Japanese Journal of Applied Physics **53**, 07KC14 (2014) **REGULAR PAPER**
http://dx.doi.org/10.7567/JJAP.53.07KC14

A real-time sound rendering system based on the finite-difference time-domain algorithm

Tan Yiyu[1*], Yasushi Inoguchi[1*], Yukinori Sato[1], Makoto Otani[2], Yukio Iwaya[3], Hiroshi Matsuoka[4], and Takao Tsuchiya[5]

[1]*Research Center for Advanced Computing Infrastructure, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923-1292, Japan*
[2]*Faculty of Engineering, Shinshu University, Nagano 380-8553, Japan*
[3]*Faculty of Engineering, Tohoku Gakuin University, Sendai 985-8537, Japan*
[4]*Faculty of Engineering, Tohoku University, Sendai 980-8577, Japan*
[5]*Department of Information Systems Design, Doshisha University, Kyotanabe, Kyoto 610-0394, Japan*
E-mail: yiyu-t@jaist.ac.jp; inoguchi@jaist.ac.jp

Received November 29, 2013; accepted April 30, 2014; published online June 26, 2014

Real-time sound rendering applications are memory-intensive and computation-intensive. To speed up computation and extend the simulated area, a real-time sound rendering system based on the hardware-oriented finite difference time domain algorithm (HO-FDTD) and time-sharing architecture is proposed and implemented by the field programmable gate array (FPGA) in this study. Compared with the traditional rendering system with parallel architecture, the proposed system extends by about 37 times in the simulated area because data are stored in the on-chip block memories instead of the D flip-flops. The hardware system becomes stable after 400 time steps in the impulse response. To render a three-minute Beethoven classical music clip, the hardware system carries it out in real-time while the software simulation takes about 63 min in a computer with 4 GB RAM and an AMD Phenom 9500 Quad-core processor running on 2.2 GHz.

1. Introduction

Sound rendering technologies are widely applied in many industrial and scientific fields. In sound rendering applications, a sound space is discretized into small grids. The governing equations of sound propagation are applied on each grid to analyze the sound behavior at discrete time steps. During analysis, the sound pressures of a grid and its neighbors at previous time steps are required. Amounts of data are read from and written into memory. Sound rendering applications are therefore computation-intensive and memory-intensive. Traditional sound rendering systems are based on computer simulation and calculate sound pressures grid by grid at each time step. Their performance improvement suffers from the arithmetic units and memory bandwidth of computers. Although the arithmetic performance may be improved by increasing the clock frequency or using parallel techniques in general-purpose microprocessors, the memory bandwidth is the bottleneck of performance improvement. During simulation, lots of data are read from and written into the external memory. Since there are no special hardware units tailored to accelerate data access, the memory system does not work effectively owing to the intensive data requirement and insufficient bandwidth of the external memory.[1)] For example, in a two-dimensional sound rendering system with parallel architecture, if data are 32 bits, the system has 1,024 grids, and the clock frequency of the uniform computing cell at each grid is 100 MHz, the data throughput is 3,200 Gbps, which is beyond the memory bandwidth of current general-purpose computer systems.[2)] The simulation therefore becomes time-consuming even if computers become much faster.

General-purpose graphic processing units (GPGPUs) have been applied to enhance the arithmetic performance through coarse-grain parallelism of the arithmetic units in recent sound rendering systems.[3–6)] However, these general-purpose processors are designed for high arithmetic performance without sufficient memory bandwidth, and consequently computations are not efficient for the memory-intensive sound rendering. In addition, solutions based on GPGPU and computer simulations are not easily applied for real-time applications because their rendering results require further post-processing, particularly for multi-channel audio applications.

To address these problems, custom computing with programmable logic devices, such as FPGA, is promising in sound rendering applications, where the customized system allows processing modules to operate with high utilization. FPGA-based sound rendering solutions provide direct implementation of sound propagation equations by the configurable logic blocks (CLBs) inside a FPGA chip.[7–11)] By cascading hundreds of arithmetic units together, and coordinating them to work in parallel, a FPGA-based sound rendering system may achieve much higher computation performance than software simulations on generic computer systems. Since the utilization and interconnection of internal memory resources of a FPGA are explicitly tailored according to the system data flow, data are reused more efficiently, and the external memory access is reduced significantly. Furthermore, the input/output interfaces are easily built and attached for real-time applications.

In this paper, a real-time sound rendering system based on the HO-FDTD algorithm[8)] is proposed and implemented. The present work mainly discusses the system design and implementation based on a programmable hardware structure fully tailored for the sound rendering. The major contributions of this work are as follows:

1) Derivation of the boundary condition of the HO-FDTD algorithm.
2) Time-sharing architecture to extend the simulated area.
3) Design and implementation of the FPGA-based prototype system.
4) Detailed analysis and evaluation of system performance based on the prototype system, including system stability and rendering time.

The rest of this paper is organized as follows. Section 2 introduces the HO-FDTD algorithm and boundary conditions. Section 3 describes system design and implementation, including system architecture, system design, and so on. Section 4 presents the performance estimation of the prototype system. Finally, Sect. 5 concludes the paper with a summary and discussion of future work.

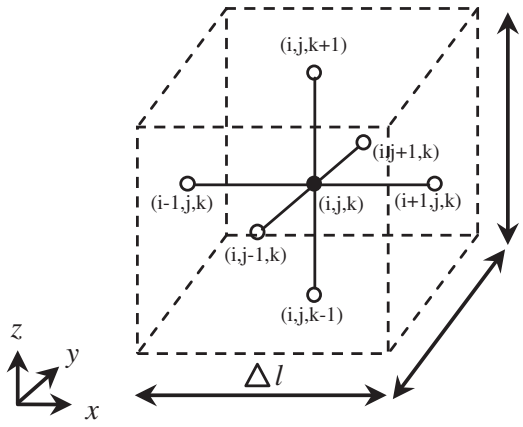

Fig. 1. A three-dimensional cubic element.

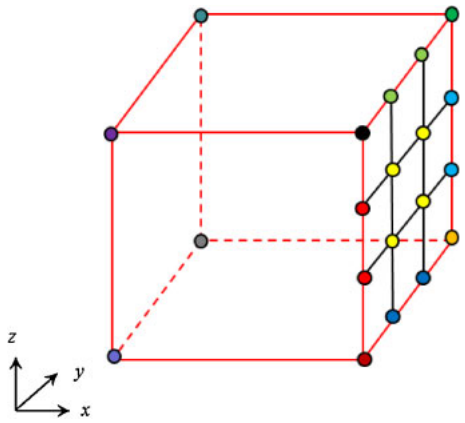

Fig. 2. (Color online) A rectangular sound space.

2. HO-FDTD algorithm

2.1 General grids

The FDTD algorithm is widely applied to analyze sound behavior.[12–16] In a cubic element shown in Fig. 1, sound wave propagation is governed by the following formula:

$$\frac{\partial^2 P}{\partial t^2} = c^2\left(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2}\right), \tag{1}$$

where P denotes the sound pressure, c is the sound speed, and x, y, z are the axes of a rectangular coordinate system. By applying the center differential method in Eq. (1), and let $\Delta x = \Delta y = \Delta z = \Delta l$, then

$$\begin{aligned} P^{n+1}(i,j,k) &= 2P^n(i,j,k) - P^{n-1}(i,j,k) \\ &+ \chi^2[P^n(i+1,j,k) + P^n(i-1,j,k) \\ &+ P^n(i,j+1,k) + P^n(i,j-1,k) + P^n(i,j,k+1) \\ &+ P^n(i,j,k-1) - 6P^n(i,j,k)], \end{aligned} \tag{2}$$

where $\chi = c\Delta t/\Delta l$ denotes the courant number. For a three-dimensional sound space, $\chi \leq 1/1\sqrt{3}$. When Eq. (2) is implemented by hardware, two multipliers are required, which consume more hardware resources and decrease the system clock frequency due to the long routing delay. To eliminate the multipliers, χ is assumed to be $1/2$, then Eq. (2) is changed as[8]

$$\begin{aligned} P^{n+1}(i,j,k) &= \frac{1}{4}[P^n(i+1,j,k) + P^n(i-1,j,k) \\ &+ P^n(i,j+1,k) + P^n(i,j-1,k) + P^n(i,j,k+1) \\ &+ P^n(i,j,k-1) + 2P^n(i,j,k)] - P^{n-1}(i,j,k). \end{aligned} \tag{3}$$

In Eq. (3), the multipliers are easily implemented by right and left shifters in hardware.

2.2 Boundary condition

Reflections from the boundaries of an acoustic space play a pivotal role for sound rendering, and therefore attention has been given to the problem of formulating numerical approximations of boundaries. A reflective boundary can be modeled as a locally reacting surface by assuming that waves do not propagate vibrations in the direction parallel to the boundary surface. The acoustical behavior of a reflective boundary is therefore based on the sound pressure and the particle velocity perpendicular to the boundary surface, and described by the boundary impedance. If a sound wave travels in a positive axis (x, y, z) direction, the boundary impedance Z relates the sound pressure to the particle vibration by[17]

$$Z = \frac{P}{U}, \tag{4}$$

where U denotes the particle velocity component perpendicular to the boundary. For sound waves traveling in a negative axis direction, Eq. (4) is changed to $Z = -P/U$. Differentiating both sides of Eq. (4), then

$$\frac{\partial P}{\partial t} = Z\frac{\partial U}{\partial t}. \tag{5}$$

For a boundary perpendicular to an axis, the momentum conservation equation of wave propagation is

$$\nabla P + \rho\frac{\partial U}{\partial t} = 0, \tag{6}$$

where ρ is the air density. Substituting $\partial U/\partial t$ in Eq. (5) with Eq. (6) yields the boundary condition in terms of sound pressure[18]

$$\frac{\partial P}{\partial t} = -c\xi\nabla P, \tag{7}$$

where $\xi = Z/\rho c$ is the normalized boundary impedance, also known as the specific boundary impedance.

For a rectangular sound space, such as the sound space with $4 \times 4 \times 4$ grids shown in Fig. 2, grids on boundaries are classified into three types according to their positions, namely, interior grid of a boundary, edge, and corner. Different equations are derived to update the sound pressures for different types of grids on boundaries. During derivation, the normalized boundary impedances of all boundaries are assumed to be ξ.

2.2.1 Interior grids of a boundary

When a grid is on the interior of a boundary, it matches the boundary condition defined by Eq. (7). For example, for the interior grids on the right boundary (the yellow points in Fig. 2), wave propagates along the positive x direction. Equation (8) is achieved by using the centered finite difference method on Eq. (7):

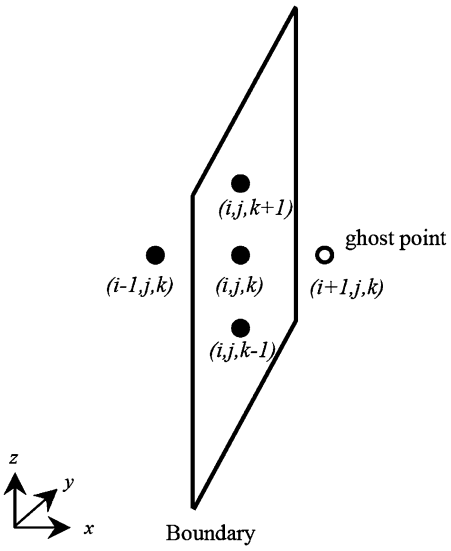

Fig. 3. Right boundary condition.

$$\frac{P^{n+1}_{i,j,k} - P^{n-1}_{i,j,k}}{2\Delta t} = -c\xi \frac{P^{n}_{i+1,j,k} - P^{n}_{i-1,j,k}}{2\Delta x}. \tag{8}$$

By rearranging the related terms in Eq. (8) and replacing $c\Delta t/\Delta x$ with χ, Eq. (9) is derived as an expression for a point lying outside of the modeled space, which is referred to as a "ghost point" shown in Fig. 3:[19)]

$$P^{n}_{i+1,j,k} = P^{n}_{i-1,j,k} + \frac{1}{\chi\xi}(P^{n-1}_{i,j,k} - P^{n+1}_{i,j,k}). \tag{9}$$

Inserting Eq. (9) into Eq. (2), then

$$P^{n+1}_{i,j,k} = \left[\chi^2(2P^{n}_{i-1,j,k} + P^{n}_{i,j+1,k} + P^{n}_{i,j-1,k} + P^{n}_{i,j,k+1} + P^{n}_{i,j,k-1}) + 2(1-3\chi^2)P^{n}_{i,j,k} + \left(\frac{\chi}{\xi} - 1\right)P^{n-1}_{i,j,k}\right] \Big/ \left(\frac{\chi}{\xi} + 1\right). \tag{10}$$

If the boundary reflection factor R is defined as $(\xi - 1)/(\xi + 1)$ and χ is assumed to be $1/2$, Eq. (10) is changed to

$$P^{n+1}_{i,j,k} = \left[\frac{1}{4}(2P^{n}_{i-1,j,k} + P^{n}_{i,j+1,k} + P^{n}_{i,j-1,k} + P^{n}_{i,j,k+1} + P^{n}_{i,j,k-1} + 2P^{n}_{i,j,k}) - \frac{3R+1}{2(1+R)}P^{n-1}_{i,j,k}\right] \Big/ \left(\frac{3+R}{2(1+R)}\right). \tag{11}$$

2.2.2 Edges When grids are on edges, which are intersections of two boundary planes, two boundary conditions are satisfied simultaneously. For example, the boundary conditions for the edge of the right and front boundaries (the red points in Fig. 2) are

$$\frac{\partial P}{\partial t} = -c\xi \frac{\partial P}{\partial x}, \quad \frac{\partial P}{\partial t} = c\xi \frac{\partial P}{\partial y}. \tag{12}$$

In Eq. (12), the first formula is the boundary condition for the right boundary, and the second one is for the front boundary where the wave propagates in the negative y direction.

By using the centered finite difference method on Eq. (12), the sound pressures of the corresponding ghost points are calculated using Eq. (13):

$$P^{n}_{i+1,j,k} = P^{n}_{i-1,j,k} + \frac{1}{\chi\xi}(P^{n-1}_{i,j,k} - P^{n+1}_{i,j,k}), \quad P^{n}_{i,j-1,k} = P^{n}_{i,j+1,k} + \frac{1}{\chi\xi}(P^{n-1}_{i,j,k} - P^{n+1}_{i,j,k}). \tag{13}$$

By inserting Eq. (13) into Eq. (2), then

$$P^{n+1}_{i,j,k} = \left[\chi^2(2P^{n}_{i-1,j,k} + 2P^{n}_{i,j+1,k} + P^{n}_{i,j,k+1} + P^{n}_{i,j,k-1}) + 2(1-3\chi^2)P^{n}_{i,j,k} + \left(\frac{2\chi}{\xi} - 1\right)P^{n-1}_{i,j,k}\right] \Big/ \left(\frac{2\chi}{\xi} + 1\right). \tag{14}$$

Eliminating ξ by R and assuming χ to be $1/2$ in Eq. (14), Eq. (15) is derived to update the sound pressures of grids on the edge of right and front boundaries.

$$P^{n+1}_{i,j,k} = \left[\frac{1}{4}(2P^{n}_{i-1,j,k} + 2P^{n}_{i,j+1,k} + P^{n}_{i,j,k+1} + P^{n}_{i,j,k-1} + 2P^{n}_{i,j,k}) - \frac{2R}{1+R}P^{n-1}_{i,j,k}\right] \Big/ \left(\frac{2}{1+R}\right). \tag{15}$$

2.2.3 Corners If a grid is at the intersection of three boundary planes, three boundary conditions are required simultaneously. For example, for the intersection grid of front, left, and top boundaries (the purple point in Fig. 2), the boundary conditions are

$$\frac{\partial P}{\partial t} = c\xi \frac{\partial P}{\partial x}, \quad \frac{\partial P}{\partial t} = c\xi \frac{\partial P}{\partial y}, \quad \frac{\partial P}{\partial t} = -c\xi \frac{\partial P}{\partial z}. \tag{16}$$

The formulas in Eq. (16) show the boundary conditions for the left, front, and top boundaries, respectively. Equation (17) is obtained to update the sound pressure of the corner by grids and edges:

$$P^{n+1}_{i,j,k} = \left[\frac{1}{4}(2P^{n}_{i+1,j,k} + 2P^{n}_{i,j+1,k} + 2P^{n}_{i,j,k-1} + 2P^{n}_{i,j,k}) - \frac{5R-1}{2(1+R)}P^{n-1}_{i,j,k}\right] \Big/ \left(\frac{5-R}{2(1+R)}\right). \tag{17}$$

Equations (11), (15), and (17) show that the sound pressures of grids on a boundary are calculated using the sound pressure of its neighbors at previous time steps, and no data dependency exists during computation. The equations consist of two parts. One is the sum of the sound pressures of a grid and its neighbors at the time step $n-1$, and the other is the sound pressure of a grid at the time step $n-2$, i.e., $P^{n-2}_{i,j,k}$. Furthermore, these equations have a similar format except for the multiplicands for different type of grids. Thus two uniform multipliers can be applied to implement different boundary conditions by hardware. For the same type of grids on boundaries, the multiplicands of two multipliers are same, and the terms in the sum part are different. For example, when a grid is located at the interior of the left boundary, the multiplicands in the equation to calculate the sound pressure are same as those in Eq. (11), but the sum part is changed from $(2P^{n}_{i-1,j,k} + P^{n}_{i,j+1,k} + P^{n}_{i,j-1,k} + P^{n}_{i,j,k+1} + P^{n}_{i,j,k-1} + 2P^{n}_{i,j,k})$ to $(2P^{n}_{i+1,j,k} + P^{n}_{i,j+1,k} + P^{n}_{i,j-1,k} + P^{n}_{i,j,k+1} + P^{n}_{i,j,k-1} + 2P^{n}_{i,j,k})$.

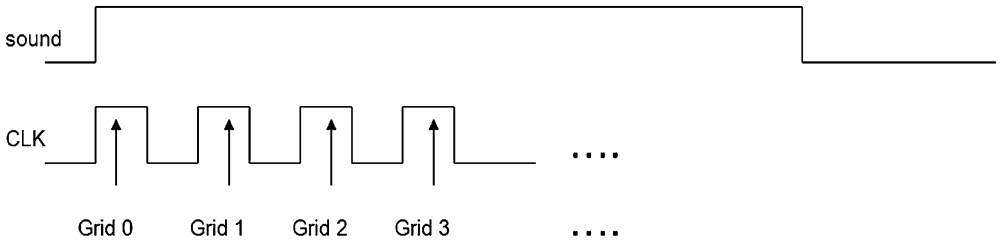

Fig. 4. Principle of time-sharing architecture.

3. System design

In traditional rendering systems based on the parallel architecture, a uniform computing unit is applied at each grid to model the acoustical phenomena. At each time step, computing units read data from their neighbors, carry out computations, and finally output the calculation results to their neighbors. This procedure loops again until the calculated time steps are finished. The main disadvantage of these rendering systems is the small simulated area by a FPGA chip. Since a computing unit is located at each grid, the hardware resources are increased linearly with the number of grids. When a sound space becomes larger, the hardware resources increase significantly. Hence, a sound rendering system with small number of grids is implemented by a FPGA chip owing to its limited hardware resources. To enlarge the simulated area, many FPGA chips are required to be cascaded and work in parallel, but data exchange between FPGA chips is a troublesome problem when the clock frequency is increased. An alternative solution is to apply the time-sharing architecture.

3.1 Time-sharing architecture

Because computing units generally run at a high clock frequency, and the sound frequency is relatively low, a computing unit can be used to analyze several grids. As shown in Fig. 4, the sound pressure of a grid is calculated at each clock cycle by the computing unit. The computation is moved to the next grid at the next clock cycle until the sound pressures of all grids are obtained. In principle, the sound pressures of all grids are calculated grid by grid through a computing unit in the time-sharing architecture, and hardware resources are saved. If a computing unit runs at 100 MHz, and the sound frequency is 10 kHz, a computing unit may calculate 10^4 grids. During calculation, the sound pressures of all grids are required to be kept for further computation. From Eqs. (3), (11), (15), and (17), two RAMs are required to store the sound pressures at the time steps $n-1$ and $n-2$.

When two RAMs are used to store data, an incident datum is read into the system after system initialization. The reading addresses of the RAMs are generated according to the grid position. On the basis of these addresses, the related data are read from two RAMs, computations are carried out by the computing unit, and the calculation result is written back into the corresponding RAM. Computations are then moved to the next grid, and the calculation procedure repeats until the sound pressures of all grids are obtained. Once computations at a time step are completed, the writing/reading operations on two RAMs are switched. Another incident datum is then read into the system, and computations occur again at the next time step. This whole procedure is iterated until the calculated time steps are over.

3.2 System architecture

The whole rendering system, as shown in Fig. 5, consists of two FPGA boards, and each FPGA board contains two FPGA chips. The incident signal, such as a song, is sampled by a high-speed A/D board (ADS5474), which is attached to the FPGA 1 on board 1. Then, the sampled data are processed by the rendering engine DHM module. The sound pressures at the observation point are sent to the FPGA 2 through the extended data transfer module (EDT_IF). Finally, they are transferred to the D/A board (DAC5682Z) on board 2 through the ATCA bus, and output to drive the speaker system. The whole procedure is handled in real-time. The hardware system is extended by modifying the data transfer interface between FPGAs (EDT_IF) and FPGA boards (ATCA_IF) to make multiple FPGAs work in parallel to enlarge the simulated area. As shown in Fig. 5, only one FPGA is applied as the computing engine in the current system.

3.3 System design

The system diagram of the DHM module, which is the sound rendering engine, is shown in Fig. 6. The DHM module mainly consists of the Computing unit, Grid position controller, System controller, buffers, and two block memories: Block_RAM_1 and Block_RAM_2. The Computing unit is the arithmetic unit used to calculate the sound pressure of a grid according to the input sound pressures of its neighbors. The Grid position controller generates the grid position. The System controller maintains the computation flow and generates control signals according to the grid position, such as the read/write enable signal (we) of the block RAMs, addresses to read out data from or write the calculation results into the block RAMs (raddr_RAM and waddr_RAM), and the RAM selection signal (ram_we_sel). Two block memories are used to alternatively store the sound pressures of all grids at different time steps. Buffers are applied to read data in advance to reduce latency during calculation.

In Fig. 6, the Computing unit, Grid position controller, and System controller are simple, and most of the hardware resources are consumed by the block RAMs and buffers. The size of each block RAM is determined by the number of grids and the data width. If the data width is 32 bit, and the sound space is divided into $N \times M \times L$ grids, each block RAM is $4NML$ bytes and each buffer is $4NM$ bytes in size. As the simulated space area becomes larger, the number of grids increases significantly, and the size of the block RAMs and buffers becomes larger too.

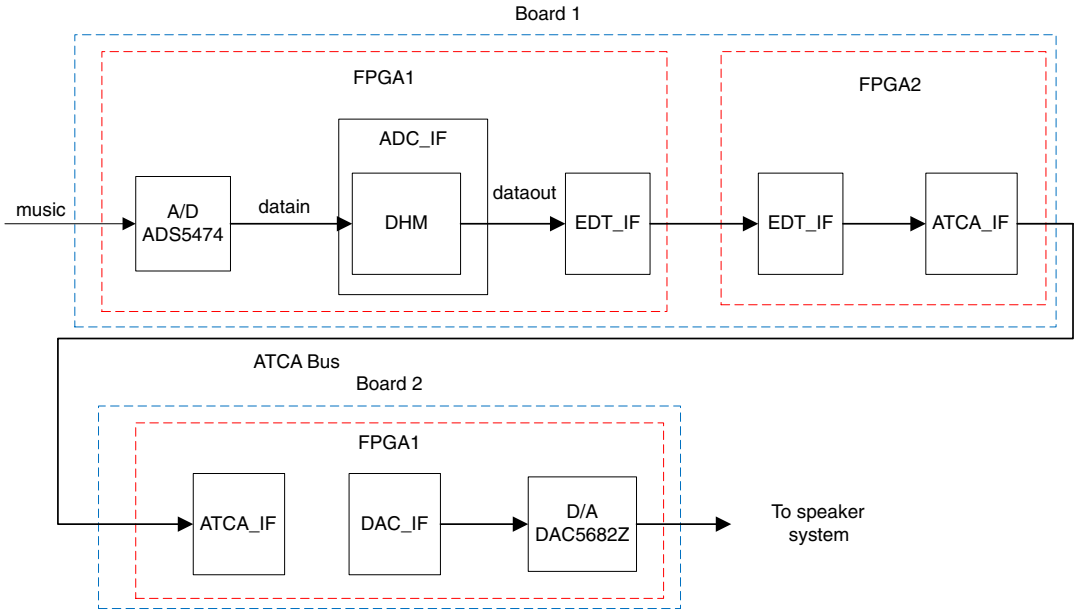

Fig. 5. (Color online) System architecture.

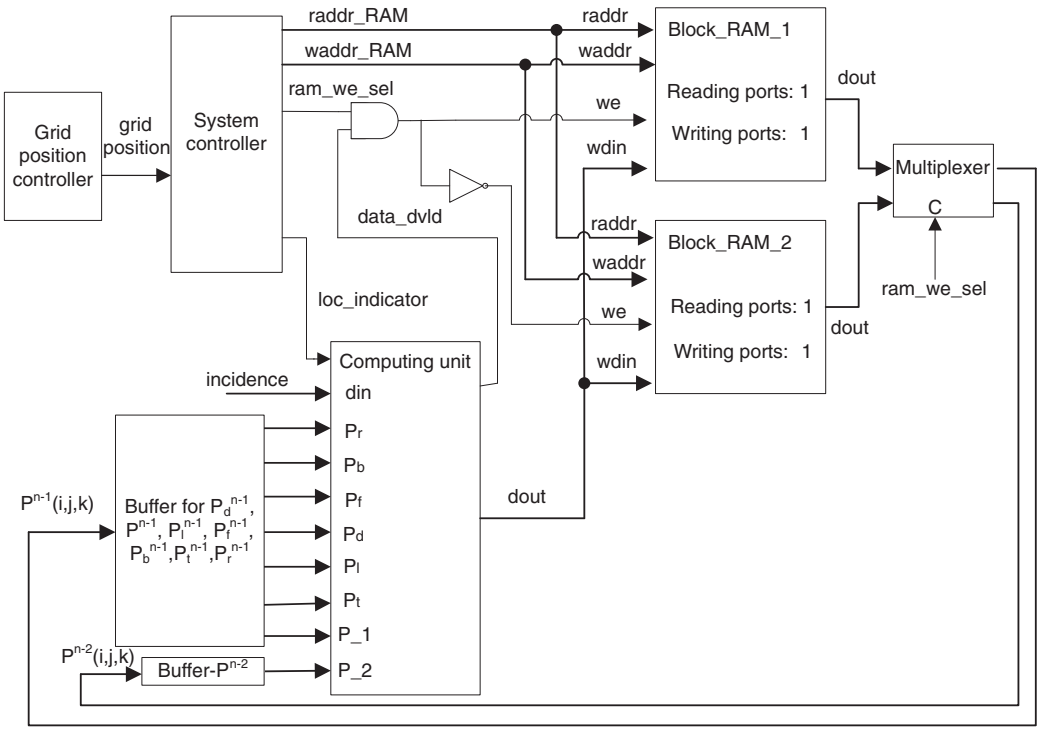

Fig. 6. Sound rendering system based on the time-sharing architecture.

3.3.1 Computing unit The Computing unit is designed based on the HO-FDTD algorithm. As discussed in Sect. 2, different equations are used to calculate the sound pressures according to the grid position, and they are similar except for the multiplicands. Based on this, a uniform computing unit is designed as shown in Fig. 7. The Computing unit contains one adder, one subtractor, two fixed-point multipliers, and four multiplexers. The multipliers are used to implement the boundary conditions. The multiplicands are calculated from the boundary reflection factor, and chosen by the signal Loc_indicator, which is a boundary flag to indicate the grid position. If a grid is not on a boundary, the multiplication operations are replaced by shift operations, and the multipliers are by passed. The adder (Adder1) is applied to sum up the sound pressures of the neighbor grids.

3.3.2 RAM operation As shown in Eq. (3), seven sound pressures at the time step $n-1$, and one sound pressure at the time step $n-2$ are required to calculate the sound pressure of a grid. As shown in Fig. 6, two block RAMs are used to store the sound pressures of all grids at the time steps $n-1$ and $n-2$. At each time step, data are read out from

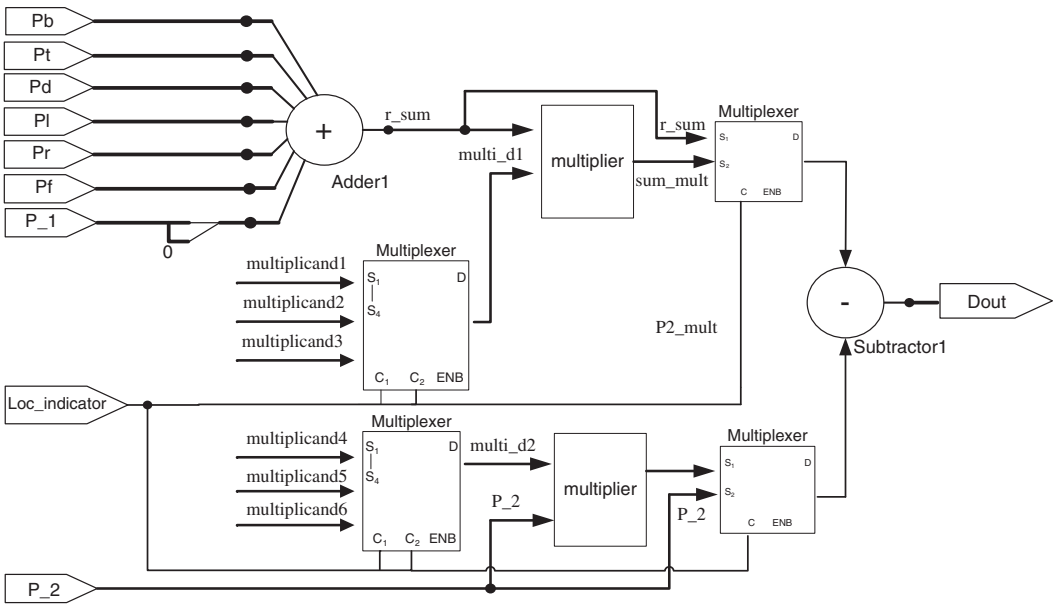

Fig. 7. Computing unit.

the block RAMs, calculation is carried out, and the results are stored back to the block RAMs. Block_RAM_1 and Block_RAM_2 are written alternately. For example, at a time step, seven data, namely, $P^{n-1}(i-1,j,k)$, $P^{n-1}(i+1,j,k)$, $P^{n-1}(i,j-1,k)$, $P^{n-1}(i,j+1,k)$, $P^{n-1}(i,j,k-1)$, $P^{n-1}(i,j,k+1)$, and $P^{n-1}(i,j,k)$, are read out from Block_RAM_2, and $P^{n-2}(i,j,k)$ is read out from Block_RAM_1 to the related buffers to calculate the sound pressure of a grid (i,j,k). When the sound pressures of all grids are obtained and stored, the reading and writing operations on the block RAMs are switched at the next time step. Seven data are then read out from Block_RAM_1 while $P^{n-2}(i,j,k)$ is taken from Block_RAM_2, and the calculation results are stored in Block_RAM_2. Such switching of reading and writing operations on block RAMs repeats along with the time steps increasing. The block RAMs are written in turn until all the calculated time steps are over. At the same time, two buffers are applied to reduce the latency during reading data.

The writing-enable signals of the block RAMs are controlled by the signal ram_we_sel from the System controller and the signal data_dvld from the Computing unit. A counter inside the Grid position controller is updated at every clock cycle. When its value is equal to the number of grids, which indicates that the computations at a time step are finished, the signal ram_we_sel is reversed to make the writing-enable signals of two block RAMs be inversed. The calculation results $P^n(i,j,k)$ are alternately written into one block RAM at the same time.

4. System performance

To verify and estimate the performance of the proposed sound rendering system, a three-dimensional sound rendering system with $32 \times 32 \times 16$ grids was investigated and implemented by a processor-based FPGA machine TD-SPP3000. The reflection factor of the boundaries was 0.95. The incident and observation points were at the middle of the sound space. The hardware development environment was a Windows XP platform with EDA tools Xilinx ISE 14.3 and ModelSim SE 10.1d. As a comparison, the related rendering systems based on the HO-FDTD and Yee-FDTD were developed by using the C++ programming language, and executed on a personal computer (PC) with 4 GB RAM and an AMD Phenom 9500 Quad-core processor running at 2.2 GHz. The operating system of the PC was Windows XP, and the development environment was Microsoft Visual Studio 2008. The reference C++ codes were compiled and optimized for the maximum speed with the option of /O2.

4.1 System stability

In sound rendering systems, when multiplications are replaced by shift operations in hardware, a numerical problem will be introduced.[20] For example, when a right-shift operation is performed by simply keeping the sign bit and truncating the rest bits, the result is rounded toward zero in the case of positive numbers and rounded away from zero in the case of negative numbers. Thus rounding down in both cases is carried out, and a negative offset is introduced into the system, which may eventually lead to numerical instability. In addition, since data are fixed-point in the hardware system, computational errors occur due to data truncation, which may be accumulated during calculation and make system divergent and unstable. To investigate system stability, the impulse response of the sound space, shown in Fig. 8, was obtained by the FPGA-based sound rendering system when the incidence was a pulse with amplitude of 16,384 Pa, and the time steps were 1000. Figure 8 shows that the rendering system becomes stable and convergent after 400 time steps. This system stability owes much to the compensation in the hardware system to solve the numerical problem resulting from the shift operations. For example, when the operand is negative and the least significant bit is 1 in one-bit right-shift operations, the result of shift operation is added by 1. Thus rounding toward zero is carried out in both the negative and positive cases, and the numerical problem is eliminated.

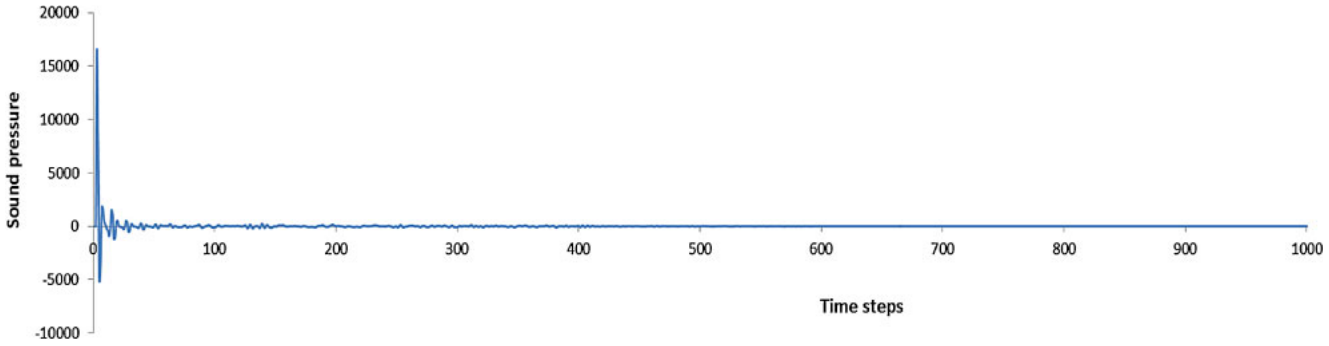

Fig. 8. (Color online) Impulse response.

Table I. Number of grids implemented by a XC5VLX330T.

Parallel architecture	Time-sharing architecture
1700	65536

Table II. Rendering time (s).

Grid	FPGA system	Software simulation	
	HO-FDTD	HO-FDTD	Yee-FDTD
$32 \times 32 \times 16$	Real-time	3799.921	3803.859

4.2 Simulated area

To estimate the system scale, different sound rendering systems with the parallel architecture and time-sharing architecture were described by VHDL, and synthesized by the ISE 14.3. Table I shows the maximum number of grids in the sound rendering systems implemented by a FPGA chip XC5VLX330T. When the time-sharing architecture is applied, the system scale is improved by about 37 ($65536/1700 - 1$) times, namely, the simulated sound area is increased by about 37 times. As shown in Figs. 6 and 7, data are stored in the block memories in the hardware system with the time-sharing architecture, while they are stored by the D flip-flops in the sound rendering system with the parallel architecture.[7,8] A FPGA chip XC5VLX330T contains 51,840 slices, and each slice includes four flip-flops. In contrast, it contains 11,664 Kb block RAMs and about 3 MB distributed RAMs. The capacity of the block memories is much larger than that of the D flip-flops, which results in the increase of the number of grids implemented by a FPGA chip in the rendering system based on the time-sharing architecture. The maximum system scale with the time-sharing architecture is consequently determined by the size of block RAMs inside a FPGA. When the whole rendering system with 65,536 grids was implemented by an XC5VLX330T, 57% of the block RAMs and 4% of the look-up tables were occupied.

4.3 Rendering time

Table II shows the rendering time taken by the software simulations and FPGA-based system when a three-minute Beethoven wave sound was as an incidence. The sampling rate of the wave sound was 44.1 kHz. In the FPGA-based rendering system, the Computing unit ran at 200 MHz. The wave sound was played by a player, and sampled by the A/D converter. The rendering results were output by the D/A converter to drive the speaker system directly. Data were 32 bit fixed-point in the hardware system while they were integers in the software simulations on the PC. As a comparison, the software simulations based on the HO-FDTD and Yee-FDTD were carried out and analyzed. The Courant number was $1/\sqrt{3}$ in the software simulation based on the Yee-FDTD while it was $1/2$ in the system based on the HO-FDTD.

In Table II, the rendering takes about 63 ($3799.921/60$) min in the computer-based software simulation based on the HO-FDTD algorithm while it is carried out in real-time in the FPGA-based sound rendering system. Although computations are performed grid by grid in both the FPGA-based rendering system and the software simulations in the PC, the system performance is improved significantly in the FPGA-based rendering system due to small data access overhead and parallel processing inside the FPGA. Compared with the software simulation based on the Yee-FDTD, although the simulation based on the HO-FDTD is a little faster, the HO-FDTD is simple, and the multiplication operations are substituted by shift operations during the calculation of the sound pressures of general grids. Thus the system data path in the computing unit is easily optimized in the hardware implementation, and the maximum clock frequency of the system is increased. For example, the maximum clock frequency in the computing unit based on the HO-FDTD is about 273 MHz while it is about 73 MHz in the computing unit based on the Yee-FDTD.[8]

Although the music is rendered in real-time in the current hardware system, the output music quality is worse than the software simulation results because of the low sampling rate. In the hardware system, the sampling rate of the output music (f_{sample}) is determined through the formula shown as

$$f_{\text{sample}} = \frac{M f_{\text{clk}}}{N}, \tag{18}$$

where f_{clk} is the clock frequency of the Computing unit, M denotes the number of computing units, and N is the number of grids. In Eq. (18), f_{sample} can be increased by increasing f_{clk} and M, or decreasing N. f_{clk} is usually determined by the maximum clock frequency of the Computing unit. N is limited by the hardware resources inside a FPGA, and kept as large as possible to extend the simulated area. In the current system, f_{clk} is 200 MHz, M is 1, and N is 16,384

$(32 \times 32 \times 16)$, then f_{sample} is about 12.5 kHz (200/16 kHz). Although a system with more grids, such as 65,536 $(32 \times 32 \times 64)$, can be implemented by a FPGA, f_{sample} is decreased to be about 3 kHz and the sound quality is bad and unacceptable.

On the other hand, the rendering system based on the Yee-FDTD was easily constructed by designing the computing unit with the Yee-FDTD, replacing the related modules, and adjusting the system timing, but the maximum system clock frequency was decreased to about 73 MHz because of the complexity and data dependency during calculation in the Yee-FDTD.[8] If f_{clk} is therefore chosen to be 70 MHz, f_{sample} is about 4.3 kHz (70/16 kHz), which results in the poor quality of the output sound.

To increase f_{sample}, an alternative solution is to use two or more computing units. If two computing units are applied, since there is no data dependency during calculation, the computations for grids 0 to $N/2-1$ and $N/2$ to $N-1$ are carried out by two computing units, respectively. The two computing units work in parallel and there is no overhead during computation. The calculation time at a time step is then shortened in half and f_{sample} is doubled. However, the disadvantages of this solution are:

(1) The buffers require more reading ports, which results in low utilization of the buffers and increasing of hardware resource consumption.
(2) The clock frequency to read out data from the block RAMs and write them into the buffers is increased.

Another solution to increase f_{sample} is to divide the sound space into several small spaces. For example, the system mesh $32 \times 32 \times 16$ is divided into four $16 \times 16 \times 16$ sub-meshes. Each sub-mesh is implemented as a small DHM module, and all small DHM modules are connected to each other to calculate the sound pressures in parallel. f_{sample} is therefore increased to be about 50 kHz $[200 \times 1024/(16 \times 16 \times 16)\,\text{kHz}]$. This solution is promising to increase f_{sample} because f_{sample} is easily determined by controlling the sub-mesh scale. However, buffers are required to exchange data between two neighboring small DHM modules, and the hardware resource consumption is slightly increased.

5. Conclusions

Sound rendering is computation-intensive. Due to limited hardware resources, a FPGA chip can be used to implement a sound rendering system with a small number of grids based on the parallel architecture. To extend the simulated area by a FPGA, time-sharing architecture is applied in this study, and a real-time rendering system based on it and the HO-FDTD is designed and implemented. The simulated area by a FPGA is enlarged by about 37 times against the rendering system with the parallel architecture because data are stored in the on-chip block memories instead of the D flip-flops. The sound field rendering is carried out in real-time while it takes a long time in the PC-based software simulations. Furthermore, since the computation time at a time step is sufficient enough to perform data exchange between FPGA chips, the proposed architecture is easily cascaded many FPGA chips to work in parallel to enlarge the simulated sound space.

On the other hand, computations in the rendering system with time-sharing architecture are much slower because they are carried out grid by grid. From this point of view, the simulated area is extended by the expense of the computation speed in the rendering system with time-sharing architecture. The sampling rate of the output sound is determined by the clock frequency of the computing unit, and the number of grids and computing units. When the number of grids is increased, the output sound quality becomes worse because of the low sampling rate, which can be increased by adding the computing units or using the sub-mesh partition technique. The related systems based on these solutions are under development.

Acknowledgement

This work was supported in part by the Strategic Information and Communications R&D Promotion Programme (SCOPE), Ministry of Internal Affairs and Communications.

1) S. Williams, A. Waterman, and D. Patterson, Commun. ACM **52**, 65 (2009).
2) Y. Tan, Y. Inoguchi, Y. Sato, Y. Iwaya, H. Matsuoka, M. Otani, and T. Tsuchiya, Proc. Int. Conf. ITNG, 2012, p. 484.
3) T. Ishii, T. Tsuchiya, and K. Okubo, Jpn. J. Appl. Phys. **52**, 07HC11 (2013).
4) T. Tsuchiya, Jpn. J. Appl. Phys. **49**, 07HC10 (2010).
5) L. Savioja, Proc. Int. Conf. DAFx, 2010, p. 1.
6) M. Tanaka, T. Tsuchiya, and K. Okubo, Jpn. J. Appl. Phys. **50**, 07HE17 (2011).
7) T. Yiyu, Y. Inoguchi, E. Sugawara, M. Otani, Y. Iwaya, Y. Sato, H. Matsuoka, and T. Tsuchiya, J. Sound Vib. **330**, 4302 (2011).
8) T. Yiyu, Y. Inoguchi, Y. Sato, M. Otani, Y. Iwaya, H. Matsuoka, and T. Tsuchiya, Jpn. J. Appl. Phys. **52**, 07HC03 (2013).
9) Y. Y. Tan, Y. Inoguchi, Y. Sato, M. Otani, Y. Iwaya, H. Matsuoka, and T. Tsuchiya, Proc. Int. Conf. DAFx, 2012, p. 93.
10) T. Yiyu, Y. Sato, E. Sugawara, Y. Inoguchi, M. Otani, Y. Iwaya, H. Matsuoka, and T. Tsuchiya, Proc. Int. Conf. Field Programmable Technology (FPT), 2010, p. 304.
11) Y. Inoguchi, Y. Y. Tan, Y. Sato, M. Otani, Y. Iwaya, H. Matsuoka, and T. Tsuchiya, Proc. Int. Conf. DAFx, 2011, p. 69.
12) Y. Miyazaki and T. Tsuchiya, Jpn. J. Appl. Phys. **51**, 07GB02 (2012).
13) N. Matsuda and S. Biwa, Jpn. J. Appl. Phys. **51**, 07GB14 (2012).
14) T. Yasui, K. Hasegawa, and K. Hirayama, Jpn. J. Appl. Phys. **52**, 07HD07 (2013).
15) T. Yoda, K. Okubo, N. Tagawa, and T. Tsuchiya, Jpn. J. Appl. Phys. **50**, 07HC12 (2011).
16) K. Kowalczyk and M. van Walstijn, IEEE Trans. Audio Speech Lang. Process. **19**, 34 (2011).
17) H. Kuttruff, *Room Acoustics* (Taylor & Francis, London, 2009) 5th ed., p. 36.
18) K. Kowalczyk and M. van Walstijn, Acta Acust. United Acust. **94**, 891 (2008).
19) J. W. Thomas, *Numerical Partial Differential Equations: Finite Difference Methods* (Springer, New York, 1998) 1st ed., p. 336.
20) S. A. Van Duyne and J. O. Smith, III, Proc. Int. Conf. Computer Music, 1993, p. 40.