전용 PLD를 가진 새로운 SoC 플랫폼
A New SoC Platform with an Application-Specific PLD

이재진*, 송기용*
Jae-Jin Lee*, Gi-Yong Song*

요 약

SoC는 소프트웨어와 하드웨어가 통합 설계되는 시스템 수준 설계 플랫폼이며 상위 수준 합성은 SoC 설계방법론의 중요한 과정이다. 최근 SPARK라 불리는 병렬 상위 수준 합성 툴이 개발되었다. SPARK는 C코드를 입력받아 코드 이동과 다양한 변형 기술을 이용해서 스케줄하고 최종적으로 합성 가능한 RTL VHDL를 생성한다. 기본 적인 디지털 신호 및 영상처리 알고리즘은 반복 순환문으로 표현되며, 합성을 통해 SPARK는 다양한 루프 변형 알고리즘을 적용한다. 그러나 이 기법에 의한 합성 결과는 디자이너가 수동으로 직접 설계한 최적구현과 비교했을 때 성능 면에서 만족할 만한 결과를 생성하지 못한다. 본 논문에서는 전용 프로그램 논리소자를 가지는 새로운 SoC 플랫폼을 제안하고, C로 기술된 행위 수준 반복 순환문을 2차원 시스톨릭 어레이로 매핑하는 과정을 기술한다. 최종적으로 유도된 시스톨릭 어레이는 제안된 SoC 플랫폼 상의 전용 프로그램 논리소자 상에 구현된다.

Abstract

SoC which deploys software modules as well as hardware IPs on a single chip is a major revolution taking place in the implementation of a system design, and high-level synthesis is an important process of SoC design methodology. Recently, SPARK parallelizing high-level synthesis software tool has been developed. It takes a behavioral ANSI-C code as an input, schedules it using code motion and various code transformations, and then finally generates synthesizable RTL VHDL code. Although SPARK employs various loop transformation algorithms, the synthesis results generated by SPARK are not acceptable for basic signal and image processing algorithms with nested loop. In this paper we propose a SoC platform with an application-specific PLD targeting local operations which are feature of many loop algorithms used in signal and image processing, and demonstrate design process which maps behavioral specification with nested loops written in a high-level language (ANSI-C) onto 2D systolic array. Finally the derived systolic array is implemented on the proposed application-specific PLD of SoC platform.

Keywords : System-level design, high-level synthesis, application-specific PLD, super-systolic array.

I. Introduction

The input to system-level design methodology is the specification of an application in a high-level language and the output is an implementation of the application on a SoC platform. System-level design consists of an allocation of system components such as typical microprocessor, memories, buses and hardware resources, and a partitioning of functionalities among those components. The focus of this paper is an implementation of an application on the hardware component using high-level synthesis, followed by logic synthesis and place and route. High-level synthesis [1][2][3] has received significant attention over the past decade. Recently, SPARK parallelizing HLS(high-level synthesis) software tool [3][4] has been developed. It is a high-level synthesis methodology that incorporates parallelizing compiler and compiler transformation into a traditional high-level synthesis framework during pre-synthesis and scheduling phases. It takes an input of specification in the C high-level language and produces an output of synthesizable register-transfer level (RTL) VHDL code. To handle loop algorithm, SPARK employs various transformations such as loop-invariant code motion, loop unrolling, loop index variable elimination and loop shifting. In practice, however, SPARK does not produce circuit description whose quality can compete with manual designs in term of circuit complexity and

*충북대학교 전기전자컴퓨터공학부
논문 번호 : 2007-4-4 접수 일자 : 2007. 8. 20
심사 완료 : 2007. 10. 20
This work was supported by the research grant of the Chungbuk National University in 2006.

execution time. Most commercial FPGAs cannot handle systolic structure with fast sampling rate for their general purpose nature. This paper proposes a super-systolic array-based application specific PLD(programmable logic device) which combines the high performance of ASICs with the flexibility of PLDs.

Our work has been motivated by an enhancement of SPARK through synthesizing a loop construct into a 2D systolic array and a proposal of a new application-specific PLD architecture targeting systolic design.

II. System-Level Hardware Design

In a system-level design, uniform specification of systems, hardware/software partitioning, hardware synthesis, software compilation and co-simulation have become important research area. System-level design consists of an allocation of system components such as typical microprocessor, memories, buses and hardware resources, and a partitioning of functionalities among those components. Our system-level design methodology containing an application-specific PLD, which is a slight modification of a typical system-level design methodology, is shown in Fig.1

PLDs have progressed through a long evolution to reach the complexity today to support an entire SoC and they are making inroads against ASICs along the development time required and the level of investment risk. Manufacturing technology advances have also allowed PLDs to close significantly the price premium gap with ASICs. Today, PLDs have become a critical part of every system design. However, most commercial PLD such as FPGAs and CPLDs cannot handle designs with very fast sampling rate for their general-purpose architecture nature. We adopt an application-specific PLD which combines high performance of ASICs and flexibility of PLDs in a new SoC platform as shown in Fig. 1.

In general, the input to the system is the behavioral specification of an application written in a high-level language such as C, C++, SystemC, SpecC and System-Verilog, and the output is an implementation of the application on a SoC platform. In our approach, the input to high-level synthesis is a behavioral description written in ANSI-C as in SPARK.

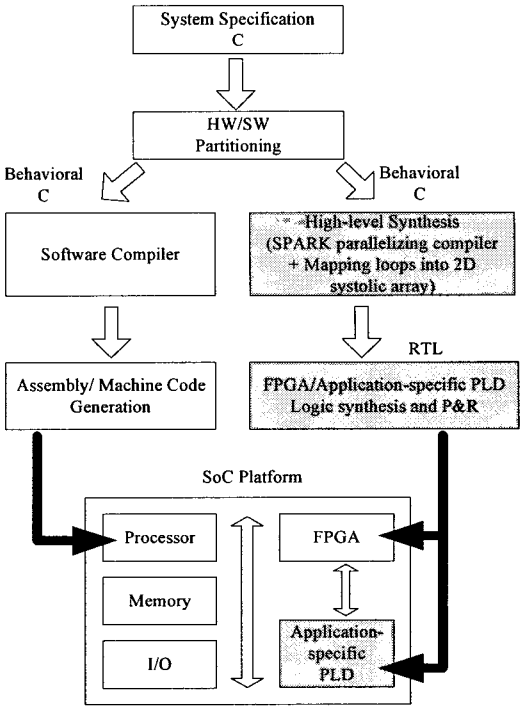

Fig. 1. System-level design methodology.

An examination of the system specification is needed to determine a hardware/software partitioning that meets the timing, hardware, performance, and power constraints required by the applications. After hardware/software partitioning is performed as shown in Fig. 1, the software portion is compiled using software compiler for the processor core while the hardware portion is synthesized using SPARK and loop transformation methodology proposed in this paper, followed by logic synthesis and P&R. This paper is only focused on high-level synthesis of loop construct during system-level design process.

During the high-level synthesis, our approach attempts to transform only the nested loop without a complex mix of conditional (if-then-else) constructs inside into 2D systolic array if the number of operations mapped to a resource type in any clock cycle does not exceed the upper bound of that resource type, and generate RTL VHDL code for the derived systolic arrays. Other portion of input specifications except nested loops is synthesized using SPARK and implemented on general FPGA.

This idea is motivated by the fact that synthesis result generated by SPARK is not acceptable for designs with nested loops such as matrix-matrix multiplication and FIR filter, IIR filter, 2D convolution in terms of circuit complexity and execution time. As a

consequence, we consider synthesizing a nested loop into a 2D systolic array to improve the synthesis result by incarnating the parallelism available in the algorithm description, and then finally implement the derived systolic array on the application-specific PLD proposed in this paper.

III. SPARK

Recently, SPARK parallelizing high-level synthesis software tool has been developed in [3]. The SPARK accepts a behavioral description of a design in ANSI-C, creates the intermediate representation, runs a data dependency analysis pass, schedules the design, binds the resources, performs control synthesis, and finally generates an output in RTL VHDL.

The SPARK first applies a set of coarse-grain and fine-grain code transformations to the input description during a pre-synthesis phase before performing the traditional high-level synthesis tasks of scheduling, allocation and binding. The transformations in the pre-synthesis phase include (a) coarse-level code restructuring by function inlining and loop transformations (loop unrolling, loop fusion, etc.) (b) transformations that remove unnecessary and redundant operations such as common sub-expression elimination (CSE), copy propagation, and dead code elimination and (c) transformations such as loop-invariant code motion.

The pre-synthesis phase is followed by the scheduling and allocation phase. The transformations toolbox contains speculative code motion transformations, the percolation and trailblazing code motion techniques and dynamic renaming of variables, etc. Besides the traditional high-level synthesis transformations such as dynamic CSE and dynamic copy propagation, branch balancing technique also dynamically adds scheduling steps in conditional branches to enable code motions.

The presence of nested loops in application codes limits the scope of parallelizing code motion transformations to within loop iterations. To achieve the performance improvement, SPARK uses a technique called loop shifting that incrementally exploits loop level parallelism across loop transformation.

Although SPARK employs various loop transformations such as loop-invariant code motion, loop unrolling, loop index variable elimination and loop shifting, the synthesis results generated by SPARK is not acceptable for basic signal and image processing algorithms with nested loops such as matrix-matrix multiplication and FIR filter. Accordingly, we consider a transformation mapping a nested loop onto a 2D systolic array to improve the synthesis result by incarnating as much parallelism as can be exploited with the available resource

IV. Mapping Loops onto 2D Systolic Array

This approach is performed during high-level synthesis only if the number of operations mapped to a resource type in any clock cycle does not exceed the upper bound of that resource type. The final output of our approach is the generation of RTL VHDL code for derived systolic array.

A systolic array [5][6] formed by interconnecting a set of identical data-processing cells in a uniform manner is a combination of an algorithm and a circuit that implements it, and is closely related conceptually to arithmetic pipeline. The underlying principle of systolic array is to achieve massive parallelism with a minimum communication overhead, and generally speaking, a systolic array is easy to implement because of its regularity and easy to reconfigure because of its modularity. Most algorithms of signal processing and other engineering application require the use of a massively parallel computing structure, that is, systolic array structure, to achieve acceptable performance.

Methodology mapping nested loop algorithms into 2D systolic array has been proposed in [7] and we briefly summarize the loop transforming algorithm in [7] for readers' convenience.

Mapping policy in this paper guarantees that neighboring iterations of the N-dimensional $(1 \leq N \leq 3)$ index space will be assigned to neighboring cells of the $(N-1)$-dimensional target systolic array.

Localized operations, intensive computation, and matrix operations are features of many loop algorithms used in signal and image processing. In this paper, we assume that the computational structure consists of a 2D mesh-connected systolic array. Each cell in a 2D systolic array can be indexed as 2-tuple in 2 coordinates as shown in Fig. 2. We show only 4 by 4 cells for the purpose of simplicity.

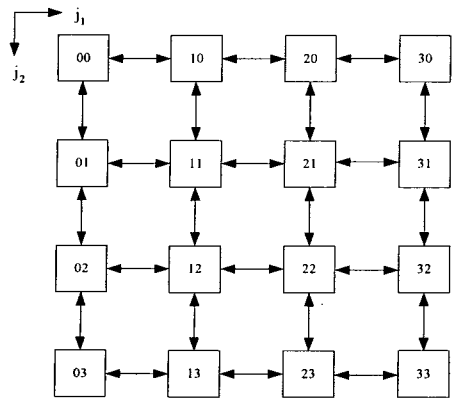

Fig. 2. Structure of 2D systolic array.

In this structure, each cell can only be connected to its neighbor. There are at most 5 communication connections, 4 bidirectional connections to nearest neighbors and one connection within the cell making self-loop. The interconnections between cells are described by the difference vectors between the coordinates of adjacent cells, and can be represented by a matrix of inter-connection as follow.

$$P=[p_1\ p_2\ p_3\ p_4\ p_5]=\begin{bmatrix}0&0&1&0&-1\\0&1&0&-1&0\end{bmatrix} \qquad (1)$$

The structural details of the cells, I/O, execution time and communication are determined after mapping loops onto systolic array processor. Throughout this paper, we deal with algorithms that are regular in terms of the computational patterns, and focus on algorithms which have the form of the nested FOR-loop structure with uniform data dependencies.

Two steps are involved in mapping a loop algorithm onto a systolic array [5]. The first step is a scheduling. Once the scheduling is fixed, the second step is process assignment. Scheduling specifies the sequence of operation in all the cells. A schedule function represents a mapping from the N-dimensional index space onto a 1D schedule (time) space. A linear schedule is based on a set of parallel and uniformly spaced hyperplanes called equitemporial hyperplane. All the nodes on the same hyperplane must be processed at the same time. Mathematically, the schedule can be represented by a schedule vector S, pointing to the normal direction of the hyperplane. Processor assignment maps each node of a dependency graph(DG) onto a systolic array cell. It is common to use linear projection for processor assignment. Mathematically, linear projection is often represented by projection vector N.

To map loops onto a systolic array mathematically, we search for the transformation matrix defined below.

$$M=\begin{bmatrix}S\\R\end{bmatrix}=\begin{bmatrix}t_{11}&t_{12}&t_{13}\\t_{21}&t_{22}&t_{23}\\t_{31}&t_{32}&t_{33}\end{bmatrix} \qquad (2)$$

Where mapping S and R are defined as $S: J^n \rightarrow J^1$ and $R: J^n \rightarrow J^{n-1}$. M is n by n matrix since we consider only linear transform in this paper. Algorithm dependencies D are transformed into $D' = MD$. The mapping S is selected such that the transformed data dependencies matrix D' has positive entry in the first row. This means that a causality should be enforced in a permissible schedule.

After mapping loop algorithms onto systolic array, a backend code generation pass outputs RTL VHDL code which is synthesizable using a logic synthesizer such as Synopsys Design Compiler [8, 9]. The length of the register is easily identified from algorithm code. The computational elements such as multiplier and adder are selected from a set of computation C, and are designed using component instantiation of Synopsys Design Ware. Timing and data communication captured by transformed data dependencies give crucial information on generating VHDL code for datapath of each cell and intercommunication between cells. The generated VHDL output is a structural description, so it determines a fully specified systolic architecture through the proposed mapping method and leaves no freedom to the CAD tool to reconstruct it.

V. Application-Specific PLD

In a systolic array, to achieve even higher degree of concurrency, it is desirable to make cells of systolic array themselves systolic array as well. This systolization procedure of implementing cell as another systolic array could be applied repeatitively in a hierarchical manner until a cell having only primitive operators is obtained. We will refer to the systolic array resulted from the above strategy as super-systolic array [10].

After loop construct is mapped onto 2D systolic array, the derived systolic array is transformed into super-systolic array, and then finally implemented on the application-specific PLD proposed in this paper.

This architecture combines the high performance of ASICs with the flexibility of PLDs and it offers a significant alternative view on the programmable logic devices. The super-systolic array is ideal for a newly proposed PLD architecture when it comes to area-efficiency, P&R and clock speed.

In this section, we describe architecture of the application-specific PLD. A PLD architecture targeting a super-systolic array for arithmetic operations is shown in Fig. 3.

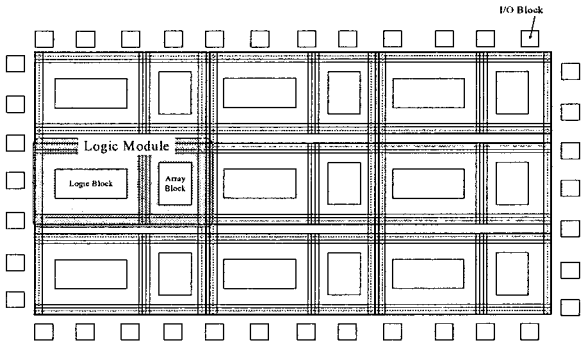

Fig. 3. Application-specific PLD architecture.

As shown in Fig. 3, the architecture consists of configurable Logic Modules(LMs), configurable I/O Blocks, and programmable interconnections to route signals. The number of LMs in a real chip would be determined on considering the design constraints such as chip size, target applications, cost, etc.

5.1 Logic Module

Each LM is made up of a Logic Block(LB) which consists of five Logic Units(LUs) and an Array Block(AB) as shown in Fig. 4

The LU and AB will be explained in detail in the subsequent sections. There are three kinds of programmable interconnections: global interconnection, inter-LM connection and intra-LM connection. The overall need for routing can be classified into three categories, routing in each category being mapped to one of three programmable interconnections: global, inter-LM or intra-LM. The routing is to be performed in a hierarchical manner on the proposed PLD. Fig. 4 shows one LM including all of its components with detailed programmable interconnections. The routing channel, or programmable interconnections such as global, inter-LM and intra-LM, can be added to accommodate the demand for signal propagations of the target application. The proposed architecture works on circuits with feedback and bi-directional dataflow to satisfy the nature of systolic array. Global interconnection represented by bold line is used to route global signals such as power, clock and primary inputs/outputs.

5.2 Logic Unit(LU) and PSFG

The fundamental logic cell in the proposed PLD is based on PSFG(programmable symmetric function generator) and builds systolic array efficiently while that of existing FPGAs such as Xilinx is based on multiplexer and look-up table without any preference for computational structure. The 3-variable PSFG consists of two planes as shown in Fig. 5.

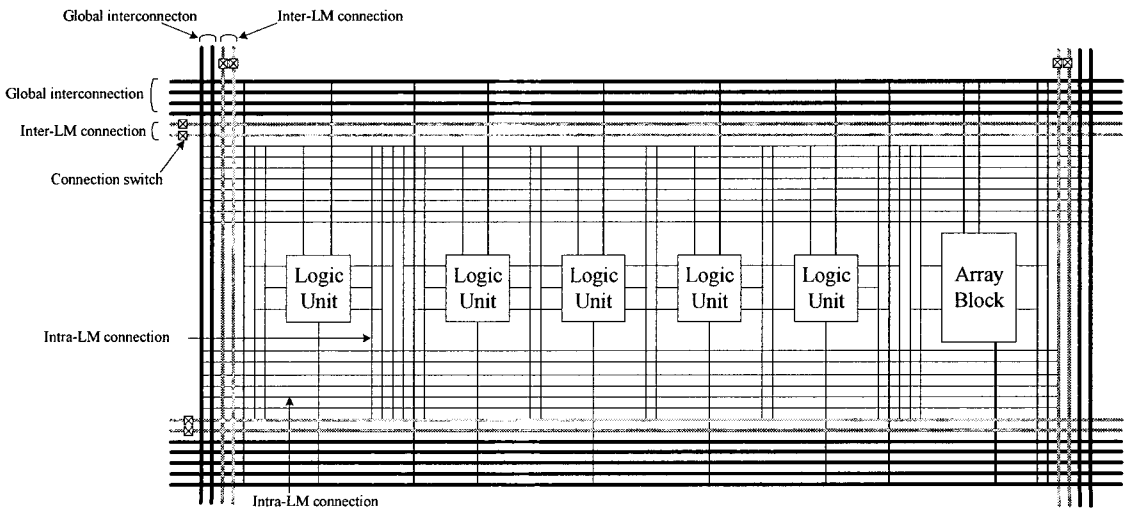

Fig. 4. Layout of a Logic Module

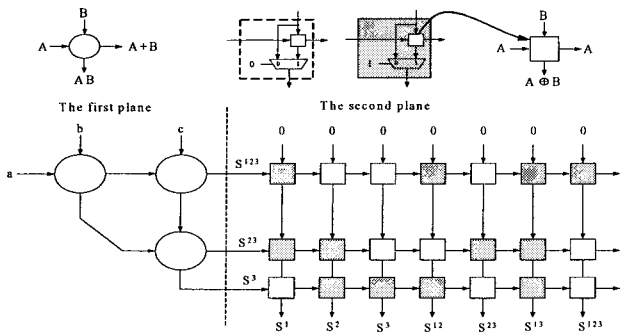

Fig. 5. 3-Variable PSFG.

The triangular structure of the first plane is planar and regular. It realizes all positive unate 3-variable symmetric functions. A function is unate if it can be represented by a disjunctive or conjunctive expression in which no variable appears in both its complementend and uncomplemented forms. Horizontal outputs from the first plane are EXOR-ed in the second plane to generate arbitrary symmetric functions at the bottom. The control signals on the second plane can take a value of '0' or '1'. The programmability on the second plane is obtained by connecting or disconnecting rectangular shaped EXOR gates in each column to achieve certain symmetric functions. As shown in Fig. 6, arbitrary 2-variable symmetric functions such as NOR, XOR, AND, OR and NAND can efficiently be realized using 3-variable PSFG by setting one of inputs to 0.

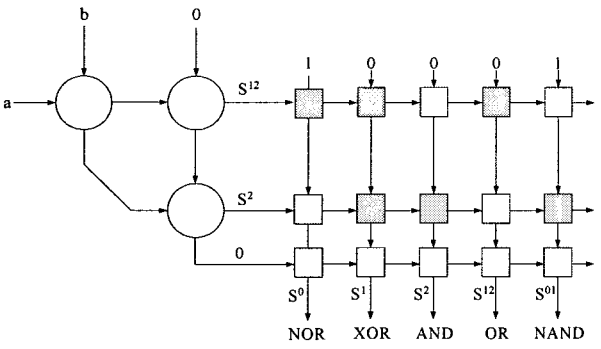

Fig. 6. Realization of 2-variable symmetric functions.

The LU based on 3-variable PSFG is shown in Fig. 7. It contains three flip-flops for storage elements, three multiplexer to route the logic outputs to external resources and three 2-input AND gates for input control of PSFG. The LU in Fig. 7 is programmed as a systolic multiplier [10]. In Fig. 8, one PSFG slice implements both sum and carry of a 1-bit full adder. Symmetric function with more than three inputs can be implemented as a cascade of 3-variable PSFG through EXOR-based Davio expansion [11].

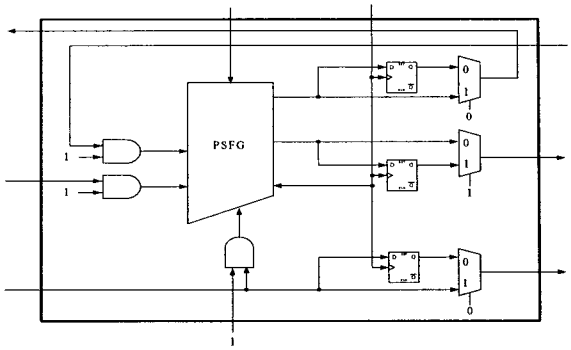

Fig. 7. Structure of a Logic Unit.

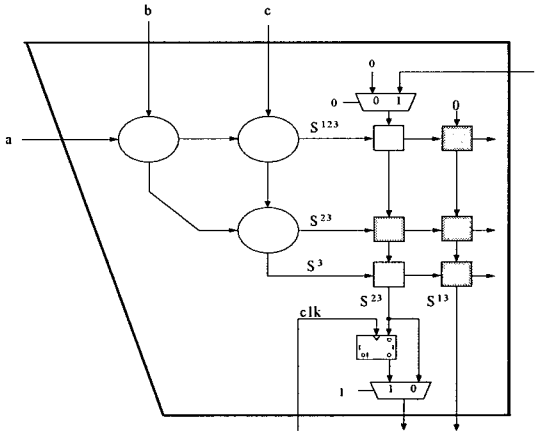

Fig. 8. Structure of a PSFG.

5.3 Array Block(AB)

The AB consisting of two 32-bit shift registers and two multiplexers as is shown in Fig. 9 is introduced to synchronize the bit dataflow between LBs to guarantee the generation and summation of the intermediate results according to recurrence relation of the behavior to be implemented on the proposed PLD.

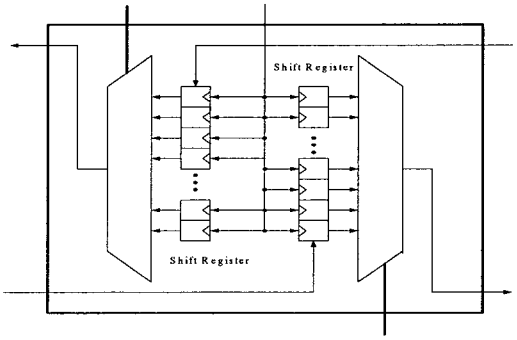

Fig. 9. Structure of an Array Block

VI. Experimental Results and Performance Evaluation.

To compare the results from transforming loops into a 2D systolic array mentioned in this paper and those from SPARK, the VHDL codes for matrix-matrix multiplication and FIR filter from each case are synthesized using Synopsys Design Compiler based on Hynix 0.35㎛ cell library. Table 1 and Table 2 list the synthesis results of IIR filter and 2D convolution, respectively. As shown in Table 1 and Table 2 the proposed C-to-VHDL loop synthesis achieves synthesis results that are better than those achieved from a current version of SPARK. Compared to SPARK, our approach achieves up to 73% reduction in the hardware complexity and up to 55% speed up in the execution time for two test designs.

To show the superiority of the PLD architecture proposed in this paper to the existing FPGA architecture, super-systolic IIR filter and super-systolic array for 2D convolution are implemented onto a Xilinx FPGA. Compared to the existing FPGA architecture whose routing process uses some of SLICEs to complete the whole connectivity, it is estimated that the proposed PLD, which does not require extra LUs to complete routing process, is more efficient in terms of hardware complexity. Table 3 and Table 4 show that our proposed architecture does much better in speed for three designs under test. Note that we have assumed that the proposed PLD adopts the same technology as Xilinx XCV200 [12], applying the same data to the PLD.

The proposed PLD architecture can easily implement basic arithmetic operations and signal processing algorithms such as ripple carry adder, bit-serial multiplier, matrix-matrix multiplication, FIR filter, IIR filter, 2D convolution, and FFT because of the property that the recurrence relation of such an algorithm is efficiently mapped onto a systolic structure. Our experimental results are not meant to be a justification for the architecture in terms of physical data. Even though we provide only simulation results without actual fabrication of the proposed PLD chip, it is interesting to see that the proposed PLD can achieve implementation results that are better than those achieved on existing FPGAs for such designs that are highly regular like arithmetic circuits.

Table 1. Synthesis of IIR filter

Description	SPARK	The proposed approach
The number of coefficient	4	4
Length of input and coefficient (bit)	4	4
Total cell area (2-input NAND)	6843	986
Worst clock cycle(ns)	24.04	12.23

Table 2. Synthesis of 2D convolution

Description	SPARK	The proposed approach
Mask size	2 by 2	2 by 2
Length of input and coefficient (bit)	4	4
Total cell area (2-input NAND)	12792	3460
Worst clock cycle(ns)	32.08	14.35

Table 3. Implementation of super-systolic IIR filter

Description	FPGA (XCV200)	The proposed PLD
The number of coefficient	4	4
Length of input and coefficient (bit)	4	4
Hardware complexity	72 SLICEs	45LU
Total P&R time(sec)	2	1
Worst clock cycle(ns)	7.432	1.25

Table 4. Implementation of super-systolic array for 2D convolution

Description	FPGA (XCV200)	The proposed PLD
Mask size	2 by 2	2 by 2
Length of input and coefficient (bit)	4	4
Hardware complexity	151 SLICEs	92LU
Total P&R time(sec)	3	1
Worst clock cycle(ns)	12.132	1.25

VII. Conclusions

This paper proposed a new system-on-a-chip platform with an application-specific PLD targeting local operations involving loop algorithms used in signal and image processing. Also, this paper demonstrated process which maps behavioral specification with nested loops of an application written in a high-level language onto the 2D systolic array. The derived systolic array is transformed into super-systolic array and finally implemented on the proposed application-specific PLD.

Being compared to the nested loop synthesis of SPARK, the loop synthesis approach in this paper results in about 70% reduction in both area and execution time, and implementation of the derived systolic array on application-specific PLD can achieve implementation results that are better than those achieved on existing FPGAs for such a design that is highly regular like MAC operations.

References

[1] D.D.Gajski, N.D.Dutt, A.C-H.Wu, and S.Y-L.Lin, High-Level Synthesis: *Introduction to Chip and System Design*, Kluwer Academic, 1992.

[2] G.De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[3] S.Gupta, R.K.Gupta, N.D.Dutt and A.Nicolau, *SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits*, Kluwer Academic, 2004.

[4] S.Gupta, *User Mannual for the SPARK Parallelizing High-Level Synthesis Framework version 1.1*, http://mesl.ucsd. edu/spark, 2004.

[5] S.Y.Kung, *VLSI Array Processors*, Prentice Hall, 1988.

[6] H.T.Kung, "Why Systolic Architectures?," *IEEE Computers*, vol.15, no.1, pp.37-46, January 1982.

[7] J.J.Lee and G.Y.Song "High-level Synthesis Using SPARK and Systolic Array," *Lecture Notes in Computer Science (Reconfigurable Computing: Architectures and Application, ARC 2006)*, vol. 3985, pp.455-460, 2006.

[8] K.C.Chang, *Digital Systems Design with VHDL and Synthesis*, Computer Society, 1999.

[9] http://www.synopsys.com.

10] J.J.Lee and G.Y.Song, "Implementation of the Super Systolic Array for Convolution," *ASP-DAC 2003*, pp. 491-494, 2003.

[11] T.Sasao, "EXMIN2 : A Simplification Algorithm for exclusive OR sum of products expressions for multi-valued input two-valued output functions," *IEEE Trans. CAD.*, vol. 12, no. 5, pp.621-632, May 1993.

[12] *Xilinx, Virtex 2.5V Field Programmable Gate Arrays*, http://www.xilinx.com.

Jae-Jin Lee received the B.S., M.S., and Ph.D. degrees in computer engineering from Chungbuk National University in 2000, 2003, and 2006, respectively.

His research interests include the areas of VLSI design, design automation and computer architecture.

Gi-Yong Song received the B.S. and M.S. degrees from the department of industrial education and department of electronic engineering, Seoul National University in 1978 and 1980, respectively and the Ph.D. degree from University of Louisiana in 1995.

He is currently a professor in theSchool of Electrical and Computer Engineering.

His research interests include the areas of VLSI design, design automation, and computer architecture.