Article Information

Title	An FPGA Implementation of the Two-Dimensional FDTD Method and Its Performance Comparison with GPGPU
Authors	Ryota Takasu, Yoichi Tomioka, Yutaro Ishigaki, Ning Li, Tsugumichi Shibata, Mamoru Nakanishi, Hitoshi Kitazawa
Citation	FDTD method and its performance comparison with GPGPU." IEICE Transactions on Electronics 97.7 (2014): 697-706.
Copyright	copyright@2014 IEICE
IEICE Transactions Online URL	https://search.ieice.org/

PAPER Special Section on Recent Advances in Simulation Techniques and Their Applications for Electronics

An FPGA Implementation of the Two-Dimensional FDTD Method and Its Performance Comparison with GPGPU

Ryota TAKASU[†a)], ***Nonmember***, **Yoichi TOMIOKA**[†b)], ***Member***, **Yutaro ISHIGAKI**[†c)], **Ning LI**[†d)], ***Nonmembers***, **Tsugimichi SHIBATA**[††e)], ***Senior Member***, **Mamoru NAKANISHI**[††f)], ***Member***, ***and*** **Hitoshi KITAZAWA**[†g)], ***Fellow***

SUMMARY Electromagnetic field analysis is a time-consuming process, and a method involving the use of an FPGA accelerator is one of the attractive ways to accelerate the analysis; the other method involve the use of CPU and GPU. In this paper, we propose an FPGA accelerator dedicated for a two-dimensional finite-difference time-domain (FDTD) method. This accelerator is based on a two-dimensional single instruction multiple data (SIMD) array architecture. Each processing element (PE) is composed of a six-stage pipeline that is optimized for the FDTD method. Moreover, driving signal generation and impedance termination are also implemented in the hardware. We demonstrate that our accelerator is 11 times faster than existing FPGA accelerators and 9 times faster than parallel computing on the NVIDIA Tesla C2075. As an application of the high-speed FDTD accelerator, the design optimization of a waveguide is shown.

key words: *FDTD method, FPGA, parallel processing, SIMD array*

1. Introduction

The finite-difference time-domain (FDTD) method is widely used for electromagnetic field analysis of waveguides, antennas, and so on. However, analysis using the FDTD method is a time-consuming process. For example, in [1], the authors mention that it takes from a few hours to several days for simulation using the FDTD method. To reduce the simulation time significantly, a hardware accelerator is essential. Owing to the rapid advancements in field programmable gate arrays (FPGAs), FPGAs play an important role in the speeding up of numerical analysis, as do multicore CPUs and general-purpose computing on graphics processing units (GPGPU) [2]. However, a computing circuit should be designed to match the calculation objective and to fit the target FPGA device in order to achieve a high-performance FPGA accelerator.

Several FPGA accelerators for the FDTD method have been proposed in [1], [3]–[9]. An accelerator for the FDTD method was earlier proposed in [1]. This accelerator mainly focuses on the one-dimensional FDTD method, and it is composed of pipelined bit-serial circuits. However, this architecture is not suitable for recent FPGAs because many digital signal pocessor (DSP) [10] modules are available for highly parallel computation. The accelerator proposed in [3] is for the three-dimensional FDTD method. In this accelerator, computing units are not pipelined to update the electric and magnetic fields, and the performance of this method is not sufficient. On the other hand, an accelerator using a pipelined adder/subtracter and multiplier for updating the electric and magnetic fields has been proposed in [4]. This accelerator is 24 times faster than a 3.0 GHz CPU. However, it is difficult to expand the number of pipelines because of memory access contention. Thus, this accelerator is not suitable for highly parallel computing using recent high-end FPGAs.

An FPGA accelerator based on a two-dimensional single instruction multiple data (SIMD) [10] array architecture has been proposed in [6]–[8]. This accelerator is designed to accelerate general numerical integration methods such as the FDTD method, Red-Black-SOR, and fractional-step method. However, this architecture is not optimized for FDTD computation. Since an FPGA is reconfigurable, the architecture of the accelerator should be optimized for a specific purpose to obtain the highest performance. Recently, the scalable computation method has been proposed [9]. It can be applied to large scale problems by applying a spatio-temporal pipeline technique. However, structure data, driving signal generation, and impedance termination were not considered.

In this paper, we propose a SIMD based two-dimensional array architecture that is optimized for highly parallel computations of the FDTD method, for the following three reasons. First, in the FDTD algorithm, most calculations are performed between two adjacent nodes. Such calculations can be efficiently executed in parallel on a two-dimensional array architecture in which four adjacent processing elements (PEs) are always directly connected. Second, since there is a wealth of hardware resources such as lookup tables (LUTs) [11], registers, and DSPs for recent FPGAs, highly parallel computations using many PEs can be realized on a single FPGA. Third, driving (excitation) signal generation and impedance termination are also implemented in the proposed hardware. Many existing accelerators do not pay special attention to driving signal generation and impedance termination. However, these processes

Manuscript received September 27, 2013.
Manuscript revised January 28, 2014.

[†]The authors are with the Tokyo University of Agriculture and Technology, 2-24-16 Naka-cho, Koganei, Tokyo.
[††]The authors are with the NTT Microsystem Integration Laboratories, 3-1 Morinosato Wakamiya, Atsugi, Kanagawa.
a) E-mail: 50013645214@st.tuat.ac.jp
b) E-mail: ytomioka@cc.tuat.ac.jp
c) E-mail: 50012257502@st.tuat.ac.jp
d) E-mail: 50010257096@st.tuat.ac.jp
e) E-mail: shibata.tsugumichi@lab.ntt.co.jp
f) E-mail: nakanishi.mamoru@lab.ntt.co.jp
g) E-mail: kitazawa@cc.tuat.ac.jp
DOI: 10.1587/transele.E97.C.697

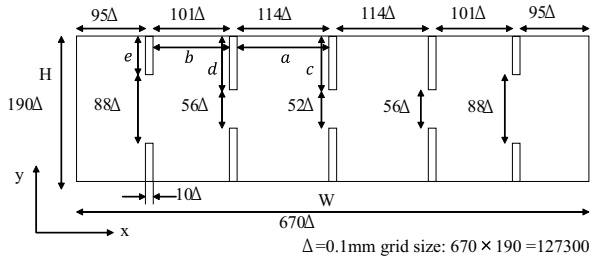

Fig. 1 Canonical problem of waveguide analysis given in [20].

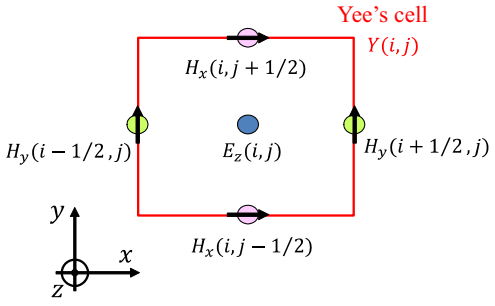

Fig. 2 Two dimensional Yee's cell.

are difficult to execute parallely, and they reduce the overall performance heavily. We show that our accelerator provides superior performance compared to existing FPGA accelerators [4], [6]–[9].

Recently, GPGPU has attracted attention for accelerating numerical analysis, and GPGPU accelerators have been proposed in [12]–[19]. We also compare our accelerator with GPGPU. In [18] and [19], it is reported that two-dimensional FDTD computation on the Tesla C1060 is about 20 times faster than that on a 2.26 GHz CPU. We implemented a two-dimensional FDTD accelerator with Tesla C2075, and we demonstrate that our accelerator implemented with Altera Stratix V (5SGSMD5K2F40C2N) is nearly 9 times faster than parallel computing on the Tesla C2075.

This paper is organized as follows. In Section 2, Maxwell's differential equations and impedance termination equations are explained, and they are rearranged for applicability to parallel processing. In Section 3, the hardware structure and the control method are explained. In Section 4, we compare the accuracy of fixed-point and floating-point calculations to decide the specification of the computing units. In Section 5, we explain our implementation and compare the performance with those of existing methods. In Section 6, we apply our accelerator to the optimization and sensitivity analyses of a waveguide filter structure. The paper is concluded in Section 7.

2. Equations to be calculated in the FDTD method

In this paper, we focus on analysis for metal rectangular waveguides. In this section, we explain Maxwell's differential equations and impedance termination for waveguides. An example of a bandpass filter realized by iris-coupled waveguide cavities [20] is shown in Fig. 1.

2.1 Maxwell's differential equations for waveguides

An analytical region is discretized on a uniform grid, called Yee's cell, as shown in Fig. 2. For each Yee's cell, one electric node is located at the center, and four magnetic nodes surround it.

Based on Yee's cell, the FDTD equations of the H-plane (TE_{n0} modes) field are as follows:

$$H_x^{n+1/2}(i, j+1/2) = H_x^{n-1/2}(i, j+1/2) - \frac{\Delta t}{\mu_x \mu_0 \Delta y}\left\{E_z^n(i, j+1) - E_z^n(i, j)\right\}, \quad (1)$$

$$H_y^{n+1/2}(i+1/2, j) = H_y^{n-1/2}(i+1/2, j) + \frac{\Delta t}{\mu_y \mu_0 \Delta x}\left\{E_z^n(i+1, j) - E_z^n(i, j)\right\}, \quad (2)$$

$$E_z^n(i, j) = E_z^{n-1}(i, j) + \frac{\Delta t}{\epsilon_z \epsilon_0 \Delta x}\left\{H_y^{n-1/2}(i+1/2, j) - H_y^{n-1/2}(i-1/2, j)\right\} - \frac{\Delta t}{\epsilon_z \epsilon_0 \Delta y}\left\{H_x^{n-1/2}(i, j+1/2) - H_x^{n-1/2}(i, j-1/2)\right\}, \quad (3)$$

where H_x is the x-component of the magnetic field, H_y is the y-component of the magnetic field, E_z is the z-component of the electric field, μ_x and μ_y are the relative permeabilities, μ_0 is the vacuum permeability, ϵ_z is the relative permittivity, ϵ_0 is the vacuum permittivity, Δx and Δy are the sizes of the Yee's cell, and Δt is the time interval. The superscripts of the E and H components denote discrete points in time. The electric and magnetic fields are alternately updated by using these equations. For ease of explanation of the hardware implementation, we represent one electric and two magnetic field components by the three one-dimensional arrays $E_z[p]$, $H_x[p]$, and $H_y[p]$. Let w be the number of columns in the two-dimensional array of Yee's cells. Formally, $E_z(i, j)$, $H_x(i, j+1/2)$, and $H_y(i+1/2, j)$ in Yee's cell $Y(i, j)$ are denoted by $E_z[i+jw]$, $H_x[i+jw]$, and $H_y[i+jw]$, respectively. By representing the coefficient of each term by k_l ($l = 1, \cdots, 4$), the above three equations can be replaced by the following four pseudo codes (4)–(7):

$$H_x : H_x[p] \leftarrow H_x[p] + k_1 \times \{E_z[p] - E_z[p+w]\}, \quad (4)$$

$$H_y : H_y[p] \leftarrow H_y[p] - k_2 \times \{E_z[p] - E_z[p+1]\}, \quad (5)$$

$$E_{z1} : E_z[p] \leftarrow E_z[p] - k_3 \times \{H_x[p] - H_x[p-w]\}, \quad (6)$$

$$E_{z2} : E_z[p] \leftarrow E_z[p] + k_4 \times \{H_y[p] - H_y[p-1]\}, \quad (7)$$

$$k_1 = \frac{\Delta t}{\mu_x \mu_0 \Delta y}, \quad (8)$$

$$k_2 = \frac{\Delta t}{\mu_y \mu_0 \Delta x}, \quad (9)$$

$$k_3 = \frac{\Delta t}{\epsilon_z \epsilon_0 \Delta y}, \quad (10)$$

$$k_4 = \frac{\Delta t}{\epsilon_z \epsilon_0 \Delta x}, \tag{11}$$

where p is an array index calculated by $p = i + jw$, and "$\leftarrow$" denotes the update of an array variable. Note that Eq. (3) is decomposed into pseudo codes (6) and (7). Pseudo codes (4)–(7) can be calculated by the same operation pipeline represented as $d = c \pm k \times (a \pm b)$.

2.2 Impedance termination

For the boundary conditions of waveguide ports, we employ the impedance boundary condition proposed in [21]. For example, the impedance boundary condition for the left boundary of a waveguide is represented by

$$\begin{aligned} E_z^n(0, j) = & \frac{1 - Z_0 G/2}{1 + Z_0 G/2} E_z^{n-1}(0, j) \\ & + \frac{Z_0}{1 + Z_0 G/2} \Big\{ 2H_y^{n-1/2}(1/2, j) \\ & - H_x^{n-1/2}(0, j + 1/2) + H_x^{n-1/2}(0, j - 1/2) \\ & - 2GV_s^{n-1/2}(t) e(0, j) \Big\}, \end{aligned} \tag{12}$$

where Z_0 is the characteristic impedance of free space calculated by $\sqrt{\mu_0/\epsilon_0}$, G is an arbitrary terminating impedance of a waveguide, $V_s^{n-1/2}$ is an applied voltage, and e is an eigen function of a mode. We also simplified Eq. (12) using one-dimensional arrays for the hardware implementation. In order to use the same operation pipeline as pseudo codes (4)–(7), we decomposed Eq. (12) into the following four pseudo codes (13)–(16):

$$X_1 : X[p] \leftarrow 0 + k_6 \times \{H_y[p] + H_y[p] \}, \tag{13}$$
$$X_2 : X[p] \leftarrow X[p] - k_6 \times \{H_x[p] - H_x[p - w]\}, \tag{14}$$
$$X_3 : X[p] \leftarrow X[p] + S[j] \times \{ 0 + V'_k[t] \}, \tag{15}$$
$$X_4 : E_z[p] \leftarrow X[p] + k_5 \times \{ 0 + E_z[p] \}, \tag{16}$$
$$k_5 = \frac{1 - Z_0 G/2}{1 + Z_0 G/2}, \tag{17}$$
$$k_6 = \frac{Z_0}{1 + Z_0 G/2}, \tag{18}$$

where $p = 0 + jw$, $X[p]$ is a temporary array variable, $V_k[t]$ is a one-dimensional array representing $2GV_s^{n-1/2}(t)$, $V'_k[t]$ is $-k_6 \times V_k[t]$ and $S[j]$ is a one-dimensional array representing $e(0, j)$. In our accelerator, for each PE, we employ a pipelined module with two additions/subtractions and one multiplication. In the next section, we explain the proposed FDTD accelerator for analyzing waveguides.

3. Hardware architecture

3.1 Parallel and pipeline processing

Parallel processing on two-dimensional arrays is suitable for the FDTD method, because most operations of the FDTD method are performed between adjacent nodes. The proposed hardware consists of a two-dimensional array processor with $R \times C$ processing elements (PEs), a control processor, and data transfer modules, as shown in Fig. 3. Each of the PEs calculates the electric fields and magnetic fields for $h \times w$ nodes, as shown in Fig. 4. Therefore, the number of processable nodes is $Rh \times Cw$.

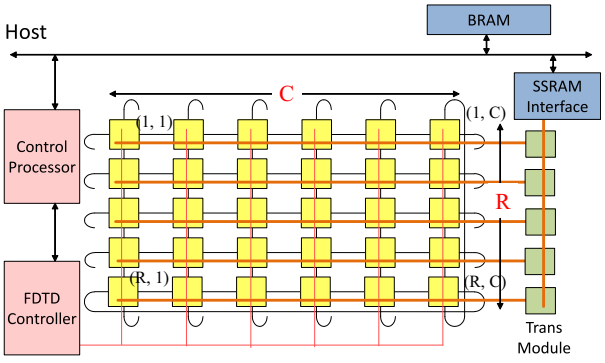

Fig. 3 Two-dimensional array processor.

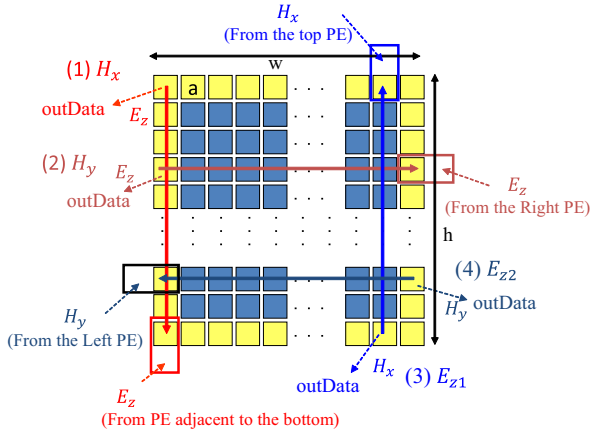

Fig. 4 The node map in a PE. $w \times h$ nodes is processed by the PE, $\longrightarrow$ denotes the processing direction for the designated equations.

Each PE has a pipelined computing unit, two dual-port memories, and two single-port memories as shown in Fig. 5. The computing unit is a six-stage pipeline that is composed of read address set, read, addition/subtraction, multiplication, addition/subtraction, and write-back. It can calculate pseudo codes (4)–(7) and (13)–(16) with one clock throughput. Data E_z are stored in one dual-port memory. Both data H_x and H_y are stored in another dual-port memory. Constant, k_l ($l = 1, \cdots, 6$) and $S[j]$ are stored in the single-port memory called *constMem*. The structure data of the waveguide, such as metal position, driving signal, and termination are stored in another single-port memory called *indexMem*.

The equations for updating the electric and magnetic fields are parallely calculated by the $R \times C$ PEs. On the other hand, the boundary condition equations are calculated by only the leftmost nodes and the rightmost nodes.

In the first step, in each PE, the H_x of the $w \times h$ nodes are calculated as shown in Fig. 4. Then, H_y, E_{z1}, and E_{z2} are calculated in order, and finally, the driving signal generation

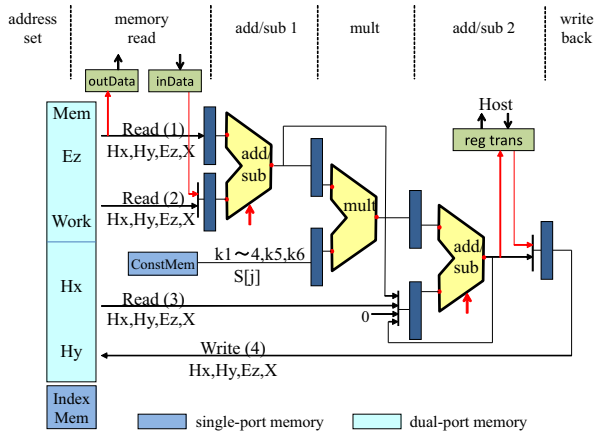

Fig. 5 Structure of a processing element (PE).

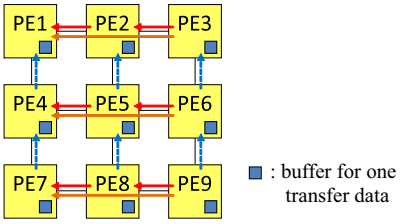

Fig. 6 Synchronous shift data transfer.

and impedance termination are calculated.

3.2 Data transfer between PEs

Each of the PEs needs to receive data from adjacent PEs when operating the nodes of the top, bottom, left, and right edges in the PE. Each PE is connected to the four adjacent PEs for data transfer between the PEs, as shown in Fig. 6. All of the PEs receive data from the same direction simultaneously. With this data transfer method, memory access contention and bus contention can be avoided. Moreover, each PE has a buffer storing one transfer data, and it can also receive data from a distant PE, which is not connected directly, by repeating transfers between adjacent PEs a certain number of times. For example, as shown in Fig. 6, each PE can receive data from two PEs to the right with. With the first left transfer, PE1 receives data from PE2. At the same time, PE2 receives data from PE3. With the second left transfer, PE1 receives data from PE2 that was received from PE3 with the first left transfer. We call this transfer technique *synchronous shift data transfer* [22]. This technique can be useful when a higher order FDTD method [23] is employed.

Each PE must transfer the topmost E_z data to its adjacent PE on the upper side to update the lowermost H_x data as shown in Fig. 4. In order to ensure accurate data transfer even between multiple FPGAs where considerable clock skew is unavoidable, we apply a double buffer structure where two clocks are required for one data transfer. Therefore, some waiting operations may be required for transfer between the PEs. The proposed hardware avoids wait

Table 1 The number of required clock cycles for one time step in each PE and transfer directions.

	# Clock cycles	Wait	Transfer direction	# Transfer cycles
H_x	hw	$l-2$	U	w
H_y	hw	$l-2$	R	h
E_{z1}	hw	$l-2$	D	w
E_{z2}	hw	$l-2$	L	h
X_{1p}	h	$l-2$	—	0
X_{2p}	h	$l-2$	D	1
X_{3p}	h	$l-2$	—	0
X_{4p}	h	$l-2$	—	0
X_{1t}	h	$l-2$	L	1
X_{2t}	h	$l-2$	D	1
X_{4t}	h	$l-2$	—	0
other[1]	3h	$3l-6$	—	0

[1] E_z integration at the driving and terminal nodes and data copy to avoid memory contention.

operations by changing the processing direction in each of the pseudo codes (4)–(7), as shown in Fig. 4. For example, when the H_x of the top node is updated, E_z data read from memory are also stored in an output buffer (outData in Fig. 5). Until calculation of the bottom node H_x is begun, the E_z data in outData of the lower PE are transferred to the input buffer (inData in Fig. 5). The bottom node operation uses the E_z data in inData. Thus, no wait operation is necessary. Similarly, the H_y, E_{z1} and E_{z2} operations are performed.

3.3 Control method of PEs

The pipeline in each PE is controlled by very long instruction word (VLIW) [10] type operation codes issued by the FDTD controller in Fig. 3. The width of the instruction codes is 80 bits of which 40 bits are assigned for the memory addresses of 4 ports, and the other 40 bits are assigned for selection signal of pseudo codes (4)–(7) and (13)–(16), transfer directions, write enables, and selectors. The instructions are determined by the array size, the number of nodes, and the structure data of an analysis target. Similarly, the initial memory data of the indexMem are decided by the structure data. The control code and the initial memory data are generated by software in the host PC and transferred to a memory in the control processor and the *indexMem* in each PE.

By using *synchronous shift data transfer* and the pipeline shown in Fig. 5, the throughput of pseudo codes (4)–(7) and (13)–(16) is 1 node per clock. When the pipeline proceeds to calculate the next equation, e.g., when it proceeds from H_y to E_{z1}, it must wait until calculations for the previous equation have finished. This is attributed to contention in the dual-port memory and computing units. In the structure of Fig. 5, the latency l of the pipeline is 6 clocks including memory-read and write-back stages. The pipeline must wait 4 clocks. The number of required clock cycles for one time step in each PE is shown in Table 1.

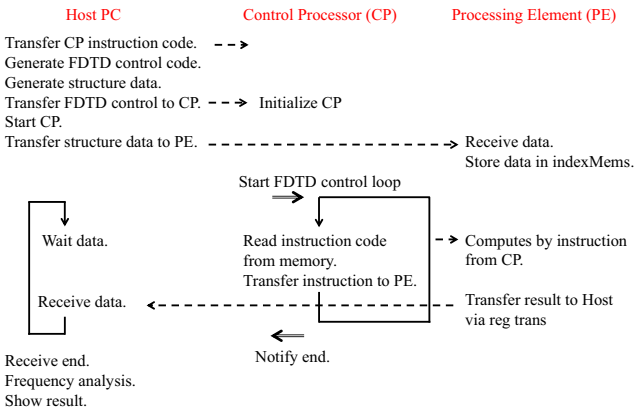

Fig. 7 Flow of control program.

3.4 Control flow of the total system

The program of the control processor is described in an assembler language. The control codes are transferred from the host PC to the FPGA board using the Joint Test Action Group (JTAG) interface which provides data transfer between the FPGA and the host PC for debugging. Moreover, the simulation result is transferred in the opposite direction. In order to control this transfer, we used Altera's system console and tool command language (Tcl) scripts consisting of commands for the system console. The overall processing flow of the system is shown in Fig. 7.

4. Accuracy comparisons of fixed-point and floating-point calculation

In order to implement the FDTD method for the FPGA, we first choose the fixed-point or floating-point calculation method and decide the bit width of the calculation. For this purpose, we compare the accuracy of the fixed-point and floating-point operations.

4.1 Expressions for the fixed-point calculations

Electromagnetic field values need to be quantized for the fixed-point expression with the selected bit-width. In order to decide the position of a decimal point for the fixed-point expression, we investigate the range of variables. The maximum value of $|S[j] \times V_k[t]|$ is 1. It is multiplied by a value of approximately 4, and the range of the electric field E_z is decided by this value. For example, for a 32-bit fixed-point operator, the value of E_z is shifted by 28 bits with a 3-bit margin. The values of H_x and H_y are nearly equal to $1/377$ of E_z, and so these are shifted by 36 bits. Each of $V_k[t]$ and $S[j]$ is similarly shifted according to these shift amounts.

4.2 Comparisons of accuracy

When comparing the accuracy of the fixed-point result with that of the floating-point result, we employed the S matrix of the waveguide. We computed each S matrix of 24, 32, 64-bit floating-point and 24, 32 fixed-point. These results are shown in Figs. 8–10 and Table 2.

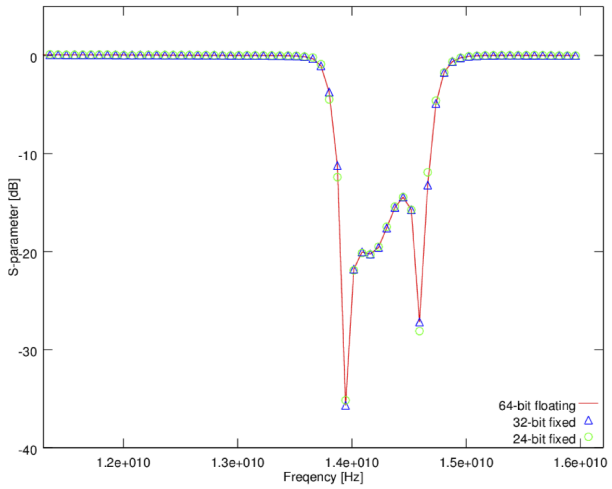

Fig. 8 S_{11} parameters obtained by fixed-point calculations.

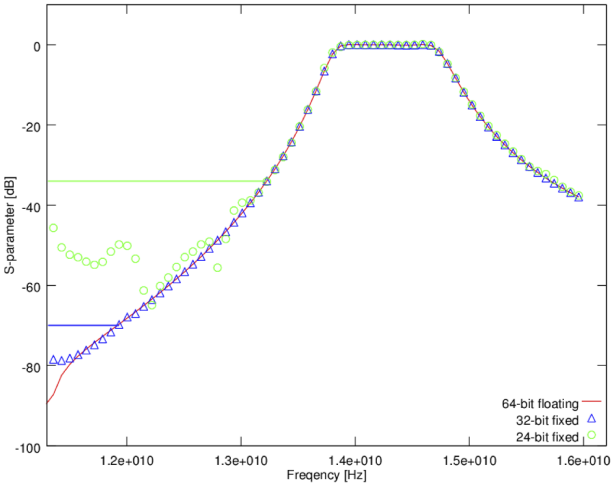

Fig. 9 S_{21} parameters obtained by fixed-point calculations. The green line under −30 dB indicates the accuracy of 24-bit fixed-point calculations. The blue line under −70 dB indicates the accuracy of 32-bit fixed-point calculations.

As shown in Fig. 8, the deviation between 64-bit floating-point and 24, 32-bit fixed-point on S_{11} parameter is very small. However, in Fig. 9, S_{21} parameter of 24-bit fixed-point coincides with that of 64-bit floating-point until −30 dB. This result suggests that the accuracy of 24-bit fixed-point calculations is 30 dB. 32-bit fixed-point coincides with 64-bit floating-point until −70 dB, and the accuracy is 70 dB. On the other hand, as shown in Fig. 10, S_{21} parameter of 32-bit floating-point coincides with that of 64-bit floating-point until the minimum S_{21} value in the graph.

4.3 Selection of the proposed hardware

Obviously, these results show that floating-point calculation is better than fixed-point calculation. However, if we do not need higher than 70 dB accuracy, the result of fixed-point calculation is sufficient. For FPGA accelerators, we must

Table 2 Accuracy comparison of fixed-point and floating-point calculations.

	Fxed-point		Floating-point		
	24-bit	32-bit	24-bit	32-bit	64-bit
Attenuation accuracy	30 dB	70 dB	60 dB	> 90 dB	> 90 dB

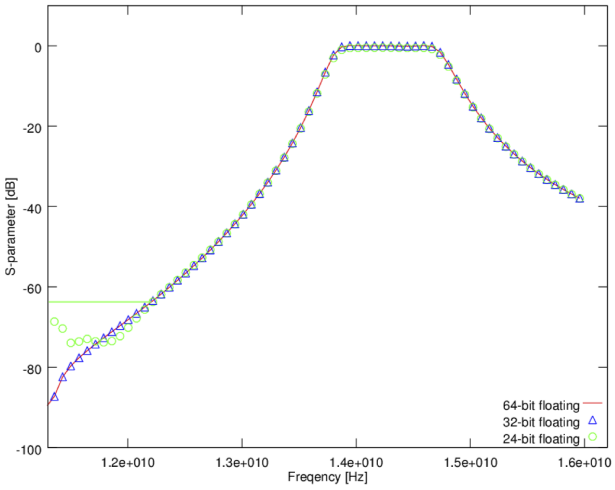

Fig. 10 S_{21} parameters obtained by floating-point calculations. The green line under −60 dB indicates the accuracy of 24-bit floating-point calculations.

consider the resource difference. A floating-point accelerator requires several times more resources than a fixed-point accelerator, and the number of PEs that can be implemented is decreased. We selected 32-bit fixed point, and our proposed hardware is applicable for analysis with 70 dB accuracy. If higher accuracy is required, we need to select 32-bit floating point.

5. Implementation and performance comparisons

5.1 Implementation

The proposed system has been implemented using an Altera Stratix V 5SGSMD5K2F40C2N FPGA. This FPGA has 1,590 DSP blocks, 2,014 block RAMs, and 172 K ALMs.

The analysis target size is 670 × 190, and the number of time steps is 65536 as in [18] and [19]. Thus, R, C, h, and w were set as 24, 20, 8, and 34, respectively, and the number of processable nodes is $680 \times 192 = 130,560$.

Each PE in the proposed accelerator needs 2 DSP blocks, 4 block RAMs, and about 300 ALMs. So, the maximum number of PEs is limited by the number of the block RAMs. Considering that the block RAMs are also used in the control processor and the data transfer buffer, we set the number of PEs as $20 \times 24 = 480$. The resources and the maximum frequency of the hardware are summarized in Table 3. We run the accelerator with a clock frequency of 100 MHz.

The number of clock cycles for a 1 time step simulation can be calculated by

$$\begin{aligned} N &= hw \times 4 + h \times 10 + (l-2) \times 14 \\ &= 272 \times 4 + 8 \times 10 + 4 \times 14 = 1224. \end{aligned} \quad (19)$$

Table 3 Resource utilization for the proposed accelerator.

Memory usage (bit)	32,912,512/41,246,720 (80%)
Dedicated logic registers	120,460/690,400 (17%)
DSP blocks	960/1,590 (60%)
Combinational ALUTs	160,651/172,600 (93%)
Maximum frequency (MHz)	109.9

The processing time T required for 65,536 time steps is

$$T = (1224 \times 65536)/100M = 0.765\text{s}. \quad (20)$$

This value includes the driving signal generation and impedance termination calculation time.

5.2 Comparisons with existing FPGA accelerators

We compare the performance of the proposed accelerator with that of existing FPGA accelerators. We use fields update per second (FUPS) for performance evaluation. It is defined as the number of updating fields per second. The comparison result is shown in Table 4.

The accelerator proposed in [4] does not adopt a two-dimensional array, and we think that the degree of parallelism is not sufficiently utilized. The accelerator in [6]–[8] adopts a two-dimensional array. It aims to analyze not only the FDTD method but also Red-Black-SOR and the fractional-step method. This architecture is not optimized for FDTD computation. The calculation pipeline consists of mult-add/sub and must be applied multiple times to process the FDTD equations. In contrast, the structure of our accelerator is the add/sub-mult-add/sub calculator which processes the basic FDTD formula in 1 clock throughput. In addition, our accelerator supports signal generation and impedance termination. Although the accelerators in [6]–[8] employ 32-bit floating-point calculation and the proposed accelerator employ 32-bit fixed-point calculation, the accuracy is comparable when the required accuracy is within 70 dB.

5.3 Comparisons with GPGPU

We implemented the two-dimensional FDTD calculation explained in Section 2 on Tesla C2075, and we applied it to waveguide analysis shown in Fig. 1. The number of time steps is 65536. In this implementation, we employed 32-bit floating calculation. We used techniques enhancing GPGPU performance, and in Table 5, we summarized the execution time for three implementations.

For the first implementation, we used padding technique, and the analyzed region was expanded to 256 × 768. Note that the actual size of analysis region is 190×670. The processing consists of two stages: (a) H_x, H_y, and E_z fields update denoted by pseudo codes (4)–(7) and (b) driving and

Table 4 Performance comparisons with existing accelerators.

	Number of nodes & calculation type	Fields update per second (FUPS)	
		H_x, H_y, E_z fields update	Driving and termination are included[1]
Chen[4]	100×100 30-bit fixed-point	0.0138G fields/s	—
Sano[6]	72×72 32-bit floating-point	0.970G fields/s	—
Proposed	670×190 32-bit fixed-point	—	$(670 \times 190) \times 65536/0.765 = 10.9G$ fields/s
Tesla C2075	670×190 32-bit floating-point	1.56G fields/s	1.19G fields/s
GTX 480[14]	3072×3072 32-bit floating-point	3.68G fields/s	—
GTX 680[16]	4096×4096 32-bit floating-point	4.27G fields/s	—
(C2075 2D array)[2]	670×190 32-bit floating-point	—	$10.9 \times 1.15G/100M = 125G$ fields/s

[1] Driving and termination calculations denoted by pseudo codes (13)–(16) and E_z integration are included.
[2] The predicted performance of the proposed array structure by using the calculator of the C2075.

termination calculations denoted by pseudo codes (13)–(16) and E_z integration. In the H_x, H_y, and E_z fields update, each thread updates these fields for one node; the grid dimension was set as $1 \times 168 \times 1$, and the block dimension was set as $192 \times 4 \times 1$. In driving and termination calculations, each thread updates the E_z field for one node; the grid dimension was set as $6 \times 1 \times 1$, and the block dimension was set as $32 \times 1 \times 1$. Moreover, E_z integration is calculated by one thread for driving and termination of the waveguide.

For the second implementation, we used shared memories to reduce the number of global memory accesses. In the H_x, H_y, and E_z fields update, these fields of 194×6 nodes are stored in the shared memory for each block. In driving and termination calculations, the H_y fields of driving and termination nodes are stored in the shared memory. Furthermore, the E_z fields of driving and termination nodes are stored in the shared memory, and we calculate E_z integration by using the reduction techniques with $128 \times 1 \times 1$ threads. We used "#pragma unroll" for all "for" sentences.

For the third implementation, we changed the H_x and H_y fields updates from the second implementation. Each thread updates the H_x and H_y fields of eight nodes on the consecutive rows, and the H_y fields of one row are stored in the shared memory. Moreover, each thread stores the H_x field of the node of the previous row in a register. The grid dimension was set as $1 \times 21 \times 1$, and the block dimension was $192 \times 4 \times 1$.

It takes 6.65 s to update the E_z, H_x, H_y fields with pseudo codes (4)–(7) for the first implementation. The time is improved to 5.61 s. Moreover, the execution time is improved to 5.35 s, which is 1.56G fields/s for the third implementation. However, when driving, termination, and E_z integration are included, the total time is 7.00 s, which is 1.19G fields/s. We list the best result in Table 4.

Our proposed accelerator is about 9 times faster than the accelerator using C2075 when driving, termination, and E_z integration are included. Moreover, Kawada *et al.* [14] have reported that NVIDIA GTX480 achieved 3.68G fields/s. From the report of Ito [16], NVIDIA GTX680 achieves 4.27G fields/s. Note that they also employed 32-bit floating calculations, and that they do not handle pseudo codes (13)–(16) and E_z integration. These results are also listed in Table 4. Our proposed accelerator is about 2.5 times faster than the accelerator reported in [16] using GTX680.

The Tesla C2075 contains 448 calculators of 32-bit floating-point and operates at 1.15 GHz, and *the number of calculators* × *operation clocks* is over 500G. On the other hand, the proposed accelerator contains 480 PEs and operates at 100 MHz when it is implemented on Stratix V 5SGSMD5K2F40C2N, and *the number of calculators* × *operation clocks* is 43G, which is about 1/12 compared to the Tesla C2075. However, the execution speed is about 9 times faster than the Tesla C2075. This result is due to the difference in the memory architecture. In GPGPU, as shown in Fig. 11, multilevel memory is shared by many PEs. It can realize high speed transfer between arbitrary PEs when the transferred data is small. However, when all the PEs transfer data to adjacent PEs, memory access contention and bus contention occur and the data transfer speed is reduced. Moreover, data transfer between multilevel caches spend much resources. This memory bottleneck is inevitable for this architecture. On the other hand, in our accelerator, memory access contention and bus contention are avoided because each PE accesses an adjacent PE in the same direction at the same time as shown in Fig. 6. By using this transfer technique and the proposed six-stage pipeline shown in Fig. 5, we can achieve high performance in the proportion of the number of calculators × operation clocks. If we construct the proposed array structure by using the calculator of the C2075 the predicted performance is improved nearly 105 times as shown in the bottom column of Table 4. As GPGPU is being improved rapidly, state-of-the-art GPGPU is more than 3 times faster than C2075 in peek speed. Even if it takes this into consideration, our proposed accelerator is

Table 5 Execution time for analyzing the waveguide shown in Fig. 1 on Nvidia Tesla C2075

Implementation	Use of techniques					Execution time [s]		
	P[1]	S[2]	M[3]	R[4]	U[5]	H_x, H_y, E_z fields update	Driving, termination, and E_z integration	Total
1st	YES	NO	NO	NO	NO	6.65	3.55	10.20
2nd	YES	YES	NO	YES	YES	5.61	1.65	7.26
3rd	YES	YES	YES	YES	YES	5.35	1.65	7.00

[1] P: Padding technique
[2] S: Shared memory
[3] M: Calculating multiple nodes by one thread
[4] R: Reduction techniques
[5] U: #pragma unroll

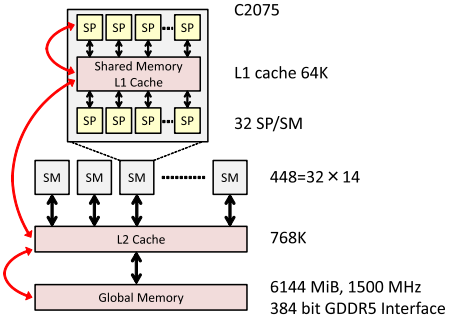

Fig. 11 The memory structure and data transfer in GPGPU.

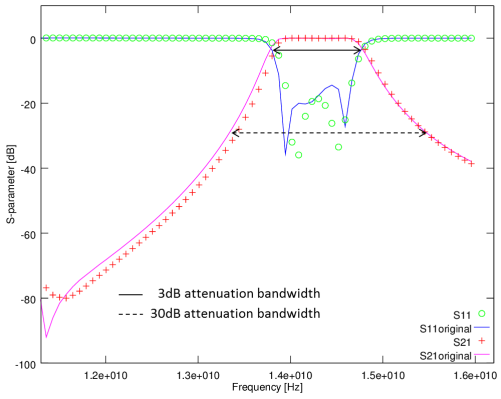

Fig. 12 Comparison of S parameters between the obtained structure and the original structure.

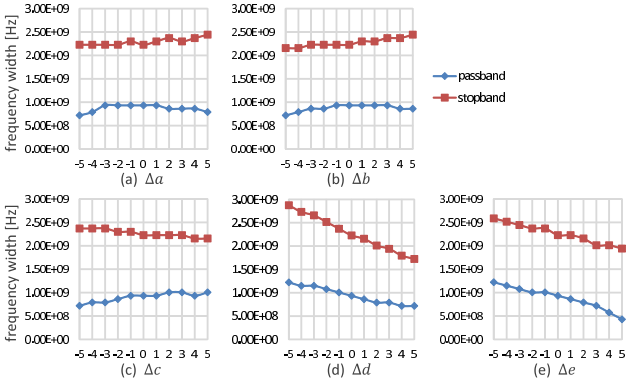

Fig. 13 The passband and stopband sensitivity for variations of each parameter.

nearly 3 times faster than GPGPU. Precise comparison with NVIDIA Tesla K20 is a future work.

6. Applications of the proposed hardware

6.1 Application to the optimization of the canonical waveguide problem

As an application of the proposed FDTD accelerator, we consider the problem of optimizing the position and length of waveguide irises. We change the length and the position of irises denoted by $a \sim e$ in Fig. 1 in the range of a ± 1 grid. The length and position of other irises are determined so as to maintain left-and-right symmetry and up-and-down symmetry. The number of all the structure combinations is $3^5 = 243$. For each structure, the electromagnetic fields and frequency characteristic are analyzed through 65536 time steps. The structure data are generated by the host PC and transferred to a memory in the FPGA board. The result analyzed in the FPGA board is returned to the host PC, and frequency analysis is performed. These processes are executed for each structure.

The result is evaluated by the passband width, the maximum ripple in the passband, and the 30 dB attenuation bandwidth. The best result is shown in Fig. 12, which is obtained when $\Delta a = \Delta b = \Delta d = 0$ and $\Delta c = \Delta e = +1$. The low-frequency side of the passband is steeper than that of the canonical waveguide. We evaluate the execution time, excluding the data transfer time between the host and the FPGA board because the implemented JTAG interface is very slow. It is possible to simultaneously execute data transfer and FDTD calculations in the FPGA board if the high-speed PCI Express interface is applied. So, the execution time can be calculated by

$$\begin{aligned} T &= (N_{init} + N_{analysis} \times 65536) \times 243/100M \\ &= (392496 + 1224 \times 65536) \times 243/100M \\ &= 186.8\text{s}. \end{aligned} \tag{21}$$

6.2 Application to the sensitivity verification of waveguide

The proposed accelerator is also useful for evaluating a characteristic variation arising from the structure variation of waveguides. When each of the parameters of the waveguide shown in Fig. 1 is changed in the range of ± 5 grid, the passband and stopband width variations are shown in Fig. 13. We can see that parameters d and e are especially sensitive.

7. Conclusions

An FPGA accelerator dedicated for the two-dimensional FDTD method is presented. The proposed accelerator implemented with Altera Stratix V (5SGSMD5K2F40C2N) is 11 times faster than existing FPGA accelerators and 9 times faster than Tesla C2075. It has an accuracy of about 70 dB on S_{21} parameters and driving signal generation and impedance termination are also implemented. The applications to the optimization of the canonical waveguide problem and the sensitivity verification of waveguide were shown. Although the applicable size is not large at present, with the rapid progress of the FPGA technology, a 1M node can be processed by a next generation FPGA. If we use multiple FPGAs, the allowable size can be further extended. This method is a good candidate for use as a high-performance computing (HPC) method in the near future.

References

[1] R. N. Schneider, L. E. Turner, and M. M. Okoniewski, "Application of FPGA technology to accelerate the finite-difference time-domain (FDTD) method," Proceedings of the 2002 ACM/SIGDA tenth international symposium on field-programmable gate arrays, FPGA '02, New York, NY, USA, pp. 97–105, ACM, Feb. 2002.
[2] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," Queue, vol. 6, no. 2, pp. 40–53, Mar./Apr. 2008.
[3] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, M. S. Mirotznik, and D. W. Prather, "Hardware implementation of a three-dimensional finite-difference time-domain algorithm," Antennas and Wireless Propagation Letters, IEEE, vol. 2, no. 1, pp. 54–57, 2003.
[4] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport, "An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm," Proceedings of the 2004 ACM/SIGDA 12th international symposium on field programmable gate arrays, FPGA '04, New York, NY, USA, pp. 213–222, ACM, 2004.
[5] S. Endo, J. Sonoda, and M. Sato, "Acceleration of FDTD method with time and space pipeline in FPGA implementation," The Transactions of the Institute of Electronics, Information and Communication Engineers. B, vol. 92, no. 1, pp. 243–249, Jan. 2009. (in Japanese).
[6] K. Sano, L. Wang, and S. Yamamoto, "Prototype implementation of array-processor extensible over multiple FPGAs for scalable stencil computation," SIGARCH Computer Architecture News, vol. 38, no. 4, pp. 80–86, Jan. 2011.
[7] K. Sano, W. Luzhou, Y. Hatsuda, T. Iizuka, and S. Yamamoto, "FPGA-array with bandwidth-reduction mechanism for scalable and power-efficient numerical simulations based on finite difference methods," ACM Trans. Reconfigurable Technol. Syst., vol. 3, no. 4, pp. 21:1–21:35, Nov. 2010.
[8] K. Sano, S. Yamamoto, and Y. Hatsuda, "Domain-specific programmable design of scalable streaming-array for power-efficient stencil computation," SIGARCH Comput. Archit. News, vol. 39, no. 4, pp. 44–49, Dec. 2011.
[9] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-FPGA accelerator for scalable stencil computation with constant memory-bandwidth," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, pp. 695–705, Mar. 2014.
[10] J. Eyre, "The digital signal processor derby," Spectrum, IEEE, vol. 38, no. 6, pp. 62–68, June 2001.
[11] K. Compton, S. Hauck, and K. Compton, "An introduction to reconfigurable computing," IEEE Computer, 2000.
[12] N. Takada, T. Shimobaba, N. Masuda, and T. Ito, "High-speed FDTD simulation algorithm for GPU with compute unified device architecture," Antennas and Propagation Society International Symposium, 2009. APSURSI '09, Charleston SC, USA, pp. 1–4, IEEE, June 2009.
[13] P. Sypek, A. Dziekonski, and M. Mrozowski, "How to render FDTD computations more effective using a graphics accelerator," IEEE Transactions on Magnetics, vol. 45, no. 3, pp. 1324–1327, Mar. 2009.
[14] N. Kawada, K. Okubo, and T. Tsuchiya, "A performance comparison of numerical analysis of electromagnetic field in time domain using graphics processing unit (GPU) parallel computing," IEICE, vol. J94-B, no. 3, pp. 480–483, Mar. 2011. (in Japanese).
[15] N. Kawada, K. Okubo, N. Tagawa, and T. Tsuchiya, "Multi-GPU numerical simulation of electromagnetic field with high-speed visualization using CUDA and OpenGL," B-Abstracts of IEICE TRANSACTIONS on Communications, vol. J95-B, no. 2, pp. 375–380, Feb. 2012. (in Japanese).
[16] T. Ito, Introduction to GPU programming, Kodansya, Tokyo, 2013. (in Japanese).
[17] N. Morita, "Acceleration of an FDTD solver for analyzing MMIC passive element circuit characteristics by using GPGPU," IEICE, vol. J96-C, no. 6, pp. 94–102, June 2013. (in Japanese).
[18] S. Hattori, A. Onozawa, and T. Shibata, "C-15-12 FDTD simulation of a waveguide canonical problem using GPGPU," Proceedings of the IEICE General Conference, vol. 2012, no. 1, p. 319, Mar. 2012. (in Japanese).
[19] http://www.brl.ntt.co.jp/event/scienceplaza/poster/no_40.pdf
[20] http://www.ieice.org/es/est/activities/kihan_mst/01/main.html
[21] T. Shibata and T. Itoh, "Generalized-scattering-matrix modeling of waveguide circuits using FDTD field simulations," IEEE Transactions on Microwave Theory and Techniques, vol. 46, no. 11, pp. 1742–1751, Nov. 1998.
[22] Y. Tomioka, R. Takasu, T. Aoki, E. Hosoya, and H. Kitazawa, "FPGA implementation of exclusive block matching for robust moving object extraction and tracking," IEICE Transaction on Information and Systems, vol. E97-D, no. 3, pp. 573–582, Mar. 2014.
[23] C. M. Jr., S. L. Broschat, and J. B. Schneider, "Higher-order FDTD methods for large problems," J. Applied Computational Electromagnetics Society, vol. 10, pp. 17–29, 1995.

Ryota Takasu received his B.S. degree in Electrical and Electronic Engineering from Tokyo University of Agriculture and Technology, Tokyo, Japan, in 2013. He is currently pursuing a master's course at the university. His research interests include image processing.

Yoichi Tomioka received his B.E., M.E., and D.E. degrees from Tokyo Institute of Technology, Tokyo, Japan, in 2005, 2006, and 2009, respectively. He was a research associate at Tokyo Institute of Technology up to 2009. Since 2009, he has been an assistant professor in the Division of Advanced Electrical and Electronics Engineering at Tokyo University of Agriculture and Technology. His research interests include image processing, security systems with mobile robots, VLSI package design automation, and combinational algorithms. He is a member of IEEE and IPSJ.

Yutaro Ishigaki graduated from Tokyo National College of Technology, Tokyo, Japan, in 2012. He is currently pursuing a bachelor's course at Tokyo University of Agriculture and Technology, Tokyo, Japan. His research interests include microprocessor and image processing. He is a member of IEEE.

Ning Li is currently pursuing a bachelor's course at Tokyo University of Agriculture and Technology, Tokyo, Japan. His research interests include digital circuits and FPGAs.

Tsugimichi Shibata graduated from Tokyo National College of Technology in 1980 and received the B.S., M.S., and Ph. D. degrees in electrical engineering from the University of Tokyo in 1983, 1985, and 1995, respectively. In 1985, he joined NTT, where he had been engaged in research on electromagnetic-field analyses, design of high-speed optical front-end ICs, and protocol-control LSIs for data transmission systems. From 1996 to 1997, he was a Visiting Scholar at the University of California at Los Angeles (UCLA), where he did research on diakoptics in numerical simulations. He is a senior member of the IEEE and IEICE, and the director of NTT Microsystem Integration Laboratories.

Mamoru Nakanishi received the B.E. and M.E. degrees in physical electronics from Tokyo Institute of Technology, Tokyo, Japan, in 1986 and 1988, respectively. In 1988, he joined the LSI Laboratories, Nippon Telegraph and Telephone Corporation (NTT). He was engaged in research on highly parallel processing system and LSI architectures. He is a project manager of the First Promotion Project, NTT Microsystem Integration Laboratories. He is a member of IPSJ and IEEE.

Hitoshi Kitazawa received his B.S., M.S., and Ph.D. degrees in Electronic Engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1974, 1976 and 1979, respectively. He joined the Electrical Communication Laboratories, Nippon Telegraph and Telephone Corporation (NTT), in 1979. Since 2002, he is a professor at Tokyo University of Agriculture and Technology. His research interests include VLSI CAD algorithm, computer graphics and image processing. He is a member of IPSJ and IEEE.