OUKA
The University of Osaka
Institutional Knowledge Archive

Title	A Novel Dynamically Programmable Arithmetic Array (DPAA) Processor for Digital Signal Processing
Author(s)	Tan, Boon-Keat; Yoshimura, Ryuji; Matsuoka, Toshimasa et al.
Citation	IEICE Transactions on Fundamentals of Electronics. 2001, E84-A(3), p. 741-747
Version Type	VoR
URL	https://hdl.handle.net/11094/51657
rights	copyright©2001 IEICE
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

PAPER Special Section of Selected Papers from the 13th Workshop on Circuits and Systems in Karuizawa

A Novel Dynamically Programmable Arithmetic Array (DPAA) Processor for Digital Signal Processing*

Boon-Keat TAN[†a)], **Ryuji YOSHIMURA**[†], ***Nonmembers***, **Toshimasa MATSUOKA**[†], ***and*** **Kenji TANIGUCHI**[†], ***Regular Members***

SUMMARY A new architecture-based Dynamically Programmable Arithmetic Array processor (DPAA) is proposed for general purpose Digital Signal Processing applications. Parallelism and pipelining are achieved by using DPAA, which consists of various basic arithmetic blocks connected through a code-division multiple access bus interface. The proposed architecture poses 100% interconnection flexibility because connections are done virtually through code matching instead of physical wire connections. Compared to conventional multiplexing architectures, the proposed interconnection topology consumes less chip area and thus, more arithmetic blocks can be incorporated. A 16-bit prototype chip incorporating 10 multipliers and 40 other arithmetic blocks had been implemented into a 4.5 mm × 4.5 mm chip with 0.6 μm CMOS process. DPAA also features its simple programmability, as numerical formula can be used to configure the processor without programming languages or specialized CAD tools.

key words: *DPAA, DSP, parallel processing, interconnection topology, routing flexibility*

1. Introduction

With the advance of process technology, the performance of microprocessor has improved drastically over the past decades. Therefore, software-based microprocessors especially those based in RISC architecture, have emerged as popular solution for digital signal processing applications. RISC processors utilize pipelining as much as possible in order to parallelize tasks and to use the existing hardware resources efficiently. However, RISC are often inadequate to meet the requirements because constructing Digital Signal Processing (DSP) applications into long pipeline is difficult [1]. Architectures consisting of two cores, a microprocessor and a DSP core had been proposed [2] but the idea is not cost effective as hardware resources are often doubled. Therefore, RISC architectures, which combined DSP functionality and general-purpose architecture into a single core, had been introduced [1].

Highly specialized processors for DSP applications of today are not based on the RISC design philosophy. Instead, clusters of dedicated data paths, hard-wired to closely match the algorithmic data flow, are used. Such architectures typically contain multiple and concurrently operating data-path pipelines. Reconfigurable or programmable architectures with coarse granularity are incorporated such as rALU and EXU [3]–[5]. Although data-driven structure often uses complicated control sequences, rALU supports automatic mapping of arithmetic and allows programming from a high level language.

Reconfigurable structures with fine granularity such as FPGA, which has improved significantly in the 1990s, became an alternative solution for DSP. However, algorithms had to be expressed in hardware description language, then compiled and mapped by specified CAD tools. In addition, due to the difficulty in obtaining 100% routing flexibility, interconnection and chip utilization of FPGA is up to 70% and out of the utilized area, more than 50% of chip area is used for interconnections [6], [7].

The overall goal in designing this new architecture (DPAA) was to provide high parallelism, high pipelining ability and high area efficiency with simple programming interface. Consequently, DPAA features multiple and concurrently operating pipelines as those in data-path structures, but instead of data-driven, Harvard Architecture is used. Compared to such data-driven architecture [3], [4], DPAA supports more parallel tasks and is more superior in terms of area efficiency which is the weak point in most reconfigurable architectures. This is achieved due to its completely flexible interconnection interface for all processing. DPAA may also be used for any Custom Computing Machines (CCM) or any kind of adaptable computer system as well as a universal accelerator co-processor.

The architecture of DPAA is introduced in Sect. 2. An implementation of a prototype chip based on the proposed architecture is discussed in Sect. 3. Section 5 explains the programming aspect of DPAA and implementation of an adaptive equalizer using DPAA is presented as an example in Sect. 6 in order to give a better understanding of the proposed architecture. Advantages of DPAA are summarized in Sect. 4. The final section presents the future work for DPAA and concludes the paper.

Manuscript received June 28, 2000.

Manuscript revised September 20, 2000.

[†]The authors are with the Faculty of Engineering, Osaka University, Suita-shi, 565-0871 Japan.

a) E-mail: tan@eie.eng.osaka-u.ac.jp

*This paper was presented at The 13th Workshop on Circuits and Systems in Karuizawa, April 24-25, 2000.

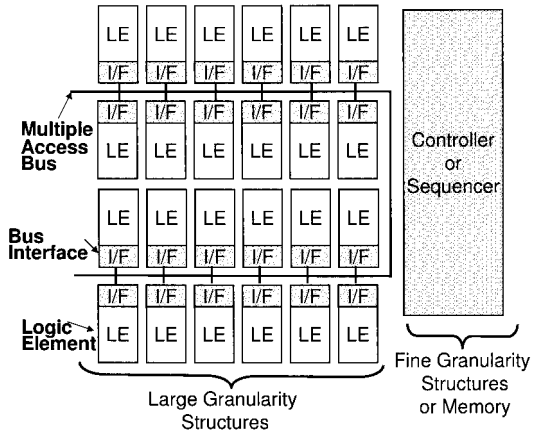


Fig. 1 Architecture and composition of DPAA.

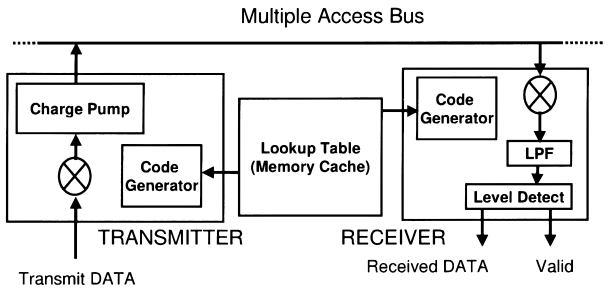


Fig. 2 Block diagram of bus interface.

2. Architecture

Figure 1 depicts the basic architecture of DPAA. Ideally, DPAA consists of a controller/sequencer and precompiled arithmetic logic elements (LE). In the proposed work, each logic element is designed to carry out a fixed arithmetic function. The LE can also consist of memory storage. All LEs are connected to a multiple access bus through a specialized interface proposed by one of our co-authors [8]. Multiplexing is implemented through the bus interface.

2.1 Bus Interface

As shown in Fig. 2, the bus interface consists of a transmitter and one or two receivers depending on the number of input/output of the logic elements. Circuit details of the interface are described in [8]. Therefore, only the structural and behavioral aspects of the interface will be dealt with in this paper. As shown in Fig. 3, the output of the logic element is modulated by Pseudo Noise (PN) sequence code. The modulated signal is then output to the multiple access bus through charge pumping circuit. At the receivers, the signals from the multiple access bus will be demodulated through mixers before being input to low pass filter where integration is being carried out. If the sequence code used during demodulation matches the code used in modulation, the signals will be received and control signals "VALID" will turn HIGH indicating that the codes match.

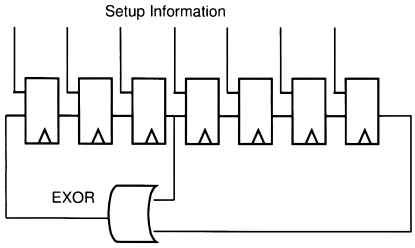


Fig. 3 PN sequence code generator.

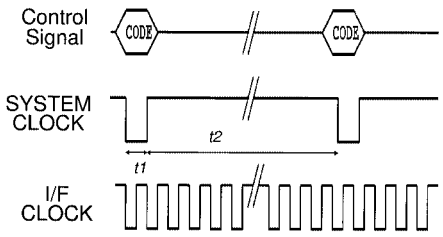


Fig. 4 Timing chart of bus interface.

The PN sequence code used in the proposed DPAA is 127 bit. The code generator is implemented by a 7-bit Linear Feedback Shift Registers (LFSR) as shown in Fig. 3. Hence, each bus interface blocks requires a 7-bit setup information [9] which is stored in a memory cache as shown in Fig. 2. For simplicity, all memory caches in the first generation DPAA are connected through several serial buses. The setup information will be loaded from the sequencer through these serial buses.

Figure 4 depicts the relation between the system clock and the bus interface clock. The code generator functions at a clock cycle 128 times the system clock, 127-cycle ($t2$) for code generation and an extra cycle ($t1$) to reload the setup information. The bus interface proposed by our co-author works at 200 MHz with 0.6 μm CMOS technology.

Figure 5 shows simulation results of the bus interface with 3 transmitters and 3 receivers. The waveform of the bus is also included. The multiple access bus exhibits a noise-like signal with small voltage swing. Peak to peak voltage swing for each signal transmitted on the bus is 20 mV. Receivers R1, R2 are connected to transmitters T1, T2 while receiver R3's connection is changed dynamically as the PN sequence code at R3 is changed every 2-bit cycles. Data received at R3 from each transmitter is shaded as shown in Fig. 5. In addition, simulation results verified that with 127-bit PN sequence code, up to 65 concurrent transmissions are supported by the proposed interface.

Our measurement results confirmed that the bit error rate of the bus interface is less than 10^{-8}. The bit error rate had been measured using 10 pairs of transmitters and receivers. The actual bit error rate

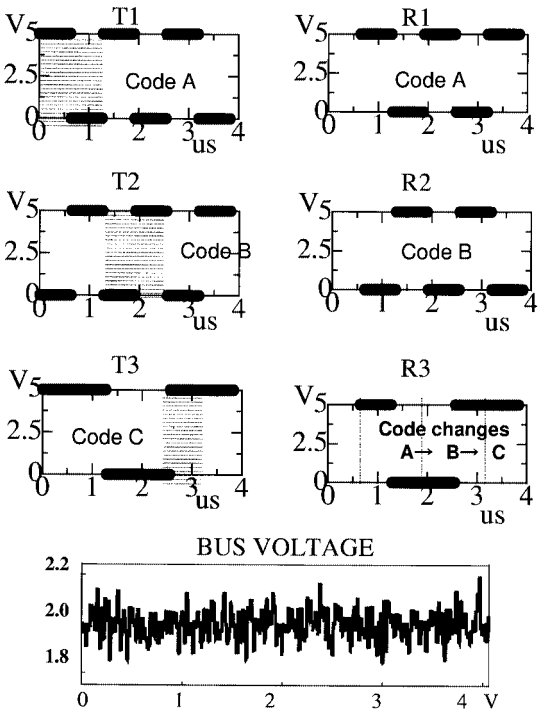


Fig. 5 Simulation results.

Table 1 Chip area: 16-bit ALU & components.

Delay Function	7,600 μm^2
Subtractor	18,100 μm^2
Multiplier	60,000 μm^2
Multiplexer (5input,1output)	15,400 μm^2
ALU (adder,subtractor,delay, multiplier)	80,000 μm^2

All Components are clock driven except multiplexer

of the multiple bus interface should be lower because the spread spectrum sequence code is noise-tolerant.

2.2 Logic Elements

Giving priority to the aspect of reconfigurability makes the designing of logic elements a difficult task, as designing precise functionality for general-purpose architecture is almost impossible. Programmable ALU is popular in most reconfigurable architecture. In contrast, DPAA consists of fixed arithmetic components such as adders, multipliers, subtractors and shifters. We choose to incorporate one arithmetic function into each LE in order to achieve better chip utilization.

For instance, Table 1 shows chip area required in implementing a 16-bit ALU and chip area required in implementing all of its function separately. Chip utilization in a programmable ALU is 30–85% because only one of its function will be used. Separating all functions enables simultaneous use of all functions and thus parallelism can be exploit. Using a fixed calculating block as a LE also has the advantage of reducing the complexity of programmability. As each LE is assigned

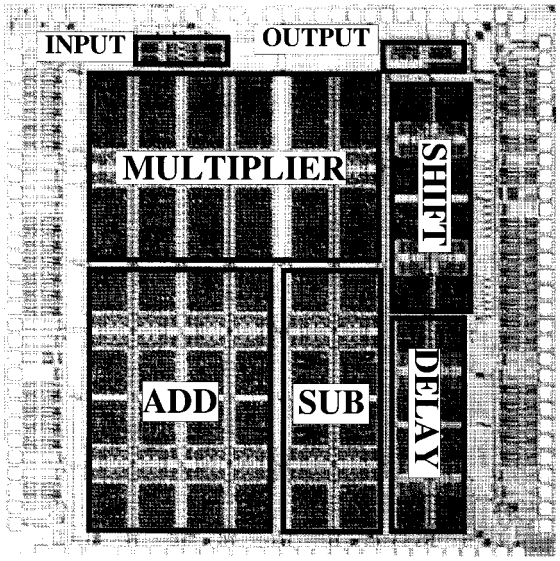


Fig. 6 Micrograph of prototype chip with 16-bit arithmetic logic. Fabricated with 0.6 μm CMOS Process, Chip Size: 4.5 mm × 4.5 mm, Performance 5.67MOPs.

to a unique PN sequence code, the program required for the DPAA consists of PN sequence codes for receivers at each LE. In addition, 100% chip utilization is possible because all of the LEs can work concurrently.

The number and combinations of LEs will be determined by the scale and types of applications. If the field of application is known, the ratio and combination of each LE can be computed by determining statistically how frequently each arithmetic function will be used while taking into consideration each popular algorithm. For instance, digital filters such as FIR require about the same number of adders and multiplier. Thus, the ratio of multipliers and adders should be approximately equal to one for the chip of digital filters.

3. VLSI Implementation

Figure 6 shows the micrograph of prototype chip of DPAA without the proposed controller or sequencer. The Lookup Table (LUT) in the bus interface is configured externally through serial buses. The prototype chip has 4 inputs and 4 outputs. The chip is fabricated by using a 0.6 μm CMOS, triple metal, double poly-silicon process and the chip size is 4.5 mm × 4.5 mm. The 16-bit processing elements are designated to function in serial so that good area utilization can be achieved.

Overall, the chip consists of 16 adders, 10 multipliers, 8 shifters, 8 subtractors and 8 delay blocks. Overflow is supported in adders, multipliers and shifters. Although the function of each arithmetic logic element is fixed, all adders, subtractors and multipliers can be converted into a delay block/register by setting one of the subtractor/adder's input to zero (addition of zero)

or by setting one of the multiplier's input to one (multiplication of one). Composition of the processing elements is not optimized to any application because the prototype chip is meant for feasibility checking purpose.

The performance of the prototype chip is 5.67MOPs because of the bus interface at research level used. The speed of the interface clock frequency can be increased from the current 200 MHz to GIGA hertz order by using circuit techniques such as MOS Current Mode Logic (MCML) [10]. However, The performance can be increased 2-times or 4-times even with 200 MHz-bus interface's clock rate by choosing the shorter PN sequence code and multiplexing of the multiple access bus.

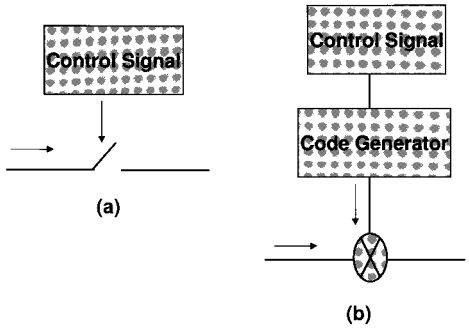


Fig. 7 Comparison between current bus interface and switching circuit.

4. Advantages of DPAA

1. High Chip Density
   Compared to conventional architecture, DPAA has higher chip density. Interconnection between logic elements usually consumes large silicon area but this is not true for DPAA as all connection is done using a single bus. For instance, the total chip area in the prototype chip for interconnection interface is 1,900,000 μm^2 in DPAA. However, implementing the interface using switch matrix circuit requires 2,800,000 μm^2. The number given excludes the area needed for wiring. The wiring area for the proposed multiple-access interface is negligible because all LEs are connected via a multiple access bus. However, wiring area for conventional switch matrix is large because it consists of more than 250 connections. In addition, unlike the proposed interface, buffers are required in the switch matrix approach.
2. Complete Routing Flexibility
   In addition to multiple access ability of the bus interface, the utilization of modulation/demodulation as connectivity also enables 100% routing flexibility because connections are done through code matching. Broadcasting of a datum is possible. Unlike conventional reconfigurable architectures, the connection topology in DPAA has no electrical or timing effect on the circuit. Therefore, physical arrangement and the topology of LE in DPAA can be optimized in order to achieve high chip utilization.
3. Dynamical Programmability
   Control signal (i.e. setup information) of the multiple bus interface is referred indirectly during a small portion of a clock time. Therefore, even when a data is being processed, transition of control signal has no effect on the interconnection of the logic elements. Hence, reloading of control signal is non-intrusive. However, the same does not apply on as switching circuits as shown in Fig. 7. Generally, a switching circuit is controlled directly by the control signal. Transition of control signal has direct effect on the switch and thus, reloading of control signal is intrusive.
4. Parallel and Pipelining
   Large number of processing elements and complete interconnection flexibility of DPAA can be utilized to realize pipelining and parallel processing. Pipelining and Parallel processing in conventional microprocessor architecture is difficult to achieved due to the limitation of interconnection flexibility and number of processing elements available. For instance, a 1 GHz RISC processor, although works at clock rate of GHz order, will have its performance significantly reduced when pipelining and parallel processing is not possible. This is no longer the case in DPAA.
5. Power Consumption
   The proposed architecture is based on parallel computing and pipelining while the clock frequency is being reduced. This would result in reduction of power consumption [11]. In addition, the interconnection interface is designed to reduce the power dissipation of the buses, as voltage amplitude of the modulated signal for each transmitter is reduced to a small swing voltage. However, applying the same technique to other bus-style such as Sonics (TDMA) style bus, is a difficult task because it will reduced SN ratio significantly and cause timing problems.
6. Identical Timing Requirement
   In the state-of-arts array processors which utilized tri-state bus, TDMA style bus or switching matrix, timing requirements for each signal path (a processor to another processor) varies, depending on the location of the two processors. In contrast, due to the bus interface in DPAA, timing requirements for transmitting signals from a processor to other processor is the same, regardless of their position in the chip.

Many novel architectures especially those of data-flow architecture, are difficult to program. However, program for DPAA can be easily generated using the

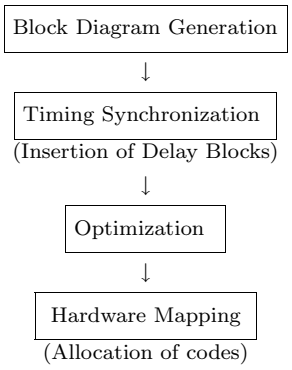


Fig. 8 Program algorithm for CAD.

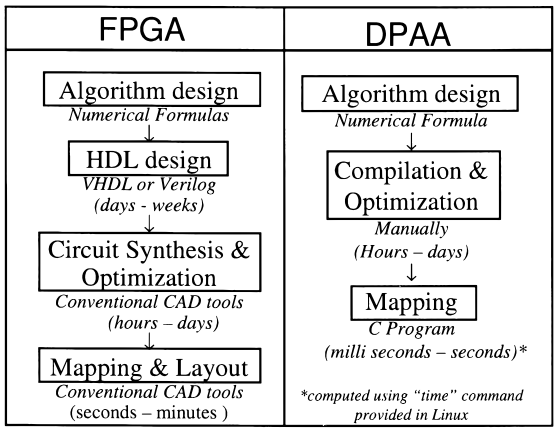


Fig. 9 Comparison of design flow: DPAA Vs FPGA.

developed Auto-Program Generation Interface which will be discussed in the next section.

5. Auto-Program Generation Interface

An auto-program generation interface had been developed to configure the proposed DPAA using simple numerical formula as input. The program algorithm will be discussed and evaluated.

5.1 Program Algorithm

In DPAA, each transmitter is assigned to a fixed PN sequence code. In order to establish a connection between a receiver and a transmitter, the receiver should be configured to use the same PN sequence code as used in the transmitter. The codes used in transmitter are stored in ROM while the codes for the receivers are stored in cache memories. The program required to configure DPAA consists of the receiver codes and can be generated manually with the algorithm shown in Fig. 8. Firstly, the application will be input in the form of block diagrams or text equations. Next, delay blocks are inserted for synchronization purposes. Before mapping the application into the hardware, optimization is carried out to remove excessive hardware. Auto-program generation interface using numerical formula as input has been developed. However, timing justification and optimization is still done manually. Compared to CAD used in fine granularity reconfigurable architecture such as FPGA, the auto-program generation used in DPAA is much easier and faster. For FPGA, every signal path had to be computed by taking into consideration timing and electrical property while auto-program generation interface developed for DPAA merely mapped the PN sequence codes.

5.2 Design Flow Comparison: DPAA vs. FPGA

Comparison of the design flow is shown in Fig. 9. The approximated design time required for each stage is given (written in bracket) by using several conventional circuits such as 5 tap FIR filters, IIR filters, DCT etc.

Any design usually starts with algorithm design that can be expressed with block diagrams, flow chart and numerical formulas. For FPGA designs, the algorithm will be converted into hardware description languages (HDL) such as VHDL or Verilog. Then HDL design will be followed by circuits synthesis and hardware mapping using conventional CAD tools. Depending on the design, these processes can take up to a few months or at least a few weeks. However, DPAA which utilized numerical formulas and block diagrams as input, is more user friendly to both software and hardware engineers. As the design duration is much shorter, DPAA poses very low design cost.

6. Example: CMA Adaptive Array

For better understanding of the proposed work, an example using CMA adaptive array will be discussed. The algorithm used is based on [12], [13]. Usually, CMA Adaptive Array consists of n-tap complex number FIR filter with the coefficient in each tap (or known as "weight" in CMA adaptive array) updated adaptively. The block diagram is shown in Fig. 10 and the details of CMA adaptive array is given as followed. Define an n-dimensional received signal vector $H(k)$ and weight vector $C(k)$ at a sampling time $kTs(k = 1, 2...)$ where Ts is a sampling period, as

$$\mathbf{H}(k) = \begin{bmatrix} h_1(k) \\ h_2(k) \\ h_3(k) \\ \vdots \end{bmatrix}, \quad \mathbf{C}(k) = \begin{bmatrix} c_1(k) \\ c_2(k) \\ c_3(k) \\ \vdots \end{bmatrix} \tag{1}$$

Both $H(k)$ and $C(k)$ are represented by complex vectors. Calculation for the CMA adaptive array is as shown in the following equations [12]:

$$\mathbf{OUT}(k) = \mathbf{H^T}(k)\mathbf{C}(k) \tag{2}$$

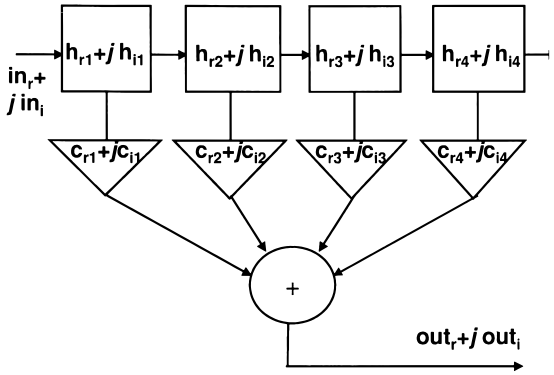


Fig. 10 CMA adaptive array block diagram.

After calculation of each data, the coefficient will be regenerated with the following equations. Where mu is a coefficient.

$$\epsilon(k) = \mathbf{OUT}(k).(|\mathbf{OUT}(k)|^2 - 1) \tag{3}$$

$$\mathbf{C}(k+1) = \mathbf{C}(k) - \mu\epsilon(k)\mathbf{H}^*(k) \tag{4}$$

The CMA adaptive array had been implemented using the proposed CAD tools for DPAA. For simplicity, the example used consists of 4-tap FIR. As all calculations listed above is done one after another, CMA adaptive array works in three different phases, i.e. Eq. (2) to generate output, (3) and (4) for coefficient regeneration. Therefore, a program input for each phase is written based on the numerical formulas given in Eqs. (2)–(4). Note that dynamically programmability of DPAA is utilized as the chip is reconfigured every cycle to carry out three different programs. Calculations are done in such a way that maximum parallelism and pipelining are exploited. All possible parallel tasks are carried out concurrently in those operating pipelines. In the above example, the pipeline depth is two while the throughput is 1 data frame per 11 cycles.

Program input for Eq. (2) which processes two taps of the CMA adaptive array is shown in Fig. 11. The program is compiled using the auto-program generation interface. The proposed CAD interface reads and examines the syntax of the input file, and then maps the numerical formula into available hardware. Information needed to set up all LE is also compiled. As expected, the mapping of DPAA is fast because each phase requires less than 0.1 ms on a 450 MHz Pentium III machine. This is much faster compare to FPGA designs because the mapping of synthesized netlist on a FPGA chip itself requires a few seconds. Note that in order to synthesize a netlist for the above applications, a designer will take at least a few weeks to write the above applications into HDL language, verified and compiled with expensive CAD tools.

```
Phase 1
========
Input:
inr
ini

Operation:
hr1 = inr; hi1=ini; //shift register
hr2 = hr1; hi2=hi1;
hr3 = hr2; hi3=hi2;
hr4 = hr3; hi4=hi3;

ta1=hr1*cr1; ta2=hr1*ci1; // complex number
ta3=hi1*cr1; ta4=hi1*ci1; // multiplication
tb1=hr2*cr2; tb2=hr2*ci2;
tb3=hi2*cr2; tb4=hi2*ci2;
or1=ta1 - ta4; oi1=ta2 + ta3;
or2=tb1 - tb4; oi2=tb2 + tb3;

totalr = or1 + or2; totali = oi1 + oi2; // addition
outr=totalr + outr; outi=totali + outi;

output:
outr
outi // output
```

Fig. 11 Program input for Eq. (2).

7. Conclusions

DPAA, a new architecture for DSP, and its VLSI implementation has been described. Programming interface for the proposed architecture has been developed and it is shown that the new architecture will reduce development cost significantly. DPAA has fulfilled its design target as a new architecture which provides high parallelism, high pipelining ability and high area efficiency with simple programming interface.

As this work is the first generation of its kind, there is much room for improvement such as speed, optimization of program for dynamical reconfigurable application, etc. However, the proposed DPAA is promising with its dynamical programmability, complete routing flexibility and high chip utilization. DPAA, with its large number of processing elements and complete routing flexibility, can be utilized to explore parallel computing. By incorporating various functions into DPAA, we believe that DPAA can be a good mean to implement System-On-A-Chip (SoC) design.

For more advanced process technology, miniaturization of device benefits DPAA because it will increase the speed of DPAA. However, with reduced power supply voltage, designing high precision circuits in the multiple access bus interface will become difficult. In order to utilize all available transmission channels, more arithmetical functions should be incorporated into each logic elements instead of one as proposed in DPAA.

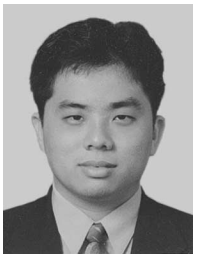
Acknowledgement

The VLSI chip in this study has been fabricated in the chip fabrication program of VLSI Design and Education Center (VDEC), the Univ. of Tokyo with the collaboration by Rohm Corporation and Toppan Printing

Corporation. This work is supported by Japan Society for the Promotion of Science (JSPS) Research for Future Program.

References

[1] M. Dolle, S. Jhand, W. Lehner, O. Muller, and M. Schlett, "A 32-b RISC/DSP microprocessor with reduced complexity," IEEE J. Solid-State Circuits, vol.32, no.7, pp.1056–1066, July 1997.
[2] R.B. Yates, N. Thacker, S.J. Evans, S.N. Walker, and P.A. Ivey "An array processor for general purpose digital image compression," IEEE J. Solid-State Circuits, vol.30, no.3, March 1995.
[3] D.C. Chen and J.M. Rabaey, "A reconfigurable multiprocessor IC for rapid prototyping of algorithm-specific high-speed DSP data paths," IEEE J. Solid-State Circuits, vol.27, no.12, pp.1895–1904, Dec. 1992.
[4] R.W. Hartenstien, R. Kress, and H. Reinig, "A reconfigurable data-driven ALU for Xputers," IEEE Workshop on FPGAs for Custom Computing Machine, FCCM'94, Napa, CA, April 1994.
[5] H. Ochi, "FPAccA: Field programmable accumulator array—Design and evaluation of FPAccA model 1.0 chip," Inf. Process., vol.40, no.4, pp.271–350, April 1999.
[6] J. Rose, R.J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency," IEEE J. Solid-State Circuits, vol.25, no.5, pp.1217–1225, 1990.
[7] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," IEEE J. Solid-State Circuits, vol.26, no.3, pp.277–282, 1991.
[8] R. Yoshimura, B.K. Tan, T. Ogawa, S. Hatanaka, T. Matsuoka, and K. Taniguchi, "DS-CDMA wired bus with simple interconnect topology for parallel processing systems LSIs," ISSCC Digest of Techinical Papers, pp.370–371, Feb. 2000.
[9] S. Glisic and B. Vucetic, Spread Spectrum CDMA systems for Wireless Communication, Artech House Publishers, 1997.
[10] M. Yamashina and H. Yamada, "An MOS current mode logic (MCML) circuit for low power sub-GHz processors," IEICE Trans. Electron., vol.E75-C, no.10, pp.1181–1187, Oct. 1992.
[11] A.P. Chandrakasan and R.W. Brodersen, Low Power Digital CMOS Designs, Chapter 4, Kluwer Academic Publishers, pp.105–139, 1995.
[12] J.R. Treichler and B.G. Agee, "A new approach to multipath correction of constant modulus signal," IEEE Trans. Acoutst., Speech & Signal Process, vol.ASSP-31, no.2, pp.459–472, April 1983.
[13] T. Ohgane, T. Shimura, N. Matsuzawa, and H. Sasaoka, "An implementation of a CMA adaptive array for high speed GMSK transmission in mobile communications," IEEE Trans. Veh. Technol., vol.42, no.3, pp.282–288, Aug. 1993.



Boon-Keat Tan was born in Penang, Malaysia in 1972. He received his M.S. degree from Osaka University, Osaka, Japan in 1999. He is now working towards his Ph.D. degree at the same university. His current research interests are reconfigurable processors, high performance digital circuits and analog circuits. He is a student member of the IEEE.



Ryuji Yoshimura was born in Nara, Japan in 1973. He received his B.S. and M.S. degrees from Osaka University, Osaka, Japan in 1996 and 1998 respectively. He is currently pursuing his studies toward the Ph.D. degree at the same university. His current research interests is Analog Circuits. He is a student member of the IEEE.



Toshimasa Matsuoka was born in Osaka, Japan in 1966. He received the B.S., M.S. and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1989, 1991 and 1996, respectively. During 1988–1991, he was involved in the research of heterostructures and superlattices of GaAs and related compounds. During 1991–1998, he worked for the Central Research Laboratories, Sharp Corporation, Nara, Japan, where he was engaged in the research and development of deep submicron CMOS devices and ultra thin gate oxides. Since 1999, he has been worked with Osaka University. His current research includes phase lock loops and CMOS RF circuits. Dr. Matsuoka is a member of the Japan Society of Applied Physics and the IEEE.



Kenji Taniguchi received the B.S., M.S. and Ph.D. degrees from Osaka University, Osaka, Japan, in 1971, 1973 and 1986 respectively. From 1973 to 1986, he worked for Toshiba Research and Development Center, Kawasaki, Japan where he was engaged in process modeling and the design of MOS LSI fabrication technology. He was a Visiting Scientist at Massachusetts Institute of Technology, Cambridge, from July 1982 to November 1983. Presently, he is a Professor of Electronics Engineering at Osaka University. His current research interests are in analog circuits, radio frequency circuits, device physics and process technology. Dr. Taniguchi is a member of the Japan Society of Applied Physics. He is a fellow of the IEEE.