PAPER

3D-DCT Processor and Its FPGA Implementation

Yuki IKEGAKI[†a)], ***Student Member***, **Toshiaki MIYAZAKI**[†], ***and*** **Stanislav G. SEDUKHIN**[†], ***Members***

SUMMARY Conventional array processors randomly access input/coefficient data stored in memory many times during three-dimensional discrete cosine transform (3D-DCT) calculations. This causes a calculation bottleneck. In this paper, a 3D array processor dedicated to 3D-DCT is proposed. The array processor drastically reduces data swapping or replacement during the calculation and thus improves performance. The time complexity of the proposed $N \times N \times N$ array processor is $O(N)$ for an N^3-size input data cube, and that of the 3D-DCT sequential calculation is $O(N^4)$. A specific I/O architecture, throughput-improved architectures, and more scalable architecture are also discussed in terms of practical implementation. Experimental results of implementation on FPGA (field-programmable gate array) suggest that our architecture provides good performance for real-time 3D-DCT calculations.

key words: *3D-DCT, array processors, data manipulation, FPGA*

1. Introduction

Advances in digital imaging applications such as high-definition television [1], teleconference, medical and space exploration images [2], and portable video players have increased demand for effective image compression techniques. Prospective systems such as portable video chatting require high-performance computing to realize video compression because the transmission bandwidth is often limited [3]. 2D-DCT (two-dimensional discrete cosine transform) is used for the spatial information compression of 2D images [4]. Three-dimensional (3D)-DCT extends the DCT energy compaction properties to integral 3D images and the spatio-temporal coding of 2D video sequences [4], [5]. 3D-DCT video compression by using quantized coefficients produces high compression ratios and good visual quality of the reconstructed video frames [6]–[8]. In addition, image data can be compressed by a factor of 10 with 3D-DCT without reducing the image quality by scan ordering [9]. In this paper, we discuss array processors that perform 3D-DCT effectively. A block diagram of the 3D-DCT/IDCT (Inverse DCT) video codec is shown in Fig. 1.

As compared to motion-estimation/compensation-based methods, the 3D-DCT approach offers three advantages for portable video compression systems.

1. No motion estimation is needed; hence, the number of en/decoding operations per pixel is much less than in a motion-estimation/compensation-based method [10].
2. The encoder and decoder have equivalent architecture; however, coefficient data stored in memory are different when they perform encoding or decoding.
3. There is no relationship between the complexity of implementation and the compression ratio [11].

The sequential calculation of 3D-DCT is time-consuming because coefficients and input data stored in memory have to be accessed randomly many times; this makes it difficult to implement a real-time application. Thus, researchers have proposed many algorithms [12], [13] and hardware architectures [14], [15] for the rapid computing of 3D-DCT, especially for real-time applications. In [16] and [17], parallel algorithms for fast 3D-DCT computation based on butterfly calculation are proposed. The butterfly calculation is carried out in $\log_2 N$ steps for an N^3-size image. However, the wiring cost connecting the 3D butterfly modules becomes high when N is large. In addition, these algorithms and hardware architectures require complex data alignments. Thus, special data alignment circuits are needed to prepare input data in an appropriate form. Moreover, time-consuming postprocesses such as recursive additions must be performed after the butterfly calculation [17]. The well established traditional design and complexity analysis of the systolic algorithms [18] usually ignore the I/O processes, but they often take a long time to maintain a 2D data alignment. Furthermore, hardware realization of the 2D data alignment is relatively difficult. Nowadays, the advances in the 3D VLSI technology [19], stacked memory [20], [21], and Giga-size sensor arrays [22] with a parallel read-out and embedded logic allow realization of massively parallel array processors with one I/O channel per PE (Processing Element) [23]. These technologies are able to achieve the multidimensional data streams [24] and solve the current I/O bottleneck problems on many core architectures. However, to show practi-

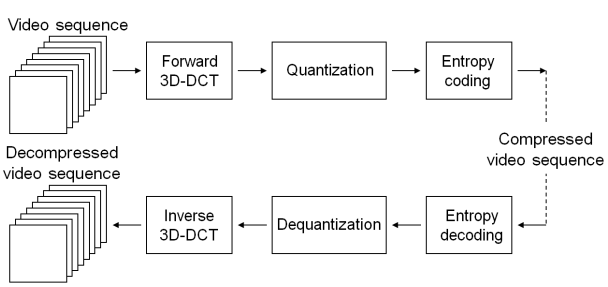

Fig. 1 Block diagram of the 3D-DCT/IDCT video codec.

Manuscript received January 7, 2011.
Manuscript revised March 20, 2011.

[†]The authors are with the University of Aizu, Aizu-wakamatsu-shi, 965–8580 Japan.
a) E-mail: m5131112@u-aizu.ac.jp

DOI: 10.1587/transinf.E94.D.1409

cal implementation results, in this paper, the proposed array processors are implemented with their standard I/O on an FPGA board.

In this research, practical array processors dedicated to 3D-DCT are proposed. Our array processor has N^3 PEs connected in the form of a 3D toroidal cube. Each PE has the same structure: MAC (multiply and accumulate) units, a register file, and wires connected to adjacent PEs. Compared to other processors [4], [11], [14], [15], our array processor can input and output data without any preprocessing or postprocessing (e.g., time-space skewing) and drastically reduce data swapping or replacement tasks during 3D-DCT calculation by introducing a smart data transfer scheme with a simple PE array structure. This is the key to realize high-speed calculation of 3D-DCT.

The rest of this paper is organized as follows. Section 2 presents the definition of 3D-DCT. In Sects. 3 and 4, the original array processor architecture and its I/O interfaces for 2D devices are proposed. Throughput-improved or scalable architectures are presented in Sects. 5 and 6. Experimental results and evaluations are given in Sect. 7. Conclusions are discussed in Sect. 8.

2. Definition Formula of 3D-DCT

Let $X_{N\times N\times N} = [X(i, j, k)]$, $0 \le i, j, k \le N-1$, be an input data cube. The 3D type-II discrete cosine transform of $X_{N\times N\times N}$ is defined as follows [4]:

$$Y_{N\times N\times N} = [Y(s, r, p)], 0 \le s, r, p \le N-1,$$

where

$$Y(s,r,p) = \Phi \cdot \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} X(i,j,k) \times C(k,p)\times C(j,r)\times C(i,s),$$

$$\Phi = \sqrt{\frac{8}{N^3}}\cdot \epsilon(s)\cdot\epsilon(r)\cdot\epsilon(p),$$

$$\epsilon(m) = \begin{cases} \frac{1}{\sqrt{2}}, for\ \ m=0; \\ 1, \quad for\ \ m>0; \end{cases} \quad m \in \{s,r,p\},$$

and elements of the coefficient matrix $C_{N\times N} = [C(u, v)]$ are pre-computed as

$$C(u,0) = \frac{1}{\sqrt{2}}, \quad C(u,v) = \cos\left(\frac{\pi(2u+1)v}{2N}\right),$$
$$u \in \{i,j,k\}, (u,v) \in \{(i,s),(j,r),(k,p)\}.$$

The 3D-DCT is a linear transformation that converts an original 3D coordinate system (i, j, k) into a new 3D system, (s, r, p). Coefficient matrix $C_{N\times N}$ is used for converting k-, j-, and i-coordinates into p-, r-, and s-coordinates, respectively. Hereafter, we call the calculation that strictly follows the above formulas as "sequential calculation."

3. Proposed Array Processor Architecture

3.1 Overview

Figure 2 overviews the proposed array processor architecture when N=4. The array processor has a cubic structure. Each PE is basically connected with six adjacent PEs, and the PEs located on the surface of the cube have wrap-around torus connections. In other words, a PE is connected to the PE on the opposite side of the cube.

The structure of a PE used in the proposed array processor is shown in Fig. 3. For N^3-size 3D-DCT, each PE is mainly composed of I/O selectors, a MAC unit, a switch (SW), and a register file. The I/O selectors are implemented by a 3-to-1 multiplexer and a 1-to-3 demultiplexer. In Fig. 3, the input data come from the input ports: $(i+1)\%N$, $(j+1)\%N$, and $(k+1)\%N$. Hereafter, the operator "%" indicates modulo operation. The calculated data go out through the output ports: $(i-1)\%N$, $(j-1)\%N$, and $(k-1)\%N$. Each selector is controlled by a state signal. The switch takes a multiplicand, a multiplier, and an accumulated data from the register file. The size of the register file is $3N+4$; $3N$ registers are for coefficients and four registers are for input data and calculation results.

The data transfer directions are shown in Fig. 4. All PEs have homogeneous structure and perform parts of the 3D-DCT calculation and data transfer systolically. The se-

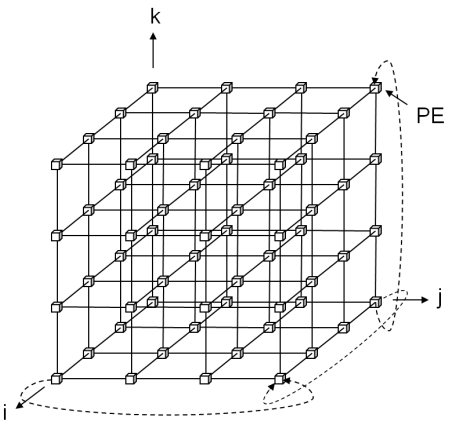

Fig. 2 An overview of the proposed array processor architecture for 4^3-size 3D-DCT (control unit is not shown).

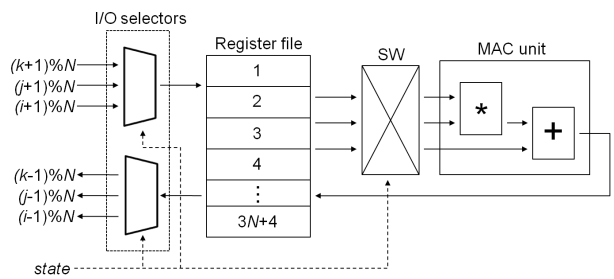

Fig. 3 PE structure of the array processor.

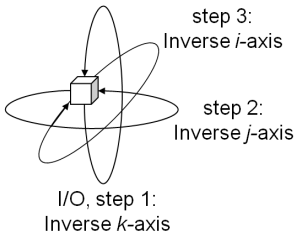

Fig. 4 Directions of the data transfer.

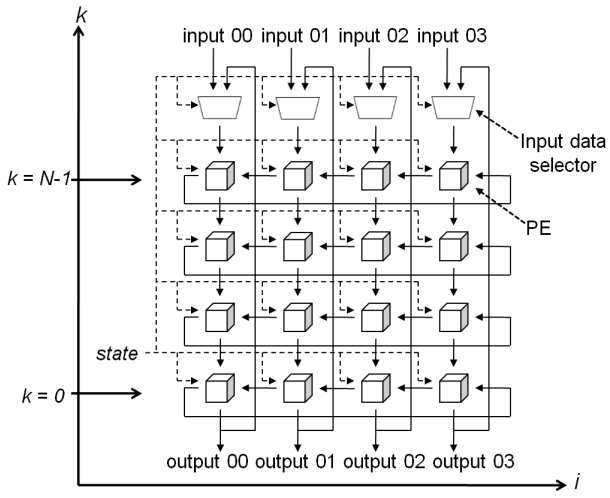

Fig. 5 A sliced view of the *i*-*k* plane.

quence to calculate the result of N^3-size 3D-DCT is data input, steps 1, 2, and 3 for 3D-DCT calculation, and data output. In traditional array processors, the data swapping or replacement during the 3D-DCT calculation is a performance bottleneck because of the high-speed operations and low-speed memory data access. The proposed array processor maintains input data integrity and locality, and there is no data rearrangement during the 3D-DCT calculation. This is a significant feature for improving the performance.

3.2 Data Input/Output

The partial I/O connections of the proposed array processor are shown in Fig. 5. This array processor inputs N^2 data simultaneously. The $N \times N$ input ports are arranged on the $k = N-1$ (i, j)-plane; each PE transfers input data to $(k-1)$-th adjacent PE opposite to k-axis. N *microsteps* are needed to input all N^3 data. The output sequence is almost the same as the input sequence. The $N \times N$ output data of PEs on the $k = 0$ (i, j)-plane are fed from the corresponding PEs on the $k = N-1$ plane using the end-around torus connections. The PEs on the $k = 0$ plane also have output ports, which are used to get the N^3 final results.

3.3 Calculation Procedure of 3D-DCT

As shown in Sect. 2, the 3D-DCT is a heavy task that requests many MAC operations, and it is difficult to realize a hardware circuit that performs the 3D-DCT directly. However, since 3D-DCT is a separable transform [25], we can implement it as the set of three 1D transforms and reduce the hardware cost. In our array processor, there are three steps to obtain the 3D-DCT result. To perform 3D-DCT,

- the coefficients and input data stored in memory have to be accessed many times, and/or
- the partial results need to be rearranged to accelerate memory data access after each 1D transform is finished.

These processes are significant performance bottlenecks for a fast 3D-DCT calculation. By prestoring the 3D-DCT coefficients for each PE and moving the partial calculation results appropriately using the torus connections between PEs, our array processor can completely eliminate the abovementioned data rearrangement in obtaining the 3D-DCT results. This is a key reason why our array processor can perform 3D-DCT effectively.

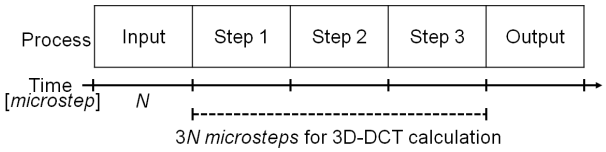

Fig. 6 Time chart to perform a N^3-size 3D-DCT.

In our array processor, actual 3D-DCT is performed in three steps: steps 1, 2, and 3. In step 1, $X(i, j, k)$ is multiplied with $C(k, p)$. In step 2, $\dot{X}(i, j, p)$, results of step 1, is multiplied with $C(j, r)$. In step 3, $\ddot{X}(i, r, p)$, results of step 2, is multiplied with $C(i, s)$. The time chart to perform a N^3-size 3D-DCT is shown in Fig. 6. As shown in Fig. 6, each step has N *microsteps*. The processing sequence of a PE in each *microstep* is as follows:

1. Receive data from the adjacent PE and store it in the register.
2. Multiply received data with the coefficients that are prestored in the register on each PE. The calculation result is then stored in the register.
3. Transfer local data to the adjacent PE.

Transferred data in each step are

- input data for step 1,
- the calculation result of step 1 for step 2, and
- the calculation result of step 2 for step 3.

In each *microstep*, the coefficient data used in PE(i, j, k) are

- $C((k + microstep)\ mod\ N, k)$ for step 1,
- $C((j + microstep)\ mod\ N, j)$ for step 2, and
- $C((i + microstep)\ mod\ N, i)$ for step 3.

The coefficients are preset in each PE before processing. For the 3D-IDCT, data of the transposed coefficient matrix C^T are used. The elements of C^T are preset on the appropriate PEs.

The total number of calculation *microsteps* for the proposed $N \times N \times N$ torus array processor is $3N$ for an N^3-size

input data cube. Thus, the time complexity is $O(N)$ while that of the 3D-DCT sequential calculation is $O(N^4)$.

4. I/O Interfaces

Although our array processor does not require any complicated I/O circuits performing input data alignment, it has to handle N^3 data at once to obtain the final output data. Thus, dedicated I/O interfaces tuned to our array processor are designed.

The $N \times N$ I/O ports are connected to the $k = N-1$ (i, j)-plane of the proposed array processor. The I/O interface is provided to implement our array processor on 2D devices. Figure 7 is an overview of the I/O architecture. The architecture consists of six parts: an input memory, input address generator, input buffer, output buffer, output address generator, and output memory. The input part of the architecture performs the following operations in each *microstep*.

- Derive the input memory address using the input address generator.
- Get one pixel data from the input memory and put it into the proper FIFO of the input buffer.
- Update control signals for the input buffer, address generator, and array processor.

In addition, the input buffer sends N^2-size sub-frame data to the array processor simultaneously when the array processor requests data for the next calculation. The input memory is used as a frame buffer for the input data. This memory is connected to a host computer and the input buffer. The video data from the host computer are stored in this memory sequentially. The output data of the input memory are managed by an input address generator. The input memory has enough capacity to contain $2N^3$ data. The array processor begins its operations after the first N^3 data arrive at the input buffer from the input memory.

The input address generator produces the memory address for the input memory access in each clock cycle. It requests four parameters: the base address (BA), vertical resolution of video data (RV), horizontal resolution of video data (RH), and N. As shown in Fig. 8, the address generator has an adder and six address tables for address calculation. The contents of each address table are calculated using the following expressions:

$$
\begin{aligned}
&MD_x[i] = i,\ for\ 0 \le i < N \\
&MD_y[i] = RH \times i,\ for\ 0 \le i < N \\
&MD_z[i] = RH \times RV \times i,\ for\ 0 \le i < N \\
&BMD_x[i] = N \times i,\ for\ 0 \le i < RH/N \\
&BMD_y[i] = RH \times N \times i,\ for\ 0 \le i < RV/N \\
&BMD_z[i] = RH \times RV \times N \times i,\ for\ 0 \le i < 2
\end{aligned}
$$

Here, MD_x, MD_y, and MD_z are the x-, y-, and z-coordinates of the data in an $N \times N \times N$-size data block. BMD_x, BMD_y, and BMD_z are the x-, y-, and z-coordinates of the data block in the processing video data.

The input address generator derives proper memory address values from these six tables. A memory address is easily obtained using the following expression:

$$ADDR = BA + MD_x + MD_y + MD_z + BMD_x + BMD_y + BMD_z$$

Here, $ADDR$ is the target memory address for the input memory access.

The input buffer is used for timing gap elimination between the array processor and the input memory. Figure 9 overviews the input buffer architecture. The architecture has

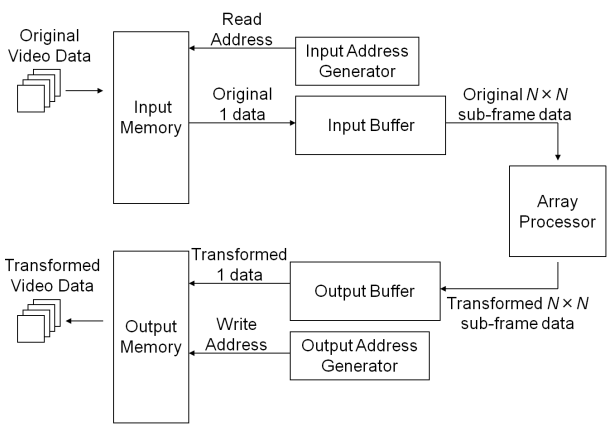

Fig. 7 Block diagram of I/O interface architecture.

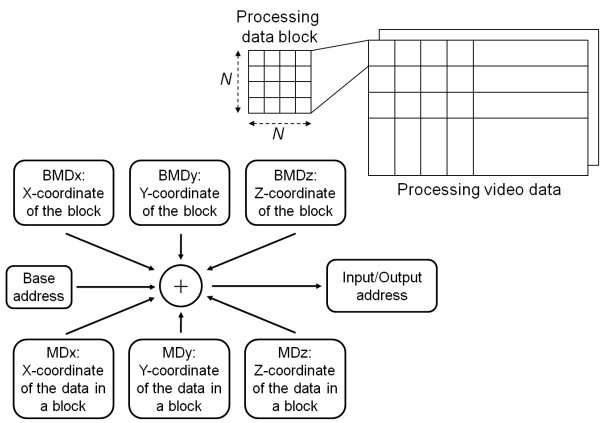

Fig. 8 An overview of the address generator.

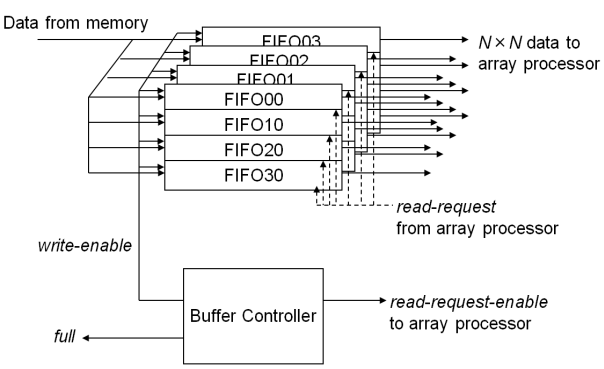

Fig. 9 An overview of the input buffer.

$N \times N$ asynchronous FIFOs and a buffer controller. The input buffer sequentially receives data from the input memory and puts them into proper FIFOs to generate $N \times N$-size sub-frame data. The input buffer sends sub-frame data at once to the array processor for the next 3D-DCT calculation. The buffer controller has three control signals: a *write-enable* signal sent to each FIFO, a *full* signal sent to the input address generator, and a *read-request-enable* signal sent to the array processor. The *write-enable* signal controls writing in FIFOs to store N^2-size sub-frame data correctly. The *full* signal is asserted when all FIFOs become full; the input address generator does not update the input memory address when *full* signal is asserted. The *read-request-enable* signal is asserted when FIFOs have enough data for the next 3D-DCT calculation; the array processor stops input operations when *read-request-enable* signal is asserted. The *read-request* signal from the array processor is directly passed to each FIFO to request the next data for the FIFO.

The data flow scheme of the output architecture is opposite to that of the input architecture. The output buffer is used for timing gap elimination between the output memory and the array processor output. Figure 10 overviews the output buffer architecture. The output buffer receives $N \times N$ data at once from the output ports of the array processor and sends them to proper locations of the output memory using a selector. As shown in Fig. 10, the buffer controller generates four control signals for the output buffer. The *select* and *read-enable* signals are used to control output data of the FIFO. The *write-request-enable* signal is asserted when each FIFO has enough capacity to receive N^3 data from the array processor; the array processor sends data to the output buffer when this signal is asserted. The *empty* signal is asserted when all FIFOs become empty. This signal is also sent to the output memory and the output address generator. The output memory address and data are updated only when the *empty* signal is negated. The output address generator generates the memory address for the output memory access. The processing sequence of the output address generator is the same as that of the input address generator. The output memory stores the output data from the output buffer. The data allocations in the output memory are the same as those in the input memory. The memory address is generated by the output address generator.

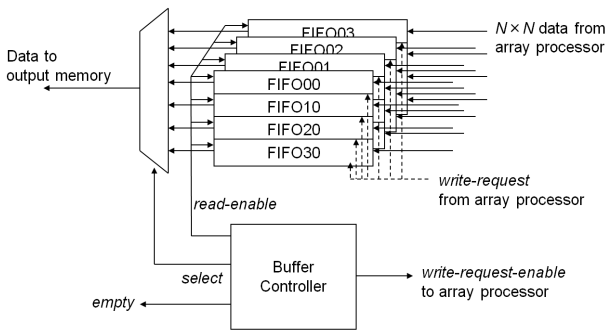

Fig. 10 An overview of the output buffer.

Using the proposed I/O interfaces, effective I/O data management is realized without data stream modification on the host computer.

5. Throughput Improvement

Hereafter, we call the array processor proposed in Sect. 3 as "original" architecture. To improve the throughput of original architecture, two pipelined architectures are also proposed. Although the improved architectures require more hardware resources than the original architecture, significant performance improvements are expected.

The first version of the pipelined architecture has PEs shown in Fig. 11 and has the following features:

- There are two MAC units in each PE.
- The number of registers to store calculated data is the same as original architecture has.
- I/O switches are used instead of the (de)multiplexers.
- Input ports are arranged on the $i = N - 1$ (j, k)-plane; each PE transfers input data to the $i - 1$ adjacent PE opposite to the i-axis.

The pipeline scheme is shown in Fig. 12. The columns represent pipeline stages and the rows represent time steps. Here, each step consists of N *microsteps*. There are three processing stages: stage A, B, and C. Each stage performs different process at each time step. In order to perform two MAC operations simultaneously, each PE has two MAC units. Here, the registers read/written on each time step are not overlapped; thus, the number of registers used for register files is same as the original architecture. So the critical paths are not lengthened, and no additional paths are used in this version. Instead, the I/O switches are used to control data flows, and the input ports are rearranged to prevent

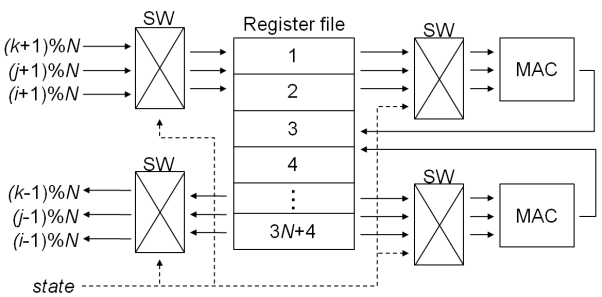

Fig. 11 PE structure of "Pipelined ver. 1".

Stage A Direction	Input *i*	Step 1 *k*	Step 2 *j*	Step 3 *i*	Output *k*		Input *i*	Step 1 *k*	Step 2 *j*
Stage B Direction			Input *i*	Step 1 *k*	Step 2 *j*	Step 3 *i*	Output *k*		Input *i*
Stage C Direction					Input *i*	Step 1 *k*	Step 2 *j*	Step 3 *i*	Output *k*

Time [*microstep*] *N*

Fig. 12 Pipeline scheme for "Pipelined ver. 1". In each box, top line shows the process in each stage; bottom line shows the data transfer direction.

data path overlapping in each processing stage. This architecture, "Pipelined ver. 1," performs 3D-DCT calculation at a rate that is 2.5 times faster that of the original architecture, theoretically.

The second version of the pipelined architecture has PEs shown in Fig. 13 and has following additional requirements:

- I/O operations are separated from the calculations. To do this, dedicated I/O ports: $in + 1$, $out + 1$, $in - 1$, and $out - 1$ are provided as shown in Fig. 13. Using these I/O ports, the data transfer to/from the adjacent PEs can be performed simultaneously with the calculation in the PE.
- There are three MAC units in each PE.
- The number of registers to store calculated data is doubled that of the original architecture.

The pipeline scheme is shown in Fig. 14; the rows and columns are the same as those in Fig. 12, except for the addition of "Input" and "Output" stages. They show the data I/O timing for the corresponding stage process. Since all three paths along the *i*-, *j*-, and *k*-axes are fully used in the three-step calculations, additional dedicated wires for the data I/O processes are required. Besides, to perform three MAC operations simultaneously, each PE has three MAC units. To keep the data integrity during calculations, the number of registers used for register files is doubled compared to the original one; the total number of registers is $3N + 8$.

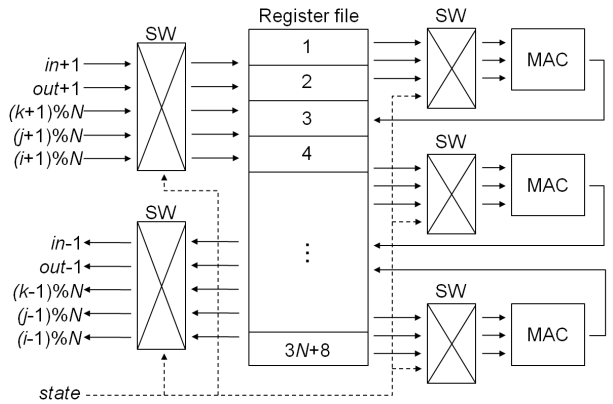

Fig. 13 PE structure of "Pipelined ver. 2".

Stage A Direction		Step 1 *k*	Step 2 *j*	Step 3 *i*	Step 1 *k*	Step 2 *j*	Step 3 *i*	Step 1 *k*	Step 2 *j*
Stage B Direction			Step 1 *k*	Step 2 *j*	Step 3 *i*	Step 1 *k*	Step 2 *j*	Step 3 *i*	Step 1 *k*
Stage C Direction				Step 1 *k*	Step 2 *j*	Step 3 *i*	Step 1 *k*	Step 2 *j*	Step 3 *i*
Input	Stage A	Stage B	Stage C	Stage A	Stage B	Stage C	Stage A	Stage B	Stage C
Output	-	-	-	-	Stage A	Stage B	Stage C	Stage A	Stage B

Time [*microstep*] *N*

Fig. 14 Pipeline scheme for "Pipelined ver. 2". In each box of the top three rows, top line shows the process in each stage; bottom line shows the data transfer direction. *Input* and *Output* rows show the input or output data for Stage X.

Under these conditions, "Pipelined ver. 2" theoretically performs at a rate 5.0 times faster that for the original architecture; however, its cost could be much higher than that of Pipelined ver. 1. In addition, its structure is more complex, especially in terms of wiring, and this could lengthen the critical path delay for the whole circuit. The practical performance improvement ratios are presented in Sect. 7.

6. Area Optimization

Our array processor requires $O(N^3)$ hardware resources, and it is important that this number is decreased for practical implementation. Figure 15 shows the estimated number of registers and MAC units in our original/pipelined architectures in the case of $N = 2$, 4, and 8. If we could perform N^3-size 3D-DCT using $(N/2)^3$-size architecture, the number of hardware resources would be polynomially less. We refer to this architecture as "Block" architecture.

Assuming N=4, a $2 \times 2 \times 2$-size array processor architecture performing 4^3-size 3D-DCT is shown in Fig. 16. The features of this Block architecture are as follows:

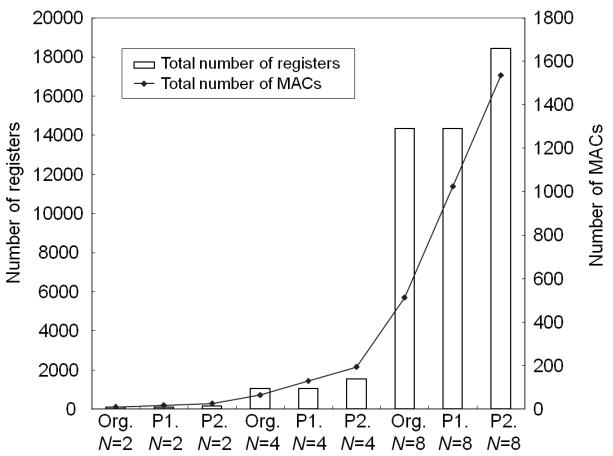

Fig. 15 Estimated number of registers and MAC units: Org: Original, P1: Pipelined ver. 1, P2: Pipelined ver. 2.

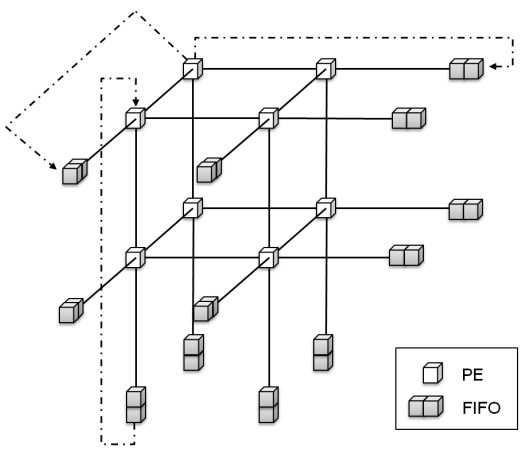

Fig. 16 4^3-size 3D-DCT calculation by using $2 \times 2 \times 2$-size array processor.

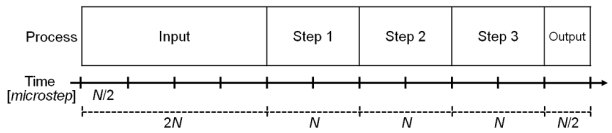

Fig. 17 Time chart to perform a $(N/2)^3$-size 3D-DCT.

- The structure of PE is the same as that of the original architecture.
- The N^3-size 3D-DCT calculation is divided into eight $(N/2)^3$-size calculations.
- $3N$ *microsteps* are needed for each $(N/2)^3$-size 3D-DCT calculation.
- FIFOs are inserted into each toroidal connection of the i-, j-, and k-axes. Each FIFO is composed of $N/2$ registers in its implementation.
- The number of operators is theoretically 1/8 of that for the original architecture.
- The number of coefficient registers in each PE is the same as for the N^3-size original architecture.

Figure 17 shows the time chart to perform a $(N/2)^3$-size 3D-DCT. This "partial" 3D-DCT has similar processing sequence as the original architecture, but the number of *microsteps* to perform each process is different from that of the original one. On the original architecture, all of the processes: input, calculation step 1, 2, and 3, and output take N *microsteps*. On the other hand, the Block architecture takes: $N/2 \times 4$ *microsteps* for input, $3N$ *microsteps* for calculations, and $N/2$ *microsteps* for output. Thus, in order to commit a N^3-size 3D-DCT and its I/O process, it takes $44N$ *microsteps* on this Block architecture; recall that the original architecture requires $5N$ *microsteps*.

However, compared with the original architecture proposed in Sect. 3, the number of MAC units is polynomially reduced. Thus, the architecture is easy to implement for area-limited hardware such as field-programmable gate arrays (FPGAs). In addition, the FIFOs attached to the array processor only require ordinary FIFO functions, and commercially available discrete FIFOs can be used in board-level integration. Note that we used an $(N/2)^3$-size processing array to simplify the proposed mechanism, but the mechanism is applicable to any $(N/m)^3$-size processing array, where $m = 2, 4, 8, ..., N/2$.

7. Experimental Results

In this section, all of four proposed architectures are implemented on an FPGA. We have already discussed theoretical issues for the proposed architectures and shown their preliminary implementation results in [26], [27], but re-evaluate all architectures and compare them with other related works.

7.1 Performance Evaluation

First, we implemented the proposed architectures for an FPGA and evaluated their resource utilization and performance. The experimental parameters and conditions are as follows:

- $N = 4$
- Input data bit-width: 16 bits
- Target FPGA: Cyclone II EP2C70F896C6

Table 1 Performance and resource utilizations of proposed architectures for $4 \times 4 \times 4$ data (N=4, Target FPGA: Cyclone II EP2C70F896C6).

	Original	Pipelined ver.1	Pipelined ver.2	Block
The number of LEs (Logic Elements)	12,646	12,248	16,327	2,335
The number of registers	7,189	7,182	13,059	1,685
The number of I/O pins	522	522	521	140
The number of embedded 9-bit multipliers	128	216	288	16
Max. clock freq. [MHz]	77.9	84.4	79.5	89.3
Throughput [$\times 10^5$]	15.6	42.2	79.5	2.0

The experimental results are presented in Table 1 for the original array processor architecture proposed in Sect. 3, Pipelined ver. 1 and Pipelined ver. 2 with pipeline schemes presented in Sect. 5, and the Block architecture described in Sect. 6. To compare their performances, throughput is defined as the number of processed 3D-DCTs per second; a 4^3-size data cube is transformed each time. In this implementation, one *microstep* is one clock cycle. As shown in Table 1, Pipelined ver. 1, Pipelined ver. 2, and the Block architectures have certain advantages over the original architecture.

- Pipelined ver. 1 performs better than the original architecture by a factor of 2.7; it requires only 88 additional multipliers than the original architecture.
- Pipelined ver. 2 has the best performance. It performs better than the original architecture by a factor of 5.1.
- The Block architecture can be implemented with the lowest hardware cost: 12.5%-26.8% of the cost of the original architecture.

Since the MAC units and their peripherals in each PE are structurally parallelized by pipelining, the critical path delays of these architectures become shorter. This is the reason why the clock frequencies of the pipelined architectures are higher than that of the original architecture.

Table 1 also shows the resource utilizations on the target FPGA. Since the number of states (combinations of the processes in each *microstep*) of Pipelined ver. 1 is smaller than that of the original architecture, the control unit of Pipelined ver. 1 can be implemented by using smaller number of LEs (Logic Elements) and registers compared to the original architecture. Pipelined ver. 2 uses all three paths along the $i-$, $j-$, and $k-$ axes in each *microstep*, as shown in Fig. 14. There is no control signal to switch data-flows of these three paths, while the control signal of the original architecture and Pipelined ver. 1 have this switching control. Hence, the bit-width of the control signal is smaller than that

Table 2 Performance and resource comparison of the proposed architectures and the architecture proposed in [14] (N = 4, input data bit-width = 8 bits) (*1) (Normalized clock speed) = (Clock freq. of [14]) × (Max. clock freq. of "Cyclone II EP2C70F896C6") / (Max. clock freq. of "EPF10K100EFC484-1") (*2) Assume the number of data cube on one frame is 1200.

	Original	Pipelined ver.1	Pipelined ver. 2	Block	The architecture proposed in [14]
Max. clock freq. [MHz]	97.2	111.7	97.2	128.6	28.0 (*1)
Frame rate [fps] (*2)	4,050	11,630	22,030	609	61
The number of LEs	18,534	22,184	29,575	3,071	1,997
Relative ratio of frame rate per the number of LEs	7.2	17.2	24.4	6.5	1
Target FPGA	Cyclone II EP2C70F896C6				EPF10K100E FC484-1

of the original architecture and Pipelined ver. 1. This control signal is assigned to output pins. Thus, in Table 1, the total number of I/O pins of Pipelined ver. 2 is 1 bit smaller than that of the original architecture and Pipelined ver. 1. Theoretically, the numbers of registers in the pipelined architectures should be ×2 and ×3 of the original architecture, respectively. However, the numbers of embedded 9-bit multipliers in Table 1 do not follow this fact. This is because the input data bit-width is 16 bits in our implementation and some parts of multipliers are realized using LEs.

In Table 2, we compared our architectures with the architecture proposed in [14] which also realizes 3D-DCT. The experimental conditions are the same as [14]: N = 4; input data bit-width = 8 bits. Our architectures are implemented using an Altera Cyclone II EP2C70F896C6. We use frame rate (frames per second) to compare the performance. Since [14] implements their architecture using an Altera EPF10K200EFC484-1, we normalized the performance using the following equation, by referring the device data sheets.

$$\begin{aligned}&\textit{Normalized clock speed}\\&= \textit{Clock freq.of}\,[14]\times\frac{\textit{Max. clock freq. of ``CycloneIIEP2C70F896C6''}}{\textit{Max. clock freq. of ``EPF10K100EFC484-1''}}\\&= \textit{Clock freq. of}\,[14]\times\frac{402.5\ MHz}{180.0\ MHz}\end{aligned}$$

Table 2 also shows hardware resource, i.e., the number of LEs, used to realize the corresponding architecture. It takes the embedded multipliers into account. Although the proposed architectures use 1.5 - 15.0 times hardware resources, they perform 10 - 361 times better throughput, compared to the architecture proposed in [14]. In fact, their relative ratios of the frame rate per the number of LEs, which indicate the relative performance per hardware resource, are improved by factors of 6.5 - 24.4.

7.2 Minimum Clock Frequency to Perform Real-Time 3D-DCT

We also evaluated the theoretical minimum clock frequencies required to perform real-time 3D-DCT analytically. We assume N = 4; the required clock frequencies of the proposed architectures under 32 and 60 fps (frame per second) conditions are presented in Tables 3 and 4.

For VGA-size input video data under 32/60 fps conditions, the minimum clock frequencies are as follows:

- Original architecture: 3.1/5.8 MHz
- Pipelined ver. 1: 1.2/2.3 MHz
- Pipelined ver. 2: 0.6/1.2 MHz
- Block architecture: 27.1 MHz

VGA is one of the most frequently used video data formats. Except for the Block architecture, the presented frequency values are only a few MHz; such low clock frequency affords low power consumption.

For UXGA-size input video data under a 60 fps condition, the minimum clock frequencies are as follows:

- Original architecture: 36.0 MHz
- Pipelined ver. 1: 14.4 MHz
- Pipelined ver. 2: 7.2 MHz
- Block architecture: 316.8 MHz

UXGA is considered a high-quality video format. The frequency values of the pipelined architectures are less than 20 MHz. Playing high-quality video on portable devices is an upcoming demand, and this architecture will be useful in such applications.

As shown in the above cases, the Block architecture requires much higher clock frequency than other architectures for real-time processing; thus, high-resolution video coding using this architecture is impractical at present. However, using the implemented circuit presented in the previous section, it can handle XGA-size video data of 32 fps and SVGA-size video data of 60 fps. Therefore, this architecture is suitable for smaller-size video encoding.

Table 3 Minimum operating clock frequency for real-time 3D-DCT (N=4, 32 fps).

	Size	Min. clock frequency for real-time 3D-DCT [KHz]			
		Original	Pipelined ver.1	Pipelined ver.2	Block
QVGA	320 × 240	768	307	154	6,758
VGA	640 × 480	3,072	1,229	614	27,034
SVGA	800 × 600	4,800	1,920	960	42,240
XGA	1024 × 768	7,864	3,146	1,573	69,206
SXGA	1280 × 1024	13,107	5,243	2,621	115,343
UXGA	1600 × 1200	19,200	7,680	3,840	168,960

Table 4 Minimum operating clock frequency for real-time 3D-DCT (N=4, 60 fps).

	Size	Min. clock frequency for real-time 3D-DCT [KHz]			
		Original	Pipelined ver.1	Pipelined ver.2	Block
QVGA	320 × 240	1,440	576	288	12,672
VGA	640 × 480	5,760	2,304	1,152	50,688
SVGA	800 × 600	9,000	3,600	1,800	79,200
XGA	1024 × 768	14,746	5,898	2,949	129,761
SXGA	1280 × 1024	24,576	9,830	4,915	216,269
UXGA	1600 × 1200	36,000	14,400	7,200	316,800

7.3 Resource Utilization

In this section, we discuss the resource utilization of the proposed architectures. Tables 5–7 present the resource utilizations of the architectures when N = 2, 4, and 8. The implementation target is Cyclone III EP3C120F780C8 FPGA.

Table 5 Resource utilization (N=2, Target FPGA: Cyclone III EP3C120F780C8).

	Original	Pipelined ver.1	Pipelined ver.2	Block
The number of LEs (Logic Elements)	1,514	1,689	2,396	-
The number of registers	910	905	1,602	-
The number of I/O pins	138	138	136	-
The number of embedded 9-bit multipliers	16	20	24	-

Table 6 Resource utilization (N=4, Target FPGA: Cyclone III EP3C120F780C8).

	Original	Pipelined ver.1	Pipelined ver.2	Block
The number of LEs (Logic Elements)	12,642	12,448	16,479	2,332
The number of registers	7,189	7,182	13,059	1,685
The number of I/O pins	522	522	521	140
The number of embedded 9-bit multipliers	128	216	288	16

Table 7 Resource utilization (N=8, Target FPGA: Cyclone III EP3C120F780C8).

	Original†	Pipelined ver.1†	Pipelined ver.2†	Block
The number of LEs (Logic Elements)	117,847	184,396	306,188	19,355
The number of registers	57,379	57,367	105,476	13,347
The number of I/O pins	2,059	2,059	2,058	525
The number of embedded 9-bit multipliers	576*	576*	576*	128

The input data bit-width is 16 bits. In Table 7, the symbol "$*$" in each table indicates that FPGA-implemented multipliers are fully used. Thus, some multipliers are implemented by using LEs. In addition, the symbol "†" indicates the corresponding three architectures, i.e., the original architecture, Pipelined ver. 1, and Pipelined ver. 2, cannot be implemented in the target Cyclone III FPGA because of its resource limitations. The values shown in the corresponding columns are the estimated values obtained by using Altera Quartus II, a logic design tool.

As shown in the second columns of Tables 5-7, the numbers of embedded multipliers are $2 \times N^3$ in the original architecture since the multiplication of 16-bit input data is performed by two embedded multiplier 9-bit elements on each PE. Because of the resource limitation of the FPGA, in the N=8 case, some multiplications are implemented using logic elements. This is one reason why the resource utilization increases greatly from the case N=4 to the case N=8. The third and fourth columns show the resource utilizations of the pipelined architectures. The numbers of LEs in the pipelined architectures are much higher than the number in the original architecture. In Sect. 7.1, Pipelined ver. 2 had the best performance among all proposed architectures. However, the resource cost of this architecture is more than twice that of the original architecture when N becomes large. On the other hand, as shown in Table 7, the Block architecture uses less than 1/6 of the resources of the original architecture when N=8; this architecture is the most practical for the implementation target Cyclone III EP3C120F780C8 FPGA.

Each of the proposed architectures has an advantage in terms of its performance or hardware cost. Thus, an appropriate architecture should be selected on the basis of the target FPGA and the target application to obtain the best balance of cost and performance.

8. Conclusion

In this paper, we proposed 3D array processors dedicated to 3D-DCT calculation and presented their implementation results for an FPGA. The array processor drastically reduced data swapping or replacement during the 3D-DCT calculation and showed significantly improved performance. The computational complexity of the proposed array processor is $O(N)$ for an N^3 input data cube, and that of the 3D-DCT sequential calculation is $O(N^4)$. In addition, the advanced architectures featuring data pipelining were proposed to improve the data throughput. The pipelined architectures performed 3D-DCT calculation faster than the original architecture based on sequential data handling by factors of 2.7 and 5.1. Furthermore, the Block architecture was proposed. This architecture could perform 8^3-size 3D-DCT using a 4^3-size processor array, which reduced the hardware cost to less than 1/6 of the cost of the original architecture. Evaluation results showed that the Block architecture, with adaptable scalability, provides good performance for real-time applications.

References

[1] M. Servais and G.D. Jager, "Video compression using the three dimensional discrete cosine transform (3D-DCT)," Proc. Communications and Signal Processing (COMSIG), pp.27–32, 1997.

[2] X. Li and B. Furht, "An approach to image compression using three-dimensional DCT," Proc. Visual 2003 Conference, 2003.

[3] F. Bai and A. Jamalipour, "3D-DCT data aggregation technique for regularly deployed wireless sensor networks," Proc. IEEE International Conference on Communication (ICC '08), pp.2102–2106, 2008.

[4] S. Saponara, L. Fanucci, and P. Terreni, "Low-power VLSI architectures for 3D discrete cosine transform (DCT)," Proc. 46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'03), vol.3, pp.1567–1570, 2003.

[5] T. Tsuchiya, K. Kirisawa, and H. Wakui, "A study of color motion picture coding using 3D-DCT," Proc. IEICE Gen. Conf. 1996, p.35, 1996.

[6] M.C. Lee, K.W. Chan, and D.A. Adjeroh, "Quantization of 3D-DCT coefficients and scan order for video compression," J. Vis. Commun. Image Represent., vol.8, no.4, pp.405–422, 1997.

[7] Y. Misuda, I. Matsuda, and S. Itoh, "3D-DCT video coding using a frame reordering method — Performance improvement by increase of the number of frames to be reordered," Proc. IEICE Gen. Conf., p.136, 1998.

[8] D. Rusanovsky and K. Egiazarian, "Postprocessing for three-dimensional discrete cosine transform based video coding," Proc. 7th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS'05), pp.618–625, 2005.

[9] H. Morishita and Y. Ohno, "Volume data compression using three-dimensional discrete cosine transform," IPSJ SIG Notes 95 (63),

pp.9–16, 1995.
[10] A. Norkin, A. Gotchev, K. Egiazarian, and J. Astola, "Low-complexity multiple description coding of video based on 3D block transforms," Proc. EURASIP Journal on Embedded Systems, vol.2007, pp.9–19, 2007.
[11] A. Burg, R. Keller, J. Wassner, N. Felber, and W. Fichtner "A 3D-DCT real-time video compression system for low complexity single-chip VLSI implementation," Proc. MoMuC2000, 1B-5-1, 2000.
[12] T. Mekky, S. Boussakta, and M. Darnell, "On the computation of the 3D-DCT," Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS) 2003, pp.1141–1143, 2003.
[13] T. Fryza and S. Hanus, "Algorithms for fast computing of the 3D-DCT transform," Proc. 15th International Workshop on the Systems, Signals and Image Processing (IWSSIP) 2008, pp.217–220, 2008.
[14] M. Bakr and A.E. Salama, "Implementation of 3D-DCT based video encoder/decoder system," Proc. 45th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS-2002), vol.2, pp.II-13–16, 2002.
[15] T. Kratochvil and T. Fryza, "Video compression hardware implementation using programmable media processor," Proc. 4th EURASIP Conference focused on Video/Image Processing and Multimedia Communications, vol.1, pp.299–304, 2003.
[16] O. Alshibami and S. Boussakta, "Fast algorithm for the 3-D DCT," Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing 2001, pp.1945–1948, 2001.
[17] M. Modarressi and H. Sarbazi-Azad, "Parallel 3-dimensional DCT computation on *k*-ary *n*-cubes," Proc. 8th International Conference on High Performance Computing in Asia Pacific Region (HPC Asia 2005), pp.91–97, 2005.
[18] D. Lavenier, P. Quinton, and S. Rajopadhye, "Advanced systolic design," in Digital signal processing for multimedia systems, ed. K.K. Parhi and T. Nishitani, pp.657–692, CRC Press, 1999.
[19] M. Koyanagi, T. Fukushima, and T. Tanaka, "Three-dimensional integration technology and integrated systems," Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific, pp.409–415, Jan. 2009.
[20] Intel Corporation, "Tera-scale Computing Research Program," Intel Corporation, http://techresearch.intel.com/articles/Tera-Scale/1421.htm, 2009.
[21] G. Loh, "3D-stacked memory architectures for multi-core processors," Computer Architecture, ISCA '08, 35th International Symposium, pp.453–464, June 2008.
[22] E.H. Heijne, "Gigasensors for an attoscope: Catching quanta in CMOS," IEEE Solid-State Circuits Newsletter, vol.13, no.4, pp.28–34, Oct. 2008.
[23] S.G. Sedukhin, T. Miyazaki, and K. Kuroda, "Orbital systolic algorithms and array processors for solution of the algebraic path problem," IEICE Trans. Inf. & Syst., vol.E93-D, no.3, pp.534–541, March 2010.
[24] S.M. Chai and D.S. Wills, "Systolic opportunities for multidimensional data streams," IEEE Trans. Parallel Distrib. Syst., vol.13, no.4, pp.388–398, 2002.
[25] H. Hoffmann, V. Strumpen, and A. Agarwal, "Stream algorithms and architecture," Technical Memo MIT-LCS-TM-636, 2003.
[26] Y. Ikegaki, T. Miyazaki, and S.G. Sedukhin, "An array processor performing 3D-DCT effectively," Proc. 71st National Convention of IPSJ, no.1, pp.137–144, 2009.
[27] Y. Ikegaki, H. Igarashi, T. Miyazaki, and S.G. Sedukhin, "An FPGA implementation of array processor performing 3D-DCT effectively," IEICE Technical Report, RECONF2009-61, 2010.

Yuki Ikegaki was born in 1986. He received his B.S. from the University of Aizu in 2008. He is a M.S. candidate of the University of Aizu in present. His current research interests are in the design of application-specific processors and reconfigurable hardware systems. Mr. Ikegaki is a student member of IPSJ.

Toshiaki Miyazaki is a professor of the University of Aizu. His research interests are in reconfigurable hardware systems, wireless sensor networks, and adaptive networking technologies. He received his Ph.D. degree in electronic engineering from Tokyo Institute of Technology in 1994. From 1983 to 2005, he had been worked for NTT. Dr. Miyazaki is a member of IEEE (CS, CAS, ComSoc), and IPSJ.

Stanislav G. Sedukhin is a professor and the vice-president of the University of Aizu. He received his Ph.D. and Dr. Sci. (habilitation) from the Russian Academy of Sciences in 1982 and 1993, respectively. His research interests are in architectural synthesis of application-specific array processors and massively-parallel algorithms. Dr. Sedukhin is a member of ACM, and IEEE.