensable component of multiple practical applications on embedded systems such as advanced driving assistant system, robotics, and surveillance. These types of systems have strong constraints on power consumption, processing performance, and detection accuracy, which are in a trade-off relationship. Recently deep learning largely improved this trade-off, and features computed by deep convolutional networks achieved remarkable inference performance beyond hand-engineered features. However, considering the power constraint and the limited resources of embedded systems, such features that require computationally intensive networks are not always the best choice.

In the field of computer vision, multiple studies have proposed deep learning methods for real-time object detection [1]–[4]. The features for object detection are extracted through a fully convolutional network, and the computational cost is often reduced by a region of interests (RoI) estimator. Here, let us take Fast YOLO in [2] as an example since it is, to the best of our knowledge, the fastest method at the current stage. Fast YOLO processes 448x448 input images at more than 150 frames per second on Titan X GPU, which means the constraints of object detection accuracy and processing performance are satisfied. However, the other constraint of power consumption is not satisfied at all: inference using Titan X GPU requires approximately 200 Watts [5], and it is not affordable for embedded systems. Alternatively, when using Tegra GPUs for embedded use, the processing performance is not sufficient.

On the other hand, researchers in hardware community focus on conventional machine learning algorithms such as support vector machines (SVMs) and shallow neural networks (NNs) due to their implementation easiness [6]–[8]. SVMs and shallow NNs mainly consist of multiply accumulation operations and are implementable with a uniformly distributed array structure. However, it is a well-known fact that object detection using SVMs and shallow NNs suffer from poor detection accuracy and the circuit area of multipliers becomes a critical issue when designing an accelerator that exploits the high degree of parallelism. For addressing this issue, it is necessary to find out an object detection method with reasonably high detection accuracy and develop its efficient hardware for high processing performance.

In this point of view, we focused on conventional object detection methods using decision tree ensembles (DTE) [9], [10]. These object detection methods have multiple advantages in hardware implementation: low computational cost with soft cascade and multiplier-free operations in classification. Even with these attractive features, only a few studies on DTE hardware architectures are reported [11], because conditional branches of decision stumps composing a DTE require random memory accesses, which prevents efficient parallel processing.

As a solution, this paper proposes a hardware architecture for DTEs. The proposed architecture processes object detection in a SIMD-like homomorphic manner even while DTEs are adopted as a classifier. Also, this paper proposes a task scheduling algorithm to control memory accesses from multiple modules to improve processing performance. The proposed architecture has two distinctive features as follows. First, it supports three-dimensional parallel memory access, 1-D for feature channels and 2-D for image space, achieving multiple times higher processing performance than conventional hardware architectures. Second, it takes advantage of algorithmic acceleration by using soft cascade, which improves processing performance by over one to two orders of magnitude. For evaluating the proposed DTE hardware architecture, we assume the feature extraction method called aggregated channel features (ACF) [10] and classification us-

Manuscript received December 4, 2017.
Manuscript revised April 20, 2018.

[†]The authors are with Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565-0871 Japan.
a) E-mail: k-mitunr@ist.osaka-u.ac.jp

DOI: 10.1587/transfun.E101.A.1298

ing multi-scale classifiers with octave-wise feature maps [9].

The rest of this paper is organized as follows. Section 2 explains DTEs and conventional hardware architectures. Section 3 provides a hardware architecture for DTEs, and Sect. 4 proposes a task scheduling algorithm to handle memory access conflict. Section 5 describes evaluation results and provides an analysis of the proposed hardware architecture. Section 6 concludes the paper.

2. Target Application and Available Architectures

This section briefly explains the object detection method based on the DTE using ACF [10], which is the target of hardware implementation in this work, and introduces DTE hardware architectures proposed so far.

2.1 Decision Tree Ensemble using Aggregated Channel Features

The underlying idea of ensemble learning is to boost the final learner's predictive performance by accumulating weighted votes from weak learners whose predictive performance is merely better than random guessing. As shown in Fig. 1, a DTE is one of ensemble learning methods using multiple decision trees (DTs) as weak learners. Each DT consists of decision nodes and leaf nodes, where a decision node selects one of its child nodes based on the comparison result between its input and threshold, and a selected leaf node returns its evaluation value. Given an input feature vector $\mathbf{x}$, the final learner H is defined as

$$H(\mathbf{x}) = \mathrm{sign}\left(\sum_{i=1}^{T} h_i(\mathbf{x})\right), \tag{1}$$

where h_i is the prediction function of the i-th DT and returns the value of a selected leaf node, and T is the number of DTs in the DTE. Compared with recent deep learning algorithms, a DTE is a shallow machine learning algorithm. However, it is reasonably deep and shows good classification performance for practical applications [12], which will also be demonstrated experimentally in Sect. 4.

In computer vision, non-rigid object detection has been a challenging issue and studied for several decades. Of many existing methods, Dollár et al. proposed a remarkably efficient and accurate object detection method based on the DTE using ACF [10]. Figure 2 shows the processing flow of object detection using ACF. As shown in Fig. 2, ACF consists of ten channels extracted from three types of features: six channels from HOG, three channels from LUV color space, and one channel from gradient magnitudes. ACF calculates raw features of each channel, aggregates each 4x4 block to build an aggregated channel, and classifies input data extracted from sliding window sampling. Non-maximum suppression clusters detection results corresponding an object into one. In [10], Dollár et al. reported that DTE using ACF achieved 17% log-average miss rate (MR) and 31.9 fps processing performance on a single CPU, which outperforms other types of state-of-the-art methods. However, to implement a hardware for exploiting ACF, it is necessary to clarify how to utilize multiple channels of features in parallel computation.

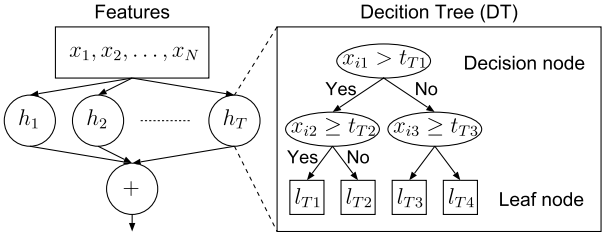


Fig. 1 Depth-two binary DTE.

2.2 Hardware Architectures for Decision Tree Ensemble

There exist multiple hardware architectures for DTEs [13], [14], and Struharik and Novak classified them into three types in [15]: threshold networks, single-path architectures, and single-node architectures. Figure 3 shows a threshold network, a single-path architecture, and a single-node architecture, which are available implementations for a depth-two DT. The threshold network, shown in Fig. 3(b), is an architecture that processes all decision nodes of a DT in parallel, calculating an output O as follows:

$$O = l_1(d_1 d_2) + l_2(d_1 \bar{d}_2) + l_3(\bar{d}_1 d_3) + l_4(\bar{d}_1 \bar{d}_3), \tag{2}$$

where d_i is the binary response of the i-th decision node, 0 or 1, and l_j is the value of the j-th leaf node. Since threshold networks enable to calculate output instantly after input, it is suitable for applications requiring short time delay between input and output. The single-path architecture, shown in Fig. 3(c), is an architecture that has pipeline stages of universal nodes, where the number of pipeline stages is equal to the depth of the DT, and the universal nodes are processing elements to carry out the function of decision nodes. Since single-path architectures adopt a pipelined homogeneous structure, it achieves equivalent throughput to the corresponding threshold network with a relatively small

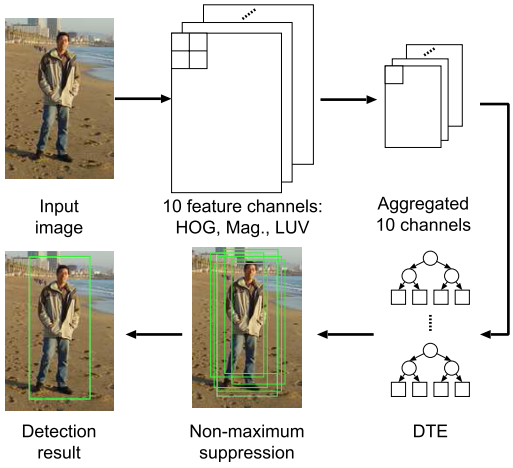


Fig. 2 Object detection flow with DTE using ACF.

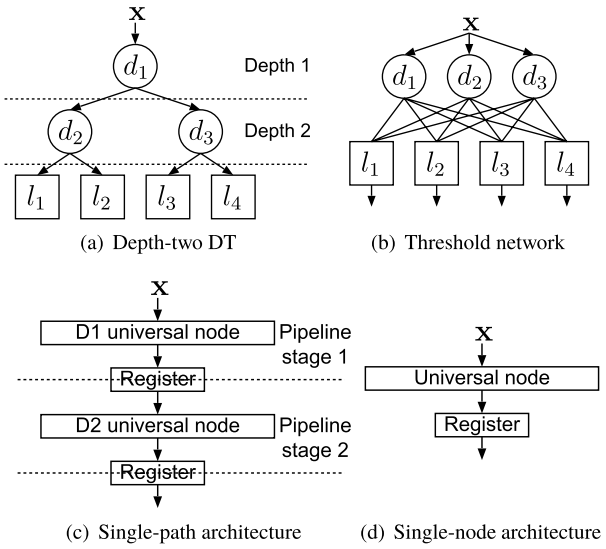


Fig. 3 DTE and available hardware architectures.

amount of hardware resources. The single-node architecture, shown in Fig. 3(d), also uses universal nodes as processing elements but does not have pipeline stages. The single-node architecture has more flexibility in its design than the others mentioned above in that it can handle any processing order of decision nodes and there exist multiple hardware architectures for storing the responses of decision nodes. However, its processing performance and required hardware resources largely depend on the design. Therefore, for the hardware implementation based on the single-node architecture, the architecture design plays an important role.

3. Parallel Implementation of Decision Tree Ensembles Using Multiple Memory Banks

The proposed hardware architecture is a single-node architecture designed for exploiting multiple memory banks with a small amount of routing resources. This section explains its overview and details in order.

3.1 Architecture Overview

ACF has multiple types of features and requires sophisticated memory access patterns. Considering the parallel feature extraction before classification, we need to allocate a dedicated memory bank for each channel. In this case, the threshold network and the single-path architecture are not suitable because they require a massive amount of routing resources for supporting random memory access to all the banks. On the other hand, the single-node architecture can resolve the routing resource problem by assigning a universal node to each channel and merging the responses of decision nodes belonging to each DT.

The proposed architecture is designed based on the idea mentioned above. Figure 4 shows the overview of the proposed hardware architecture, and Table 1 lists the notations used in Fig. 4. The proposed architecture mainly consists of three sub-modules: `decisionNodeCube`, `leafNodeCube`, and `ctrl`, where they are a 3-D array of decision nodes, a 3-D array of leaf nodes, and a control unit, respectively. The `decisionNodeCube` consists of C `decisionNodeMatrix` modules, and the `leafNodeCube` consists of M `leafNodeMatrix` modules, an `accumMatrix` module, and a `chSelMem` module, where M is less or equal to C due to the non-uniformity of the channel usage described in Sect. 4. The `decisionNodeCube` receives feature input F_{in} and outputs decision responses D_{res}. The `leafNodeCube` selects corresponding leaf values L_{val} from a part of the decision responses D_{res} and accumulates the leaf values to calculate the final responses A_{res}.

This architecture enables a 3-D parallel classification, and the hardware handles a massive amount of data. Thus, in hardware design, the scalability of the architecture for each sub-module, `decisionNodeMatrix`, `leafNodeMatrix`, and `accumMatrix` needs to be carefully considered, which are discussed in Sect. 3.2.

3.2 Details of Sub-Modules

In the proposed architecture, its processing flow completely depends on the `ctrl` and the `chSelMem` modules. The `ctrl` observes the states of all the sub-modules and dynamically provides control signals, and the `chSelMem` provides static task schedules generated by the proposed scheduling algorithm described in Sect. 4. Therefore, the proposed architecture can handle any DTEs by updating task schedules.

Each of C `decisionNodeMatrix` modules composing the `decisionNodeCube` corresponds to one of C input feature channels: HOG channels, LUV channels, and a gradient magnitude channel described in Sect. 2. Each `decisionNodeMatrix` consists of three sub-modules: a `decisionMem`, a `featureMem`, and a 2-D array of $W_{\text{node}} \times H_{\text{node}}$ `decisionNode` modules. In the `decisionNodeMatrix`, the `decisionMem` is the only module controlled by the signal c_{d} from the control unit, and the data, d_{x}, d_{y}, and d_{t}, loaded from `decisionMem` controls the others. The `featureMem` provides $W_{\text{node}} \times H_{\text{node}}$ feature values, f, of the block at the $(d_{\text{x}}, d_{\text{y}})$ position to `decisionNode` modules. To support loading the feature block at an arbitrary position, `featureMem` uses H_{node} dual port line buffers, `lineBuffer`, and H_{node} shift registers, `horShiftReg`, for location adjustment. The dual port line buffers enable to load malaligned data at any vertical position with a single cycle, and shift registers enable to extract the target data columns at any horizontal position. Each `decisionNode` generates a 1-bit comparison result as the decision response between a feature value of f and a threshold d_{t} as shown in Fig. 5(b), where d_{t} is the threshold shared in all `decisionNode` modules of a `decisionNodeMatrix`.

Each of M `leafNodeMatrix` modules composing the `leafNodeCube` consists of three sub-modules: a `leafMem`, a `ringShiftReg`, and a 2-D array of $W_{\text{node}} \times H_{\text{node}}$ `leafNode` modules. The `leafMem` provides all leaf values of each DT to `leafNodeMatrix`, the `ringShiftReg` vertically and

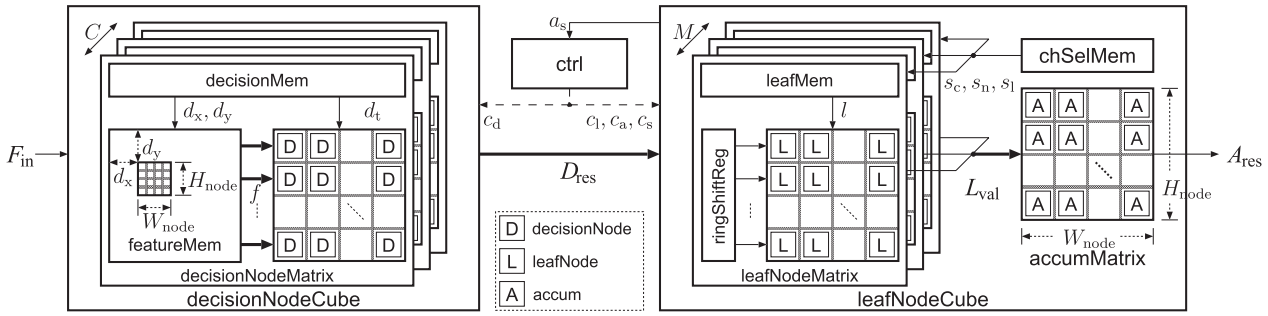


Fig. 4 Hardware architecture overview.

Table 1 Notation and description of each module.

Module	Notation	Description
Top Module	$\boldsymbol{F}_{\text{in}}$	A set of input features
	$\boldsymbol{D}_{\text{res}}$	A set of decision responses
	$\boldsymbol{L}_{\text{val}}$	A set of leaf values
	$\boldsymbol{A}_{\text{res}}$	A set of accumulated responses
	W_{node}	#horizontal D/L/A nodes of a matrix
	H_{node}	#vertical D/L/A nodes of a matrix
	C	#decision node matrices
	M	#leaf node matrices
decisionMem	d_{x}	Feature x position
	d_{y}	Feature y position
	d_{t}	Decision node threshold
featureMem	f	All feature values of a block
leafMem	l	All leaf values of a DT
accumMatrix	a_{s}	Sign bit for soft cascade
chSelMem	s_{c}	Channel index
	s_{n}	Node index
	s_{l}	Last node flag
ctrl	c_{s}	Address control signal
	c_{d}	decisionNodeCube control signal
	c_{l}	leafNodeCube control signal
	c_{a}	accumMatrix control signal
Misc.	d_{res}	A decision response
	l_{val}	A leaf value
	a_{res}	An accumulated response

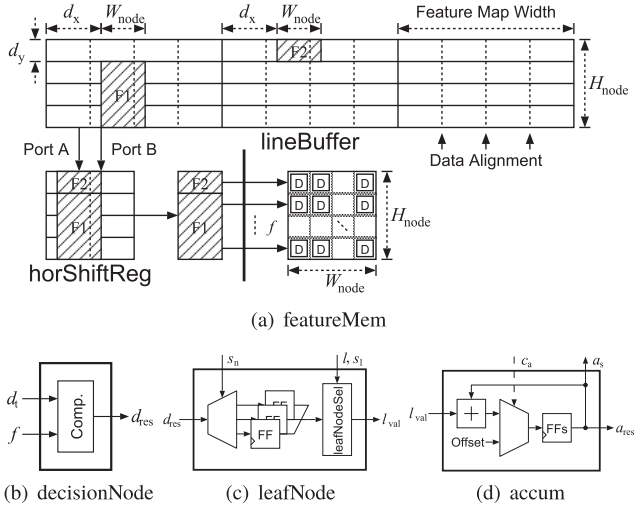


Fig. 5 Block diagrams of processing elements.

horizontally rotates the $W_{\text{node}} \times H_{\text{node}} \times C$ 1-bit decision responses to correct positions, and `leafNode` selects a leaf value from l based on each series of decision responses. Figure 5(c) shows the details of the `leafNode` for depth-two DTEs. As shown in Fig. 5(c), the `leafNode` includes a demultiplexer for rearranging the order of decision responses, three flip-flops (FFs) for storing the decision responses of a depth-two DT, a `leafNodeSel` for selecting a leaf value based on the responses. This structure enables the `leafNode` to handle the random input order of decision responses and to improve the processing performance by task scheduling technique. Also, a simple modification of the `leafNode` allows to handle DTEs deeper than depth-two DTEs: for processing depth-three DTEs, the `leafNode` requires 7-bit FFs to store seven decision responses, and the `leafNodeSel` requires an extension to select a leaf value of eight leaf values.

The `accumMatrix` consists of $W_{\text{node}} \times H_{\text{node}}$ `accum` modules. Figure 5(d) describes the details of the `accum`. The `accum` accumulates the leaf values of each DTE by using an adder and FFs, where the FFs are initialized with the offset, which is used for soft cascade rejection. When the static threshold of the soft cascade is $-s$, the offset is set to s, so that the control unit can decide soft cascade rejection only with the sign bit of the accumulated value, a_{s}.

4. Task Scheduling for Parallel Implementation

The proposed hardware architecture can process decision nodes of a DT in random order, and its processing performance depends on the efficiency of the parallel memory access. For further acceleration, we propose a task scheduling algorithm dedicated to the proposed architecture. This task scheduling is an optimization problem considering each decision node as a task subject to two constraints derived from the architecture design. This section explains how to formulate the task scheduling problem, describes the proposed algorithm, and analyzes its effectiveness.

4.1 Decision Tree Ensemble Scheduling Problem

Memory accesses resulted from conditional branches may cause memory conflict while processing multiple DTs at once. The purpose of task scheduling is to avoid this memory conflict by processing all decision nodes of a DT in a fixed order as shown in Fig. 6 and controlling parallel memory accesses from multiple DTs. Given a DTE classifier,

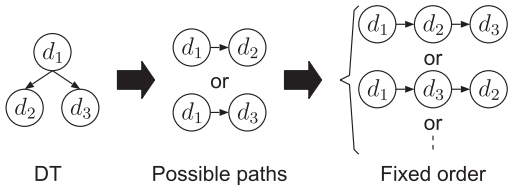


Fig. 6 Input dependency removal of DT by visiting all nodes.

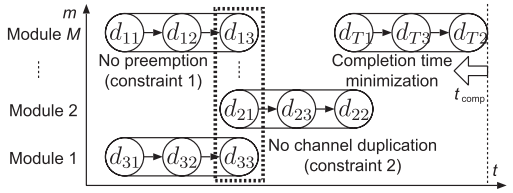


Fig. 7 Target scheduling problem.

this scheduling algorithm fixes a task schedule in an offline manner, and classification requires no additional computation for scheduling. The task scheduling is an optimization problem finding the minimum completion time t^*_{comp} and its assignment matrix A^* for M modules defined as

$$t^*_{\text{comp}} := \min t_{\text{comp}}(A), \quad A^* := \operatorname*{argmin}_A t_{\text{comp}}(A), \tag{3}$$

where $t_{\text{comp}}(A) = \max\{t \mid \exists m \in \{1, \ldots, M\}, a_{mt} \neq 0\}$, in which $t_{\text{comp}}(A)$ is the completion time using an assignment matrix A, and a_{mt} is the (m, t)-th entry of A representing the decision node processed on the m-th module at the t-th cycle. If there is no task assignment, a_{mt} will be zero. The two constraints of this scheduling problem are defined as follows. When a decision node d_{ij} is assigned to $a_{m_{ij}t_{ij}}$, the first constraint is

$$\forall i \in \{1, \ldots, T\}, \forall j_1, j_2 \in \{1, \ldots, S\}, m_{ij_1} = m_{ij_2}, \tag{4}$$

where S is the number of decision nodes in a DT, and for any i, each t_{ij} needs to be a consecutive number. The second constraint is

$$\forall t \in \{1, \ldots, T_{\max}\}, \forall m_1, m_2 \in \{1, \ldots, M\}, m_1 \neq m_2, \\ c(a_{m_1 t}) \neq 0, c(a_{m_2 t}) \neq 0, c(a_{m_1 t}) \neq c(a_{m_2 t}), \tag{5}$$

where $c(d_{ij})$ represents the channel used in d_{ij}, and $T_{\max}$ is the possible maximum completion time. These constraints represent that an identical `leafNode` processes all decision nodes belonging to a DT without preemption, and each `leafNodeMatrix` exclusively uses decision responses from a `decisionNodeMatrix` at each time, respectively. Then, the scheduling problem can be defined as an offline problem as shown in Fig. 7. This scheduling problem can be considered as an extension of the $\mathcal{NP}$-hard job shop scheduling problem.

4.2 Proposed Heuristic Scheduling Algorithm

As mentioned above, the target scheduling problem is $\mathcal{NP}$-hard, and it is difficult to find the optimal solution t^*_{comp}.

Algorithm 1 Task scheduling of a DTE

Input:
$H := \{h_i \mid h_i = \{d_{i1}, \ldots, d_{iS}\}, 1 \leq i \leq T\}$,
$M :=$ parallel degree
Output:
$A =$ Assignment matrix
$t^*_{\text{comp}} =$ completion time of A

$c \leftarrow$ SortAndMergeChannels(H)
$T_{\max} \leftarrow ST$
$A \leftarrow O \in \mathbb{N}_+^{M \times T_{\max}}$
for $n = 1$ to M **do**
  $H_n \leftarrow \{h_n \mid \exists j, c(d_{ij}) = n\}$
  for all $h \in H_n$ **do**
    $(m^*, t^*_{\text{tgt}}, t^*) \leftarrow (0, T_{\max}, T_{\max})$
    $P(h)$: a set of tuples consisting of permutation of h
    for all $P \in P(h)$ **do**
      $(m, t) \leftarrow$ SearchWithConstraint(A, P)
      $t_{\text{tgt}} \leftarrow t + \max\{i \mid c(p_i) = n\}$
      if $((t_{\text{tgt}} < t^*_{\text{tgt}}) \vee ((t_{\text{tgt}} = t^*_{\text{tgt}}) \wedge (t < t^*)))$ **then**
        $(m^*, t^*_{\text{tgt}}, t^*) \leftarrow (m, t_{\text{tgt}}, t)$, $P^* \leftarrow P$
      end if
    end for
    for $s = 1$ to $|P^*|$ **do**
      $a_{m^*, t^*+s} \leftarrow P^*_s$
    end for
  end for
  $H \leftarrow H \setminus H_n$
end for
$t^*_{\text{comp}} \leftarrow \max\{t \mid \exists m \in \{1, \ldots, M\}, a_{mt} \neq 0\}$

procedure SortAndMergeChannels(H)
  $C_{\text{hist}} \leftarrow$ calculate channel histogram from H
  $k \leftarrow |C_{\text{hist}}|$
  while $k > M$ **do**
    $(c_1, \ldots, c_{k-1}, c_k) \leftarrow$ sort C_{hist} in descending order
    $C_{\text{hist}} \leftarrow (c_1, \ldots, c_{k-2}, c_{k-1} + c_k)$: merge channels
    Update $c(d_{ij})$
    $k \leftarrow k - 1$
  end while
  $(c_1, \ldots, c_{k-1}, c_k) \leftarrow$ sort C_{hist} in descending order
  Update $c(d_{ij})$
  return c
end procedure

procedure SearchWithConstraint(A, P)
  $(m^*, t^*) \leftarrow (0, T_{\max} - |P|)$
  for $(m, t) \in \{1, \ldots, M\} \times \{1, \ldots, t^*\}$ **do**
    if $(\forall (m', t') \in \{1, \ldots, M\} \times \{1, \ldots, |P|\},$
        $a_{m, t+t'} = 0 \wedge c(a_{m', t+t'}) \neq c(p_{t'}))$ **then**
      if $((t < t^*))$ **then**
        $(m^*, t^*) \leftarrow (m, t)$
      end if
    end if
  end for
  return (m^*, t^*)
end procedure

Thus, the proposed algorithm aims to find a solution which is close to the lower bound, where the lower bound is equal to the maximum number of frequency in channel histogram. The proposed algorithm adopts a greedy approach and focuses on the frequency of channels in a DTE. The assignment is in the order that the frequency of channels represents the priority, which reduces the number of assignment candidates and reduces the amount of computation. Also, for improving

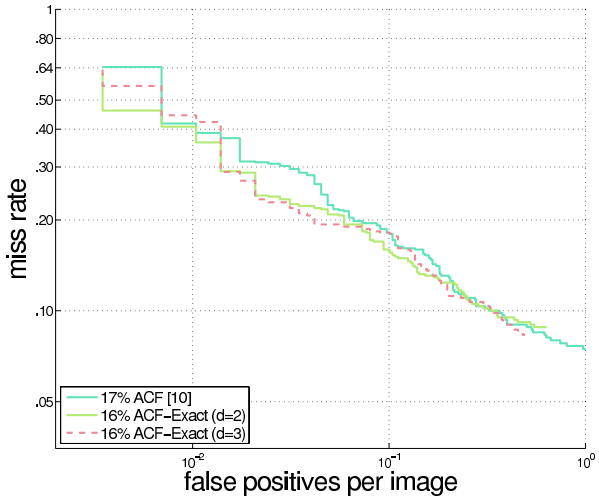
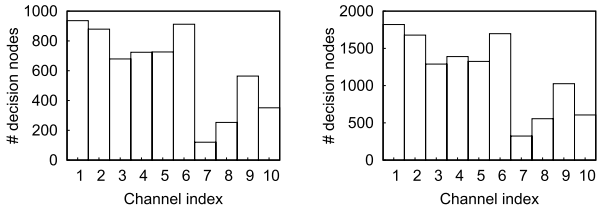
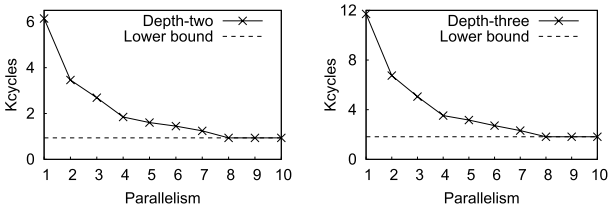


Fig. 8 Detection error trade-off curves on INRIA Person Dataset.



(a) Depth-two classifier (b) Depth-three classifier

Fig. 9 Histograms of input channels.



(a) Depth-two classifier (b) Depth-three classifier

Fig. 10 Scheduling results for the different number of modules.

the completion time, it is a promising approach to make a flat histogram by reducing the number of channels considering the variations of the frequency in the channel histogram.

Algorithm 1 shows the proposed algorithm. As a preprocessing, the proposed method merges the input C channels to M channels, where the two channels of lowest frequencies are merged in each iteration as described in lines 23–35. The assignment process consists of M iterations of the merged channels, and in its n-th iteration, DTs containing the channel n, represented as H_n, are assigned. For each DT h in H_n, the proposed algorithm searches the assignment position satisfying the constraint described in Eqs. (4) and (5) for all the patterns of processing orders as shown in lines 36–46. From all of the processing orders, the one with the earliest completion time and the channel n is selected using the condition shown in line 12, and it is assigned to an assignment matrix.

4.3 Analysis of Scheduling Algorithm

In the analysis, the target classifiers are depth-two and depth-three ACF classifiers trained in the same manner as [10], consisting of 2,048 and 1,673 DTs. The MATLAB evaluation code of Caltech Pedestrian Detection Benchmark [16] is used to evaluate detection accuracy. The log-average MRs on INRIA Person Dataset [17] are 16.5% and 16.3%, respectively. Figure 8 shows the detection error trade-off curves of these two classifiers and the classifier reported in [10], where our classifiers achieve equivalent detection accuracy to the original ACF classifier. Figure 9 shows the histograms of input channels for these classifiers, which indicates that there exists large variance of frequencies among channels in both histograms. Taking into account the memory access exclusiveness, the lower bound for this problem is equal to the maximum number of decision nodes in a channel. Figure 10 shows the relationship between the parallel degree M and the number of the cycles required for processing the classifiers based on the task schedules. For both classifiers, the number of processing cycles decreases as M increases until 8. When M is equal to 8, both numbers of processing cycles reach the lower bound drawn in dotted lines. Compared with serial classification, the proposed scheduling achieves 6.6 and 6.4 times speed up for the depth-two and depth-three classifiers, respectively. For more details, Table 2 lists the number of processing cycles and the occupancy of the `leafNodeCube`. From the result, the proposed scheduling reduces the number of cycles to the lower bound. Also, using soft cascade enables to accelerate the processing performance of negative windows. In Table 2, c_{neg} represents the average number of processing cycles for negative windows, and Table 2 shows that combining the proposed task scheduling and soft cascade can reduce both average cycles of depth-two and depth-three ACF classifiers to 3.3% and 5.4% of processing cycles required for a positive window.

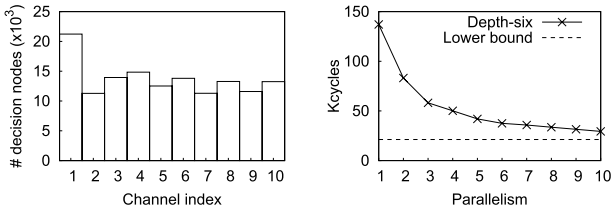
Table 2 Detailed scheduling result on $M = 8$.

Depth	Lower bound / t_{comp}	Occupancy	c_{neg}
2	936 / 936	82.1%	50.9
3	1,820 / 1,820	80.4%	59.4

4.4 Scheduling under Deeper Decision Tree Ensemble

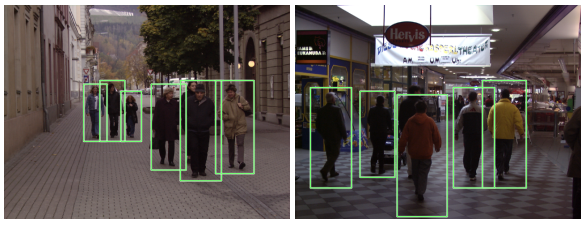
Recent work [18] reports that deeper DTs show good detection performance. To analyze the relationship between the task scheduling performance and the depth of DTs, the proposed task scheduling is applied to a deep DTE classifier. The evaluation uses a depth-six classifier provided by the authors[†], which is trained for Caltech Pedestrian Detection Benchmark [16]. The classifier consists of 3,324 DTs, and the number of decision nodes in the classifier is 137,043. The number of available permutations calculated in line 8 in Algorithm 1 is exponentially proportional to the depth

[†]https://eshed1.github.io/code/BoostICPR.zip



(a) Histogram of input channels (b) Scheduling results

Fig. 11 Result of depth-six ACF classifier.



(a) person_191 (b) person_217

Fig. 12 Detection results from INRIA Person Dataset.

of a DT, and then it is necessary to reduce the number of candidates for deep DTs. For mitigating this, the processing order is fixed to the order of channel frequency in this experiment. Figures 11(a) and 11(b) show its channel histogram and the scheduling results, respectively. The result shows the similar convergence curve to the shallow DTE and achieves 4.1 times speed-up compared with the serial implementation when M is equal to 8. However, the processing cycles do not reach the lower bound even when the parallelism is equal to the channel since Eq. (5) is difficult to satisfy for all the decision nodes. Improving the task scheduling for deeper DTs is included in our future work.

5. Evaluation

This section explains how to generate a fixed-point classifier and implementation settings, used in the evaluation, and evaluates the FPGA implementation based on the proposed hardware architecture regarding resource usage and processing performance.

5.1 Evaluation Settings

For hardware implementation, we converted the depth-two DTE described in Sect. 4 into a classifier in fixed-point representation by using the method proposed in [19]. Figure 12 shows detection results of the converted fixed-point classifier. The proposed hardware architecture is implemented using Verilog hardware description language (HDL) at register transfer level (RTL). Table 3 shows the implementation settings. The target device is Xilinx xc7z045ffg900, the target operating frequency is 100 MHz, and the degree of parallelism is 1,024: parallel degree 8 from feature channels and 128 from image blocks, respectively. In feature extraction, we use three types of feature descriptors, i.e., HOG, gradient magnitude, and RGB color channels. We adopted RGB channels instead of LUV channels because the difference of color channels does not cause notable accuracy loss and converting to LUV channels is computationally intensive [20]. Also, for hardware implementation efficiency, we assumed the classification procedure proposed by Benenson et al. [9], which uses multiple classifiers corresponding to different window sizes and feature maps extracted from scaled images, instead of the genuine ACF classification procedure using a classifier and the fast feature pyramid (FFP) proposed in [10]. Although FFP shows efficient memory usage and higher processing performance for software implementation, it is not suitable for hardware implementation because FFP needs to generate each layer of feature pyramid sequentially.

Table 3 FPGA implementation settings.

Target device	Xilinx xc7z045ffg900
Synthesis tool	Vivado 2015.4.2
Simulation tool	ModelSim SE-64 10.3
Target frequency	100 MHz
Parallelism	1,024 (channel: 8, block size: 8x16)

5.2 Resource Usage

For the evaluation of resource utilization, we synthesized the RTL implementation with Vivado 2015.4.2. Table 4 shows the resource utilization of the proposed implementation. As in Table 4, it occupies less than 35% of both slice and block RAM resources of the target FPGA for processing 1,024 decision nodes in parallel. Also, from the details of LUT usage, the balanced use of both LUTs and FFs can be confirmed.

5.3 Processing Performance

For the evaluation of processing performance, we simulated the RTL implementation with actual input images on ModelSim SE-64 10.3. In the simulation, the classification process takes 45,809 cycles or 0.46 milliseconds for a full HD image without scaling. Since processing time is linearly proportional to the image resolution, when the processing time t_{single} represents the required cycle or time for a single-scale full HD image, the entire processing time t_{all} for full search detection with sliding-window sampling is defined as follows:

$$t_{\text{all}} = \sum_{i=1}^{N_{\text{scale}}} \frac{t_{\text{single}}}{S_{\text{scale}}^{2i}}, \tag{6}$$

where N_{scale} is the number of scale images. Given t_{single} is 0.46 and the parameters shown in Table 5, the processing time t_{all} becomes 2.86 milliseconds. Thus, the proposed hardware enables to process full HD images at 350 fps.

Table 5 provides a processing performance comparison between the proposed method and three conventional methods: an ACF software implementation [10], an ACF hardware implementation [11], and a deformable part model (DPM) hardware implementation [8]. The DPM hardware

Table 4 FPGA resource utilization.

Module	Slice		LUT (Logic)		LUT (Memory)		LUT (FF)		32 Kb BRAM		DSP	
decisionNodeCube	7,796	(14.3%)	23,891	(10.9%)	0	(0.0%)	24,420	(11.2%)	170	(31.2%)	0	(0.0%)
→ 10 featureMem	5,649	(10.3%)	15,609	(7.1%)	0	(0.0%)	15,602	(7.1%)	160	(29.4%)	0	(0.0%)
→ 10 decisionMem	163	(0.3%)	252	(0.1%)	0	(0.0%)	262	(0.1%)	10	(1.8%)	0	(0.0%)
leafNodeCube	10,822	(19.8%)	35,143	(16.1%)	1	(0.0%)	36,839	(16.9%)	16	(2.9%)	0	(0.0%)
→ 8 leafMem	476	(0.9%)	1,114	(0.5%)	0	(0.0%)	1,125	(0.5%)	8	(1.5%)	0	(0.0%)
→ chSelMem	1,090	(2.0%)	2,086	(1.0%)	0	(0.0%)	2,086	(1.0%)	8	(1.5%)	0	(0.0%)
→ accumMatrix	1,952	(3.6%)	5,541	(2.5%)	0	(0.0%)	5,557	(2.5%)	0	(0.0%)	0	(0.0%)
ctrl	286	(0.7%)	503	(0.2%)	0	(0.0%)	420	(0.2%)	0	(0.0%)	1	(0.1%)
→ decisionNodeCtrl	194	(0.4%)	396	(0.2%)	0	(0.0%)	315	(0.1%)	0	(0.0%)	1	(0.1%)
→ leafNodeCtrl	92	(0.2%)	107	(0.0%)	0	(0.0%)	105	(0.0%)	0	(0.0%)	0	(0.0%)
Total	18,904	(34.6%)	59,537	(27.2%)	1	(0.0%)	61,679	(28.2%)	186	(34.1%)	1	(0.1%)

Table 5 Processing performance comparison with conventional implementations.

Method	Platform	Image size $W_{\text{img}} \times H_{\text{img}}$	Window size $W_{\text{win}} \times H_{\text{win}}$	Scale step S_{scale}	Pixel step S_{pixel}	fps N_{fps}	#window/sec. N_{wps}	
ACF [10]	CPU	640×480	48×96	$2^{1/8}$	4	31.9	2,181k	(1/105.0)
ACF [11]	FPGA	640×480	32×64	$2^{1/6}$	4	30	1,972k	(1/116.1)
DPM [8]	ASIC	$1,920 \times 1,080$	64×128	$2^{1/3}$	8	60	3,975k	(1/57.6)
Ours	FPGA	$1,920 \times 1,080$	48×96	$2^{1/8}$	4	350	229,079k	(1.0)

implementation is the fastest hardware implementation so far, using a deformable part model [8]. Evaluation based on frame rate does not provide precise result because it does not use detailed implementation settings, and window-based evaluation is suitable for a fair comparison [21]. Here, window-based evaluation is adopted, and the processing performance is evaluated by recalculating to processed windows per second, N_{wps}, using the following equation:

$$N_{\text{wps}} = \frac{N_{\text{fps}}}{S_{\text{pixel}}^2} \sum_{i=1}^{N_{\text{scale}}} \left(\frac{W_{\text{img}}}{S_{\text{scale}}^i} - W_{\text{win}} \right) \left(\frac{H_{\text{img}}}{S_{\text{scale}}^i} - H_{\text{win}} \right). \quad (7)$$

As shown in Table 5, the proposed implementation is 105.0 and 116.1 times faster than the software and the hardware implementations of ACF, respectively. Also, it is 57.6 times faster compared with the DPM hardware implementation, which was the fastest implementation.

5.4 Discussion

The processing performance of practical applications depends on both feature extraction and classification. So far, we have shown the proposed hardware architecture can process 350 fps for full HD images. Now, we discuss the processing performance of the feature extraction part. We assumed three feature descriptors as mentioned the above. To verify the feasibility of our assumption, we implemented feature extraction modules in Verilog HDL and evaluated them in terms of throughput and resource utilization. Table 6 summarizes the implementation result with 32 degrees of parallelism, where the parallelism comes from eight channels and four scaled images from a full HD image. As in Table 6, the entire utilization of feature extraction modules is less than 10% of slices. With 32 degrees of parallelism, the feature extraction modules can achieve 60 fps processing performance for full HD images. However, it does not seem to be enough for providing feature maps to the proposed DTE classifier. Besides, to realize N-class object detection, the proposed DTE classifier needs to process $N\times$ faster than the feature extraction modules. In this case, the proposed DTE classifier with the assumed feature extraction modules can classify five classes of objects simultaneously.

Table 6 Resource usage of feature extraction with 32 degrees of parallelism.

Feature	Slice LUTs		Slice registers		Throughput
HOG [22]	12,288	(5.6%)	1,440	(0.3%)	32 features/cycle
RGB	6,144	(2.8%)	6,144	(1.4%)	32 features/cycle
Magnitude	2,272	(1.0%)	1,764	(0.4%)	32 features/cycle

6. Conclusion

For practical applications using visual object detection, the improvement of the trade-offs between hardware resources, processing performance, and detection accuracy has been a critical issue, and the proposed architecture successfully resolved this issue by improving classification speed without detection accuracy degradation. The proposed architecture adopted a hardware and software cooperative design and is distinctive from other existing architectures. The hardware implementation based on the single-node architecture exploits its resources by using the proposed task scheduling method. Since the task schedules are static within a DTE, once one fixed the task schedules of a DTE, there is no processing overhead in detection phase. Also, the task schedule focusing on the lower bound of required cycles clarified that the efficient parallel degree is less than the number of feature channels used in ACF, and made it possible to reduce the hardware resource for leaf nodes without lowering processing performance. In the evaluation, the proposed architecture achieved more than 100 times faster than conventional ACF implementations without any detection accuracy degradation and more than 50 times faster than the fastest DPM

implementation. The proposed method outperforms the conventional methods in terms of the scalability and processing performance.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP16K16085.

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," Proc. Adv. Neural Inform. Process. Syst., pp.91–99, Dec. 2015.
[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit., pp.779–788, June 2016.
[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A.C. Berg, "SSD: Single shot multibox detector," Proc. Eur. Conf. Comput. Vis., pp.21–37, Oct. 2016.
[4] T.Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit., pp.936–944, July 2017.
[5] NVIDIA, "GPU-based deep learning inference: A performance and power analysis," http://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf, Nov. 2015.
[6] A. Suleiman and V. Sze, "An energy-efficient hardware implementation of HOG-based object detection at 1080HD 60 fps with multi-scale support," J. Signal Process. Syst., vol.84, no.3, pp.325–337, Sept. 2016.
[7] G.M. Lozito, A. Laudani, F.R. Fulginei, and A. Salvini, "FPGA implementations of feed forward neural network by using floating point hardware accelerators," Advances Elect. and Electron. Eng., vol.12, no.1, pp.30–39, 2014.
[8] A. Suleiman, Z. Zhang, and V. Sze, "A 58.6 mW 30 frames/s real-time programmable multiobject detection accelerator with deformable parts models on full HD 1920 × 1080 videos," IEEE J. Solid-State Circuits, vol.52, no.3, pp.844–855, March 2017.
[9] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool, "Pedestrian detection at 100 frames per second," Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit., pp.2903–2910, June 2012.
[10] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," IEEE Trans. Pattern Anal. Mach. Intell., vol.36, no.8, pp.1532–1545, Aug. 2014.
[11] H. Song, B. Jeong, H. Choi, T. Cho, and H. Chung, "Hardware implementation of aggregated channel features for ADAS," Proc. Int. SoC Des. Conf., pp.167–168, Oct. 2016.
[12] Y. Bengio, O. Delalleau, and C. Simard, "Decision trees do not generalize to new variations," Comput. Intell., vol.26, no.4, pp.449–467, Nov. 2010.
[13] A. Bermak and D. Martinez, "A compact 3D VLSI classifier using bagging threshold network ensembles," IEEE Trans. Neural Netw., vol.14, no.5, pp.1097–1109, Sept. 2003.
[14] M. Owaida, H. Zhang, C. Zhang, and G. Alonso, "Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms," Proc. Int. Conf. Field-Programmable Logic and Appl., pp.1–8, Sept. 2017.
[15] R.J.R. Struharik and L.A. Novak, "Hardware implementation of decision tree ensembles," J. Circuits, Syst., Comput., vol.22, no.05, p.1350032, 2013.
[16] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," IEEE Trans. Pattern Anal. Mach. Intell., vol.34, no.4, pp.743–761, April 2012.
[17] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit., pp.886–893, June 2005.
[18] E. Ohn-Bar and M.M. Trivedi, "To boost or not to boost? on the limits of boosted trees for object detection," Proc. Int. Conf. Pattern Recognit., pp.3350–3355, Dec. 2016.
[19] K. Mitsunari and J. Yu, "Influence of numerical precision on machine learning and embedded systems," Proc. Int. Workshop Smart Info-Media Syst. Asia, pp.164–169, Sept. 2016.
[20] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," Proc. Brit. Mach. Vis. Conf., pp.91.1–91.11, Sept. 2009.
[21] X. Ma, W.A. Najjar, and A.K. Roy-Chowdhury, "Evaluation and acceleration of high-throughput fixed-point object detection on FPGAs," IEEE Trans. Circuits Syst. Video Technol., vol.25, no.6, pp.1051–1062, June 2015.
[22] P.-Y. Chen, C.-C. Huang, C.-Y. Lien, and Y.-H. Tsai, "An efficient hardware implementation of Hog feature extraction for human detection," IEEE Trans. Intell. Transp. Syst., vol.15, no.2, pp.656–662, April 2014.

Koichi Mitsunari received his Master's degree in information science from Osaka University, Osaka, Japan, in 2016. He is currently a Ph.D. student at Graduate School of Information Science and Technology, Osaka University. His research interests include computer vision, machine learning, and pattern recognition. He is a student member of IEEE and IPSJ.

Jaehoon Yu received his B.E. degree in Electrical and Electronic Engineering and his M.S. degree in Communications and Computer Engineering from Kyoto University, Kyoto, Japan, in 2005 and 2007, respectively, and received his Ph.D. degree in Information Systems Engineering from Osaka University, Osaka, Japan, in 2013. He is currently an assistant professor in the Department of Information Systems Engineering, Osaka University. His research interests include computer vision, machine learning, and system level design. He is a member of IEEE and IPSJ.

Takao Onoye received B.E. and M.E. degrees in Electronic Engineering, and Dr.Eng. degree in Information Systems Engineering all from Osaka University, Japan, in 1991, 1993, and 1997, respectively. He is currently a professor in the Department of Information Systems Engineering, Osaka University. His research interests include media-centric low-power architecture and its SoC implementation. He is a member of IEEE, IPSJ, and ITE-J.

Masanori Hashimoto received the B.E., M.E. and Ph.D. degrees in Communications and Computer Engineering from Kyoto University, Kyoto, Japan, in 1997, 1999, and 2001, respectively. Since 2016, he has been a Professor in Department of Information Systems Engineering, Graduate School of Information Science and Technology, Osaka University. His research interest includes computer-aided design for digital integrated circuits, and high speed circuit design. Dr. Hashimoto served on the technical program committees for international conferences including DAC, ICCAD, ITC, Symposium on VLSI Circuits, ASP-DAC, DATE, ISPD and ICCD. He is a member of IEEE, ACM, and IPSJ.