Power-Conscious Configuration Cache Structure and Code Mapping for Coarse-Grained Reconfigurable Architecture

Yoonjin Kim[†] , Ilhyun Park[†] , Kiyoung Choi[†] , Yunheung Paek[‡]

[†] Design Automation Laboratory, [‡] Software Optimization & Restructuring Laboratory
School of EECS, Seoul National University, Seoul, South Korea

{awgsize1, pih102, kchoi, ypaek}@snu.ac.kr

ABSTRACT

Coarse-grained reconfigurable architecture aims to achieve both performance and flexibility. However, power consumption is no less important for the reconfigurable architecture to be used as a competitive processing core in embedded systems. In this paper, we show how power is consumed in a typical coarse-grained reconfigurable architecture. Based on the power breakdown data, we suggest a power-conscious configuration cache structure and code mapping technique, which reduce power consumption without performance degradation. Experimental results show that the proposed approach saves much power even with reduced configuration cache size.

Categories and Subject Descriptors

C.1.3[**Processor Architecture**]: Other Architecture Styles – *Adaptive architectures, heterogeneous systems, pipeline processors.*

General Terms

Design, Performance, Experimentation, Verification.

Keywords

System-on-Chip (SoC), Low Power, Coarse-Grained Reconfigurable Architecture (CGRA), Configuration Cache, Loop Pipelining, Context Pipelining, Temporal Mapping, Spatial Mapping.

1. INTRODUCTION

With the growing demand for high quality multimedia, especially over portable media, efficient algorithms for audio and/or video data transfer and processing have been developed. These algorithms have the characteristics of data-intensive computation of high complexity. For such applications, we can consider two extreme approaches to implementation: software running on a general purpose processor and hardware in the form of ASIC. In the case of general purpose processor, it is flexible enough to support various applications but may not provide sufficient performance to cope with the complexity of the applications. In the case of ASIC, we can optimize best in terms of power and performance but only for a specific application. With a coarse-grained reconfigurable architecture (CGRA), we can take advantage of the two approaches. This architecture has higher performance level than general purpose processor and wider applicability than ASIC.

A typical CGRA consists of an array of processing elements (PEs) and configuration cache memory. Each word (also called context word) of the configuration cache determines the functionality of a processing element and the interconnections to other PEs. By loading the words from the configuration cache into the array, we can dynamically change the configuration of the entire array within just one cycle. If the cache is not big enough to hold all contexts required for an application, then we may have to fetch them from the main memory. Therefore configuration cache size and structure can affect performance and flexibility of CGRA. They also affect power consumption since frequent reconfiguration of CGRA causes many cache-read operations and in turn causes much power consumption. Therefore configuration cache design is important in the trade-off among performance, flexibility, and power.

In this paper, we suggest a novel power-conscious configuration cache structure called hybrid configuration cache, which consists of spatial cache and temporal cache. Spatial cache has a few layers of memory to store contexts for spatial mapping. Each PE has its own spatial cache. Temporal cache has many layers for temporal mapping but only on one column. Other columns are configured through context pipelining. By combining spatial mapping and temporal mapping properly on this structure, we can save much power while keeping the performance.

This paper is organized as follows. After mentioning the related work in Section 2, we describe our reconfigurable architecture and power-breakdown in Section 3. In Section 4, we propose a power-conscious configuration cache structure and code mapping. We show the experimental results in Section 5 and conclude in the last section.

2. RELATED WORK

There have been many researches on CGRA as summarized in [1]. Most of the researches have been carried out in three different aspects: architecture exploration, code compilation and mapping, physical implementation. Power/energy consumption is yet another aspect of CGRA research.

Various architecture exploration flows have been suggested [2][3][4] and all these flows generate a good instance of CGRA considering area cost and performance. In the aspect of energy, interconnect architecture explorations have been suggested for low energy [5][6]. Because CGRA has complex and heavy inter-

ISLPED'06, October 4–6, 2006, Tegernsee, Germany.

connection for performance and flexibility, power consumption of interconnection is crucial. In [5] the authors have proposed energy-aware interconnection exploration that aims to minimize energy by changing the topology between global register file and function units. However, this exploration only provides the trade-off between performance and energy. In [6] the authors have suggested hierarchical generalized mesh structure exploration that continues to exploit locality while reducing the cost of long connections but it has been only evaluated for specific reconfigurable DSPs. In the case of code compilation and mapping, loops have been focused on mainly for performance [7][8][9]. To the best of our knowledge, however, energy/power aware mapping has not been studied yet.

Many reconfigurable architectures have been implemented physically with various technologies [10][11][12][13][14][15]. Most of the researches in this aspect have focused on efficient design with respect to small area and high performance. In [11][12][15], even though authors have presented power estimation data of the implemented architectures, these are only accessorial results and don't mean power/energy-aware implementation. In [16][17], authors have emphasized that the implemented architectures are power-efficient compared with fine-grained architecture such as FPGA in specific applications but the architectures are not general CGRA but specific types for running some applications with low power.

3. PRELIMINARIES

3.1 Coarse-Grained Reconfigurable Architecture

A typical coarse-grained reconfigurable architecture consists of a microprocessor, a reconfigurable array, and their interface. There are three ways of connecting the reconfigurable array to the processor [18]. First, the array can be connected to a bus as an 'Attached IP' as shown in Figure 1 (a). Secondly, the array can be placed next to the processor as a 'Coprocessor' as shown in Figure 1 (b). In this case, the communication is done using a protocol similar to those used for floating point coprocessors. Finally, the array can be placed inside the processor like a 'FU (Functional Unit)' as shown in Figure 1 (c). In this case, the instruction decoder issues special instructions to perform specific functions on the reconfigurable array as if it were one of the standard functional units of the processor.

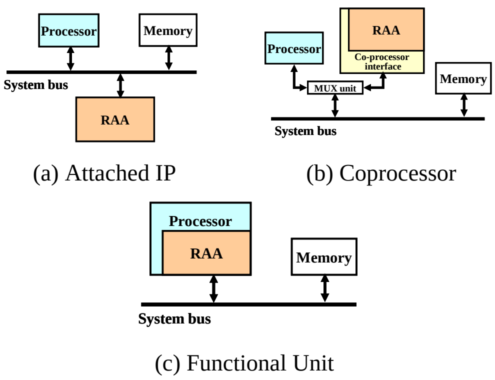


Figure 1. Basic types of reconfigurable array coupling [17].

We have implemented the first type of reconfigurable architecture connecting the reconfigurable array as an Attached IP. It consists of a RISC processor, a main memory block, a DMA controller, and a coarse-grained Reconfigurable Core Module (RCM), which is similar to the Morphosys architecture [10]. The communication bus is AMBA AHB [19], which couples the RISC processor and the DMA controller as master devices and the RCM as a slave device. The RISC processor executes control intensive, irregular code segments and the RCM executes data-intensive kernel code segments. The block diagram of the entire reconfigurable architecture is shown in Figure 2.

3.2 Power Breakdown of Coarse-Grained Reconfigurable Architecture

We have implemented the architecture shown in Figure 2 at the RT-level with VHDL [13][14]. It contains 8x5 reconfigurable array of PEs, which we think is big enough for most of the applications considered in our experiments. We have analyzed the power breakdown of the reconfigurable architecture. The architecture has been synthesized using Design Compiler [20] with technology of DongbuAnam [21] 0.18 ㎛. We have used SRAM Macro Cell library for the frame buffer and configuration cache. ModelSim [22] and PrimePower [20] have been used for gate-level simulation and power estimation. To obtain the power breakdown data, we have used 2D-FDCT as the kernel for simulation with operation frequency of 100MHz and typical case of 1.8V Vdd and 27℃.

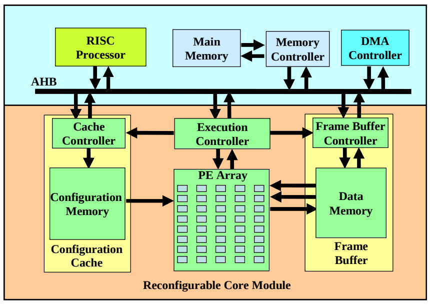


Figure 2. Block diagram of our coarse-grained reconfigurable architecture.

As can be observed from Figure 3, the reconfigurable architecture spends about 48% of total power in the PE array that is composed of many computational logics such as ALU, divider, multiplier, shifter, and register files. The PE array occupies most of the power consumption, which is natural because coarse-grained architecture aims to achieve high performance and flexibility with plenty of resources.

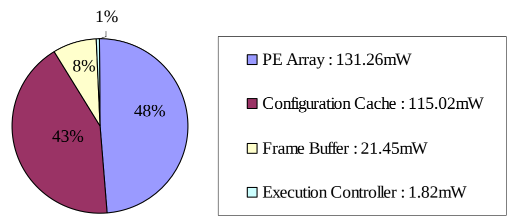


Figure 3. Power breakdown for CGRA running 2D-FDCT.

In the case of configuration cache, it spends about 43% of overall power taking the second largest portion. Even though the frame buffer uses the same kind of SRAM as the configuration cache and the size is about the same, it consumes much less power (8%). This is because the configuration cache performs read operations frequently to load the context words, one for each PE, whereas the frame buffer performs load/store operations less frequently to access data on row basis rather than for every PE.

3.3 Configuration Cache and Code Mapping

We can consider two different models for mapping loops onto coarse-grained reconfigurable architecture – SIMD and loop pipelining. SIMD computation model is efficient for computation intensive, data-parallel applications requiring less context words to configure the PE array [10]. Since data load and computation are temporarily separated in this model, array elements are not efficiently utilized. In the case of loop pipelining, different operations in a loop can be executed simultaneously in a pipeline [8]. With this flexibility, data load and computation can be simultaneously executed and all reconfigurable array elements can be efficiently used. In some loops, the performance of pipelining is roughly the same as the performance of SIMD. However, if a loop has frequent memory operations, the pipelining will render much higher performance. Therefore, we apply the loop pipelining technique to the mapping we propose in this paper.

The configuration cache of SIMD model is designed to broadcast the configuration. So PEs in the same row or column share the same context word for SIMD operation [10]. However, in the case of loop pipelining, each PE can be configured by different context word. Our configuration cache is composed of 40 Cache Elements (CEs) implemented as SRAM blocks and a cache controller for controlling each CE (Figure 4). The context register between a PE and a CE is used to keep the cache access path from being the critical path of the CGRA. It is desirable for each CE to have enough layers to enable fast dynamic reconfiguration for various context switching.

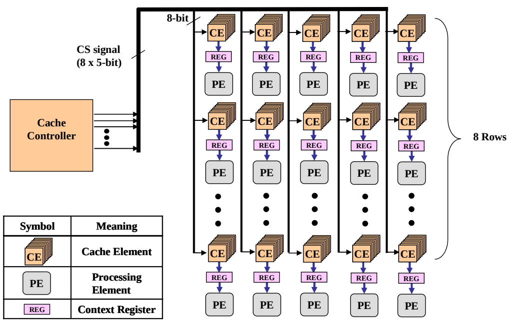


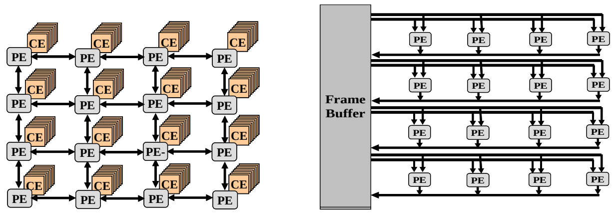
Figure 4. Distributed configuration cache structure.

4. AN APPROACH TO REDUCING POWER IN CONFIGURATION CACHE

As mentioned in the previous section, power consumption in configuration cache is critical. In this section we propose an approach to reducing the power in configuration cache. It achieves its goal by combining power conscious code mapping and a novel cache structure.

4.1 Spatial Mapping and Temporal Mapping

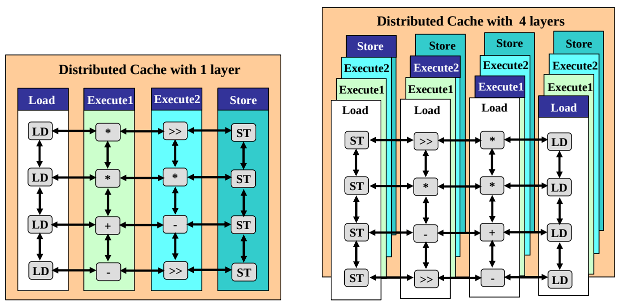
To illustrate the power-conscious code mapping, we assume a general mesh-based coarse-grained reconfigurable array of PEs, where a PE is a basic reconfigurable element composed of an ALU, an array multiplier, shifter, etc. and the configuration of each PE is controlled by a CE. Each row of the array shares read/write-buses. Figure 5 shows the case of 4x4 array with two read buses and one write-bus connecting the array to the frame buffer.



(a) Array organization (b) Data bus structure

Figure 5. 4x4 reconfigurable array.

When mapping kernels onto the reconfigurable architecture with loop pipelining, we can consider two mapping techniques: spatial mapping and temporal mapping. Figure 6 shows the difference between spatial mapping and temporal mapping. In the case of spatial mapping, each PE executes a fixed operation with static configuration. This means that loop body is spatially mapped onto the reconfigurable array. The advantage of spatial mapping is that it may not need reconfiguration during execution of a loop because of fixed functionality of each PE. However one disadvantage is that spreading all the operations of the loop body over the limited reconfigurable array may require too many resources. Moreover, data dependencies between the operations should be taken care of by allocating interconnect resources to them and we may have to use PEs as delay registers to make every stage to have the same latency in number of cycles.



Operation : Operation executed in the current cycle

(a)Spatial mapping (b) Temporal mapping

Figure 6. Mapping techniques.

Therefore, if the loop is simple enough to map the loop body to the limited reconfigurable array and there is not much data dependency between the operations, the loop can be mapped onto the reconfigurable array with small configuration data. In the case of temporal mapping (Figure 6(b)), a PE executes multiple operations by changing the configuration dynamically within a loop. Therefore, complex loops having many operations with heavy data dependencies can be mapped better in temporal fashion, provided that the configuration cache has sufficient layers to execute the whole loop body.

4.2 Spatial Mapping with Context Reuse

Because most power consumption in the configuration cache is due to memory read-operations, one of the most effective ways to achieve power reduction in the configuration cache is to reduce the frequency of read operations. Even though temporal mapping is more efficient in mapping complex loops onto the reconfigurable array, it requires many configuration cache layers for each PE and performs power consuming read-operation every cycle. On the other hand, spatial mapping can consume less power than temporal mapping because each PE executes a fixed operation within a loop and therefore it is not necessary to read a new context word from the cache every cycle.

As shown in Figure 7, since we have a context word register between a PE and the cache, one read-operation is enough in spatial mapping to configure PEs for static operations throughout the execution of a loop. In summary, spatial mapping with context reuse is more efficient than temporal mapping from the viewpoint of power consumption of configuration cache. However, all kinds of loops cannot be spatially mapped because of the limitation of the spatial mapping. Moreover, if we consider performance, temporal mapping is a better choice for loops having long and complex loop body. In the next subsection, we propose a new cache structure and mapping technique that reduce power consumption while retaining the merits of temporal mapping.

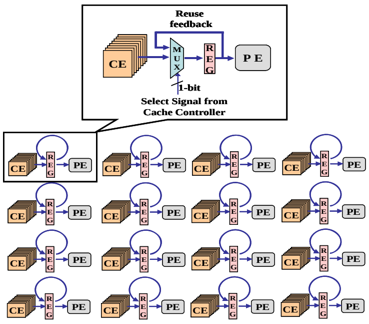
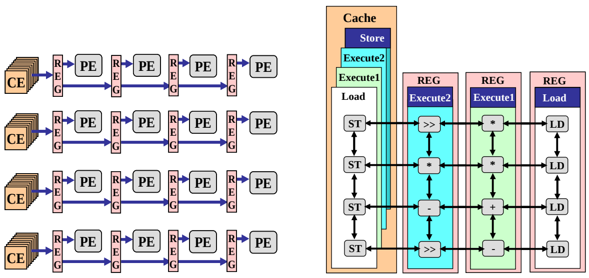


Figure 7. Configuration cache structure for context reuse.

4.3 Temporal Mapping with Context Pipelining

As shown in Figure 6 (b), in temporal mapping with loop pipelining, operations flow column by column from left to right. In Figure 6 (b) for example, the first column executes 'Load' in the first cycle and then in the second cycle, the second column executes 'Load' while the first column executes 'Execute1'. In temporal mapping, there is no need for a PE to have a CE.



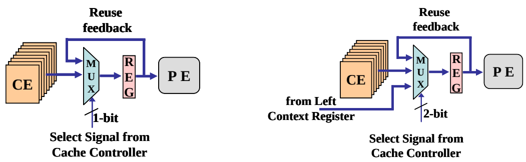
(a) Proposed cache structure (b) Context pipelining

Figure 8. Cache structure for context pipelining.

Instead, the context word can be fetched from the left neighboring column. By organizing a pipelined cache structure as shown in Figure 8, we can propagate the context words column by column through the pipeline. In this way, we can remove most of the CEs from the array, thereby saving power consumption without performance degradation. Compared with the distributed cache structure in Figure 4, the number of cache read operations is reduced to 1/5.

4.4 Hybrid Approach

Because spatial mapping with context reuse and temporal mapping with context pipelining are both power-efficient, we propose power-conscious configuration cache structure that supports both mapping techniques – we call it hybrid configuration cache structure. Figure 9 shows the connection structure for only context reuse and the one for both context reuse and context pipelining. Figure 10 shows the hybrid configuration cache structure. It is composed of cache controller, spatial cache, temporal cache, multiplexer, and de-multiplexer. Cache controller supports the same functions as the previous controller and in addition controls the selection between spatial cache and temporal cache. Compared with the temporal cache, the spatial cache has much less number of layers since it is not necessary to change the context at every cycle. Although the temporal cache has many layers, we keep only one column of cache memory thanks to context pipelining.



(a) Structure for only context reuse (b) Structure for context reuse and context pipelining.

Figure 9. Connection between CE and PE.

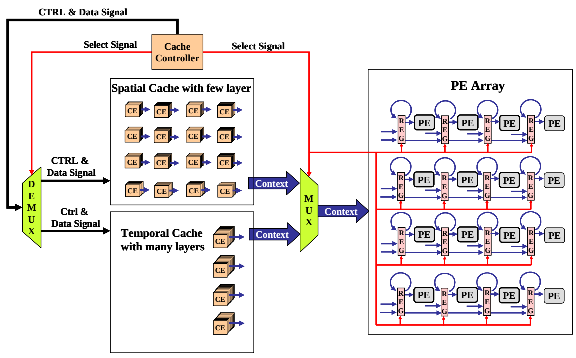


Figure 10. Hybrid configuration cache structure.

Our approach of power-conscious code mapping based on hybrid configuration cache structure exploits both spatial and temporal mapping to save power consumption. When we select spatial mapping, we save power because the number of layers has been reduced a lot. When we select temporal mapping, we save power since many memory read-operations are replaced with register shift operations. The proposed approach also saves cache area since we keep only one column of temporal CEs and much less layers of spatial CEs. As mentioned previously, the approach does not incur any performance degradation.

5. EXPERIMENTS

5.1 Experimental Setup

5.1.1 Hardware design and power estimation

We have implemented reconfigurable architecture with power-conscious configuration cache based on the 8x5 reconfigurable array model mentioned in section 3. The proposed reconfigurable architecture with new configuration cache has been designed at the RT-level with VHDL. The spatial cache of the proposed architecture has 16 layers, which is half the size of the previous one and temporal cache has 32 layers, which is the same as the previous one. The proposed architecture has been synthesized with Design Compiler [20] using technology of DongbuAnam [21] 0.18 ㎛. ModelSim [22] and PrimePower [20] are used for gate-level simulation and power estimation. Simulation conditions are the same as the previous one mentioned in subsection 3.2 (operation frequency of 100MHz and typical case of 1.8V Vdd and 27℃).

5.1.2 Application kernels

We have applied several kernels of Livermore loops benchmark [23], DSPstone [24], and representative loops in MPEG-4 AAC decoder, H.263 encoder, and H.264 decoder to the previous and proposed architectures. Table 1 shows the mapping feasibility of the kernels. Each circle marking in the table indicates the best mapping technique for the corresponding kernel. The criterion is performance. If the performance of spatial mapping is the same as temporal mapping, the choice is spatial mapping. Hybrid mapping means that both mapping techniques are used. In some case, temporal mapping is necessary for performance and spatial mapping is also indispensable to complete mapping of the kernel, which is classified as hybrid mapping.

Table 1. Mapping feasibility of evaluated kernels

Kernels	Mapping feasibility		
	Spatial	Temporal	Hybrid
*First_Diff(120†)	○		
*Tri-Diagonal(60†)	○		
*Hydro(20†)	○		
*ICCG(30†)		○	
**Dot_Product(40†)	○		
**24-Taps FIR(5†)		○	
Complex_Mult in MPEG-4 AAC dec.(30†)	○		
ITRANS in H.264 dec.			○
2D-FDCT in H.263 enc.		○	
SAD in H.263 enc			○
Matrix(10x8)-Vector(8x1) Multiplication(MVM)		○	

***** Livermore loop benchmark suite, ** DSPstone benchmark suite,
† : Iteration number

5.2 Results

To demonstrate the effectiveness of our approach, we have compared the previous and proposed architectures for spatial and temporal mapping. Table 2 shows comparison of power consumption between the two mapping models. The kernels in Table 2 have been selected based on feasibility of spatial techniques. Compared to the previous architecture, we have saved up to 71.3% of the total power consumed in the configuration cache and 28% of that in the entire architecture using only temporal mapping. We have saved more power using spatial mapping. For the configuration cache and the entire architecture we have saved up to 83.7% and 32.9% respectively. We see from these experiments that we should use spatial mapping whenever it is feasible for the kernel.

Table 2. Power comparison between mapping techniques

Kernels	Prev	Prop					
	Spatial	Temporal			Spatial		
	Power (mW)	Power (mW)	Reduced(%)		Power (mW)	Reduced(%)	
			Cache	Entire		Cache	Entire
First_Diff	360.63	259.51	**71.3**	**28.0**	241.85	**83.7**	**32.9**
Tri-Diagonal	340.05	257.95	67.7	24.1	240.20	81.9	29.4
Dot_Product	321.05	253.78	65.9	21.0	244.90	74.7	23.7
Complex_Mult	371.13	285.75	67.8	23.0	269.13	81.9	27.5
Hydro	295.20	225.04	65.9	23.8	211.10	78.2	28.5

Table 3. Power measurement of temporal and hybrid mapping

Kernels	Power(mW)		Reduced(%)	
	Prev	Prop	Cache	Entire
ICCG	326.73	241.05	67.4	26.2
24-Taps FIR	330.15	231.56	**69.2**	29.9
MVM	309.54	226.95	68.0	26.7
ITRANS	246.88	168.17	61.4	**31.9**
2D-FDCT	269.55	196.12	64.5	27.2
SAD	237.86	181.99	47.6	23.5

Power: Overall power of the architecture
Prev: Previous architecture, **Prop**: Proposed architecture
Reduced Cache: Power reduction ratio compared with previous architecture in configuration cache
Reduced Entire: power reduction ratio compared with previous architecture in entire architecture

Table 4. Memory size evaluation

Architecture	Memory Element(Byte) SRAM in config' cache	Reduced(%)
Previous	5120	0
Proposed	3584	**30.0**

Table 3 shows comparison of power consumption between two architectures. The kernels in Table 3 have been selected based on the feasibility of temporal or hybrid mapping. Compared to the previous architecture, we have reduced the power by up to 69.2% for the configuration cache and 31.9% for the entire architecture using temporal and hybrid mapping.

Table 4 shows memory size evaluation between the previous architecture and the proposed one. Compared to the previous architecture, we have reduced the memory by up to 30%. This means that reconfigurable architecture with power-conscious configuration cache structure is more efficient than previous one in the aspect of memory size as well as power saving.

6. CONCLUSION

Coarse-grained reconfigurable architectures are considered to be appropriate for embedded systems because it can satisfy both flexibility and high performance. However, power consumption is also crucial for the reconfigurable architecture to be used as a competitive processing core in embedded systems. Most reconfigurable architectures have a configuration cache for dynamic reconfiguration, which consumes much power. In this paper we suggest a novel hybrid configuration cache structure that combines spatial cache and temporal cache. For spatial mapping, each PE in the reconfigurable architecture has a spatial cache with a few layers. For temporal mapping, only the PEs on the first column have temporal caches with many layers. Other columns are configured through context pipelining. Our architecture can be used to achieve much power-saving in a reconfigurable architecture while maintaining the same performance as the previous approach.

In the experiments, we show that our approach saves much power even with reduced configuration cache size. In the case of spatial mapping, power reduction ratios in the configuration cache and the entire architecture are up to 83.7% and 32.9% respectively compared with previous architecture. In the case of temporal and hybrid mapping, we have reduced the power by up to 69.2% and 31.9% respectively.

7. ACKNOWLEDGEMENTS

This work is supported by IT R&D Project funded by Korean Ministry of Information and Communications and by the IC Design Education Center (IDEC).

8. REFERENCES

[1] Reiner Hartenstein, “A decade of reconfigurable computing: a visionary retrospective,” in *Proc. of Design Automation and Test in Europe Conf.*, pp. 642-649, Mar. 2001.

[2] Reiner Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger, “KressArray Xplorer: a new CAD environment to optimize reconfigurable datapath array architectures,” in *Proc. of Asia and South Pacific Design Automation Conf.*, pp. 163-168, Jan. 2000.

[3] Bingfeng Mei, Serge Vernalde, Diederik Verkest, and Rudy Lauwereins, “Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: a case study,” in *Proc. of Design Automation and Test in Europe Conf.*, pp. 1224-1229, Mar. 2004.

[4] Nikhil Bansal, Sumit Gupta, Nikil D. Dutt, and Alex Nicolau, “Analysis of the performance of coarse-grain reconfigurable architectures with different processing element configurations,” in *Proc. of Workshop on Application Specific Processors*, Dec. 2003.

[5] Andy Lambrechts, Praveen Raghavan, Murali Jayapala, “Energy-Aware Interconnect-Exploration of coarse-grained reconfigurable processors,” in *Proc. of Workshop on Application Specific Processors*, Sept. 2005.

[6] Hui Zhang, Marlene Wan, Varghese George, and Jan Rabaey, “Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs,” in *Proc. of VLSI’ 99*, April 1999.

[7] Frank Hannig, Hritam Dutta, and Jurgen Teich, “Mapping of regular nested loop programs to coarse-grained reconfigurable arrays – constraints and methodology,” in *Proc. of EEE Int. Parallel & Distributed Processing Symp.*, p. 148a, April 2004.

[8] Jong-eun Lee, Kiyoung Choi, and Nikil D. Dutt, “Mapping loops on coarse-grained reconfigurable architectures using memory operation sharing,” in Technical Report 02-34, Center for Embedded Computer Systems(CECS), Univ. of California Irvine, Calif., 2002.

[9] Minwook Ahn, Jonghee W. Yoon, Yunheung Paek, Yoonjin Kim, Mary Kiemb, and Kiyoung Choi, “A spatial mapping algorithm for heterogeneous coarse-grained reconfigurable architectures,” *Proc. of Design Automation and Test in Europe Conf.*, Mar. 2006.

[10] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, Nader Bagherzadeh, and Eliseu M. Chaves Filho, “MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive ap-plications,” *IEEE Trans. on Computers*, vol. 49, no. 5, pp. 465-481, May 2000.

[11] Jurgen Becker and Martin Vorbach, “Architecture, memory and interface technology integration of an indutrial/academic configurable sys-tem-on-chip (CSoC),” in *Proc. of IEEE Computer Society Annual Symp. on VLSI*, 2003.

[12] Francisco-Javier Vererdas, Michael Scheppler, Will Moffat, and Bingfeng Mei, "Custom implementation of the coarse-grained reconfigurable ADRES architecture for multimedia purposes,” in *Proc. of Int. Conf. on Field Programmable Logic and Applications*, pp. 106-111, Aug. 2005.

[13] Yoonjin Kim, Chulsoo Park, Shinwon Kang, Hyunjik Song, Jinyong Jung, and Kiyoung Choi, “Design and evaluation of coarse-grained reconfigurable architecture,” in *Proc. of Int. SoC Design Conf.*, pp. 227-230, Oct. 2004.

[14] Yoonjin Kim, Mary Kiemb, Chulsoo Park, Jinyong Jung, and Kiyoung Choi, “ Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization,” in *Proc. of Design Automation and Test in Europe Conf.*, pp 12-17, Mar. 2005.

[15] Marco Lanuzza, Martin Margala, and Pasquale Corsonello, “Cost-effective low-power processor-in-memory-based reconfigurable datapath for multimedia applications,” in *Proc. of Int. Symp. on Low Power Electronics and Design*, pp. 161-166, Aug. 2005.

[16] Sami Khawam, Tughrul Arslan, and Fred Westall, “Synthesizable reconfigurable array targeting distributed arithmetic for system-on-chip applications,” in *Proc. of IEEE Int. Parallel & Distributed Processing Symp.*, p. 150, April 2004.

[17] Francisco Barat, Murali Jayapala, Tom Vander Aa Henk Corporaal, Geert Deconinck, and Rudy Lauwereins, “Low power coarse-grained reconfigurable instruction set processor,” in *Proc. of Int. Conf. on Field Programmable Logic and Applications*, pp. 230-239, Sept. 2003.

[18] Francisco Barat and Rudy Lauwereins, “Reconfigurable instruction set processors: a survey,” in *Proc. of Int. Workshop on Rapid System Prototyping*, pp. 168-173, April 2000.

[19] ARM Corp. : http://www.arm.com/arm/AMBA

[20] Synopsys Corp. : http://www.synopsys.com

[21] DongbuAnam Semiconductor : http://www.dsemi.com

[22] Model Technology Corp. : http://www.model.com

[23] http://www.netlib.org/benchmark/livermorec

[24] http://www.ert.rwth-aachen.de/Projekte/Tools/DSPSTONE