Dynamically Compressible Context Architecture for Low Power Coarse-Grained Reconfigurable Array

Yoonjin Kim and Rabi N. Mahapatra
Embedded Systems & Co-Design Group, Dept. of Computer Science
Texas A&M University, College Station, TX 77843
{ykim, rabi}@cs.tamu.edu

Abstract

Most of the coarse-grained reconfigurable array architectures (CGRAs) are composed of reconfigurable ALU arrays and configuration cache (or context memory) to achieve high performance and flexibility. Specially, configuration cache is the main component in CGRA that provides distinct feature for dynamic reconfiguration in every cycle. However, frequent memory-read operations for dynamic reconfiguration cause much power consumption. Thus, reducing power in configuration cache has become critical for CGRA to be more competitive and reliable for its use in embedded systems. In this paper, we propose dynamically compressible context architecture for power saving in configuration cache. This power-efficient design of context architecture works without degrading the performance and flexibility of CGRA. Experimental results show that the proposed approach saves up to 39.72% power in configuration cache with negligible area overhead.

1. Introduction

In order to provide high quality multimedia on mobile and embedded systems, various efficient algorithms for audio/video data transfer and processing have been developed. These algorithms are complex and characterized by data-intensive computations. For such applications, a coarse-grained reconfigurable architecture (CGRA) can provide high performance flexibility. CGRA has higher performance than general purpose processor and wider applicability than ASIC.

In spite of the above advantages, the deployment of CGRA is prohibitive due to its significant power consumption. This is due to the fact that CGRA is composed of many computational resources such as ALU, multiplier, divider and configuration cache to perform frequent memory-read operations for dynamic reconfiguration in every cycle. The configuration cache is the main component in CGRA that provides distinct feature for dynamic configuration. Even though configuration cache plays an important role for high performance and flexibility, it suffers from large power consumption. Therefore, reducing power consumption in the configuration cache has been a serious concern for reliability of embedded systems. This paper addresses the power reduction issues in CGRA and provides a framework to achieve this. The paper has following contribution:

• Design methodology for dynamically compressible context architecture and a new cache structure to support the configurability are presented to reduce the power consumption in configuration cache without performance degradation.

This paper is organized as follows. After mentioning the related work in Section 2, we describe coarse-grained reconfigurable architecture and its context architecture in Section 3. In Section 4, we present the motivation of our approach. From Section 5 to Section 10, we describe a new design flow to implement dynamically compressible context architecture with an example. Then we show final context architecture and explain context evaluation in Section 11 and 12. We show the experimental results in Section 13 and conclude the paper in the Section 14.

2. Related works

Most of the research works in CGRA have been carried out in three different aspects: architecture exploration, code compilation and mapping, and physical implementation [1]. The architecture exploration flows that have been suggested in [2][3] generate a good instance of CGRA considering area and performance without power. In [4] the authors have proposed energy-aware interconnection exploration 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 the case of code compilation and mapping, power-conscious configuration cache structure and code mapping are proposed in [9]. They classified the computation model of loop pipelining into two cases (spatial mapping and temporal mapping) and suggested spatial mapping with context reuse and temporal mapping with context pipelining for power saving of each case. Even though they achieved power reduction compared with the base architecture, their proposed techniques are dependent on specific computation model of loop pipelining. Therefore, those techniques cannot be applied to CGRAs with other computation models. Many reconfigurable architectures have been implemented with various technologies [6][7][8][10]. Most of these researches have focused on efficient design with respect to small area and high performance. In [7][10], 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.

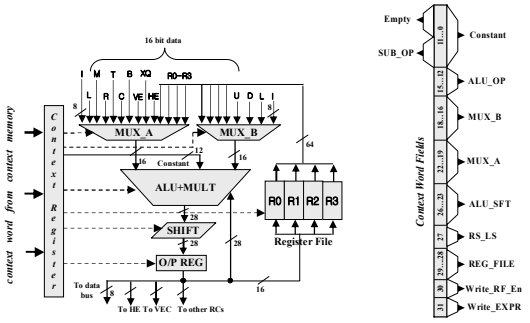
3. Preliminaries

Typically, a CGRA consists of a main processor, a Reconfigurable Array Architecture (RAA), and their interface [1]. The RAA has the array composed of identical processing

1-4244-1258-7/07/$25.00 ©2007 IEEE

elements (PEs) containing functional units and a few storage units. In addition, RAA has a data buffer to provide operand data to PE array and a configuration cache (or context memory) to store the context words used for configuring the PE array elements.

Figure 1 shows an example of PE structure and context architecture for MorphoSys [6]. 32-bit context word specifies the function for the ALU-multiplier, the inputs to be selected from MUX_A and MUX_B, the amount and direction of shift of the ALU output, and the register for storing the result as Figure 1 (a). Context architecture means organization of context word with several fields to control resources in a PE as Figure 1 (b).



(a) PE structure (b) Context architecture

Figure 1. PE structure and context architecture of Morphosys.

The context architectures of other CGRAs such as [2][3][4][5][7][8][9] are similar to the case of MorphoSys although there is a wide variance in context-width and kind of fields used by different functionality.

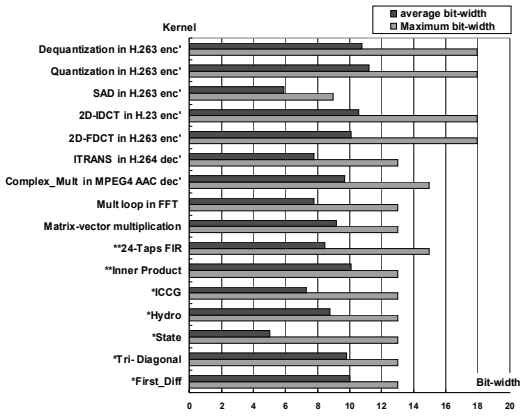
4. Motivation

4.1. Power consumption by configuration cache

By loading the context words from the configuration cache into the array, we can dynamically change the configuration of the entire array within just one cycle. However, such dynamic reconfiguration of CGRA causes many SRAM-read operations in configuration cache. In [9], power breakdown for the CGRA running 2D-FDCT was proposed with gate-level implementation at 0.18 ㎛ technology based on MorphoSys architecture. It has been shown that the configuration cache spends about 43% of the overall power, which is the second largest after the PE arrays consuming 48% of overall power budget. This is because the configuration cache performs SRAM-read operations to load the context words in every cycle at run time.

4.2. Valid bit-width of context words

When a kernel is mapped onto CGRA and application gets executed, the used context fields are limited to types of operations of the kernel executed at run time. Furthermore, operation types of an executed kernel on PE array are changed in every cycle. It means the valid bit-width of executed context word is frequently less than the full bit-width of a context word even though full bit-width can be less often used. For statistical evaluation of valid bit-width of contexts, we selected 32-bit context architecture based on [9] and mapped several kernels onto its PE array in order to maximize the utilization of the context fields. Figure 2 shows the results for various benchmark kernels and critical loops in real applications. In Figure 2, average bit-width is the average value of valid bit-widths of all the executed context words at run-time and the maximum bit-width is the maximal valid bit-width among all the context words considered at run-time. The statistical result shows that average bit-width varies from 7 to 11 bits and the maximum bit-width is less than or equal to 18 bits whereas the full bit-width is 32-bit.



*Livermore loops benchmark [13], **DSPstone [14]

Figure 2. Valid bit-width of context words.

4.3. Dynamic context compression for low power CGRA

If the configuration cache can provide only required bits (valid bits) of the context words to PE array at run time, it is possible to reduce power consumption in configuration cache. The redundant bits of the context words can be set to disable and make those invalid at run time. That way, one can achieve low-power implementation of CGRA without performance degradation while context architecture dynamically supports both the cases at run time: one case is uncompressed context word with full bit-width and another case is compressed context word with setting unused part of configuration cache disabled. In order to support such a dynamic context compression, we propose a new context architecture and configuration cache structure in this paper.

5. Design flow of dynamically compressible context architecture

In order to design and evaluate dynamically compressible context architecture, we propose a new context architecture design flow. Entire design flow is shown in Figure 3. This design starts from context architecture initialization and finally one can determine whether the initially uncompressed contexts can be compressed or not by context evaluator. From section 6 to section 9, we describe more detailed process for each stage in entire design flow.

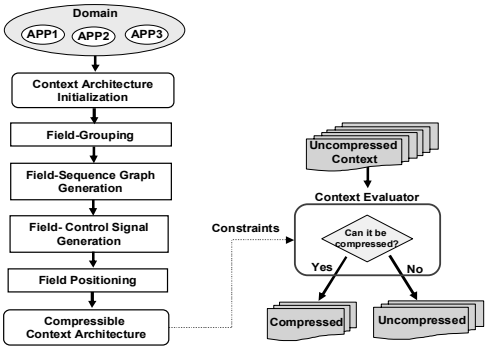


Figure 3. Entire design flow.

6. Context architecture initialization

Context architecture in CGRA design dependents on architecture specification. In the process of architecture specification, CGRA structure is evolved with PE array size, PE functionalities and their, interconnect scheme. The proposed approach starts from the conventional context architecture selection and makes it dynamically compressible context architecture through the proposed design flow. We have defined generic 32-bit context architecture as an example to illustrate the design flow to support the kernels in Figure 2. It is similar to the representative CGRAs such as MorphoSys [6], Remark [1], ADRES [2][4], PACT_XPP [7] or [9]. The PE structure and bit-width of each field are shown in Figure 4. It supports various arithmetic and logical operations with two operands (MUX_A and MUX_B), predicated execution (PRED), Arithmetic saturation (SAT_logic), shift operation (SHIFT) and saving temporal data with register file (REG_FILE). In Figure 4, all of the fields are classified by 'Control' of 2 cases – 'Processing element' and 'context register'. It means that each case is configured by the fields included in that case. Furthermore, Figure 4 shows the bit-width of each field and the component index to identify each component configured by each field.

Even though each field can be positioned on context word under conventional design flow, this initialization stage does not define any field position. It means field position for uncompressed case should be assigned by considering context compression.

Control	Field name	Bit-width	Component index
Processing Element	REG_FILE	3-bit	[0,1,2]
	MUX_A	4-bit	[3,4,5,6]
	MUX_B	4-bit	[7,8,9,10]
	ALU_OP	5-bit	[11,12,13,14,15]
	SAT	2-bit	[16,17]
	SHIFT	6-bit	[18,19,20,21,22, 23]
	WDB_EN	1-bit	[24]
	PRED	1-bit	[25]
Context register	CTXT_CTRL	6-bit	-

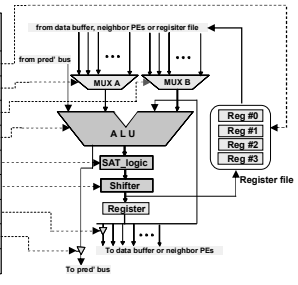


Figure 4. PE structure.

7. Field grouping

All of the context fields are grouped into three sets - necessary set, supplementary set and unnecessary set. Necessary set includes indispensable fields for all of the PE operations and supplementary set includes optional fields for PE operations. Unnecessary set is composed of fields unrelated to PE operations. It means necessary fields should be included in context words even if context words are compressed whereas supplementary and unnecessary fields can be excluded out of context words. In addition, we classify supplementary set into two subsets. One is a subset composed of fields dependent on the field of 'ALU_OP' and another is a subset composed of fields independent of 'ALU_OP'. This classification is necessary for generating field control signals in Section 9. Figure 5 shows field-grouping based on the context initialization presented in Section 6.

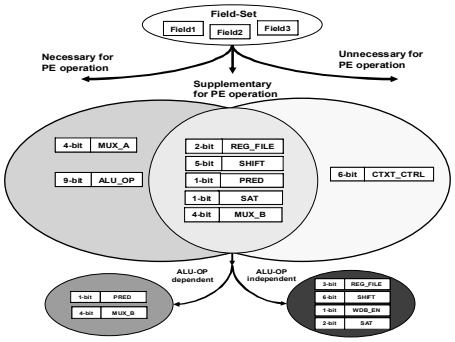


Figure 5. Field grouping.

8. Field sequence graph generation

Field sequence graph (FSG) is generated from context architecture initialization and field grouping. FSG is a directed graph composed of necessary and supplementary fields and it shows possible field combinations for PE operations based on PE structure. Each vertex of FSG corresponds to a necessary or supplementary field in field grouping and each edge of FSG shows a possible field combination between two fields. The possible field combinations can be found by vertex tracing in the edge directions and the combinations should include all of the necessary fields. Furthermore, supplementary fields can be skipped out of vertex tracing to search possible field combinations. Figure 6 shows an example of FSG from Figure 6 and Figure 5. While searching possible field combinations, some times it is possible (for example, MUX_A, ALU_OP, SAT is possible) whereas (MUX_A, ALU_OP, SAT, PRED) is not possible. FSG is a useful data structure for field positioning as described in Section 10.

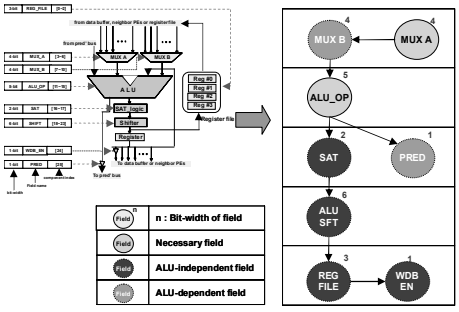


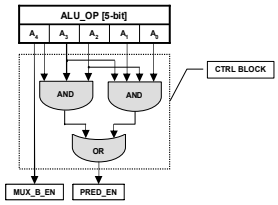
Figure 6. Field sequence graph.

9. Generation of Field control signals

When contexts are compressed, supplementary fields are relocated on compressed space and the positions of these fields may be overlapped with each other. Therefore, each supplementary field should be disabled when it is not being compressed in the context word. It means that compressed context should have control information for all of the supplementary fields in order to make unused fields disable. In this section, control signals generation for supplementary fields has been described.

Logical Operation	ALU_OP [5-bit]				
	A_4	A_3	A_2	A_1	A_0
A && B	1	1	1	0	0
A || B	1	1	1	0	1
A < B	1	1	1	1	0
A ≤ B	1	1	1	1	1
A!	0	1	1	1	1

(a) Logical operations



(b) Control signals

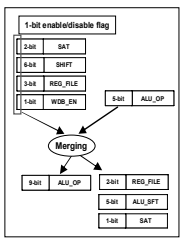
Figure 7. Control signals for 'MUX_B' and 'PRED'.

9.1. Control signals for ALU-dependent fields

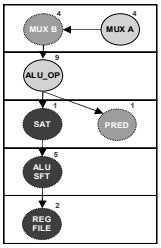
If the truth table of 'ALU_OP' is classified by the operation type, enable/disable signals for ALU-dependent fields can be generated from 'ALU_OP' with some combinational logic. Figure 7 (a) shows the truth table manipulated by classifying operations. MSB (A_4) of 'ALU_OP' is used for classifying operations according to the number of operands. For example, MSB =1 is used for the operations with two operands and MSB =0 is used for the operations with one operand. In addition, A_3~A_0 are used for classifying logical operations. Based on the truth table, we can generate control signals for two fields with some combinational logic as Figure 7 (b). We define such a combinational logic as 'CTRL BLOCK'.

9.2. Control signals for ALU-independent fields

In order to control ALU-independent fields when context words are compressed, the enable/disable flag bit on each of the ALU-independent field should be merged with a necessary field. Figure 8 (a) shows the process that 1-bit flags of ALU-independent fields are merged with 'ALU_OP'. After flag merging, the FSG should be updated because the bit-widths of some of the fields are changed and 1-bit field such as 'WDB_EN' is no longer valid in FSG. Figure 8 (b) shows an updated FSG with modified bit-widths of some of the fields.



(a) Flag merging



(b) Updated FSG

Figure 8. Updated FSG from flag merging.

10.1. Field positioning on uncompressed context word

All the fields should have default positions for the case when contexts cannot be compressed. First of all, the necessary fields are positioned to the part near to MSB and the unnecessary fields are positioned near the LSB as shown in Figure 9. Then the supplementary fields are positioned on the available space between the already occupied sides of context word. For supplementary field positioning, the bit-width of compressed context word should be determined. Compressed bit width can be different according to the definition of the capacity of compressed context word. The large capacity of compressed context word can show high compression ratio but the amount of power reduction is limited by long bit-width. However, the little capacity of compressed context word may cause low compression ratio but the power reduction ratio can be high in short bit-width. To prevent the extreme cases (much short or much long bit-width of compressed context word), we determine compressed bit-width based on following criterions.

I. *Compressed context words should be able to support all of the ALU-dependent fields.*
II. *Compressed context words should be able to include at least an ALU-independent field.*

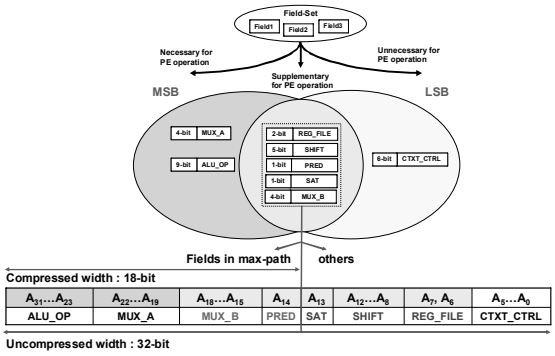


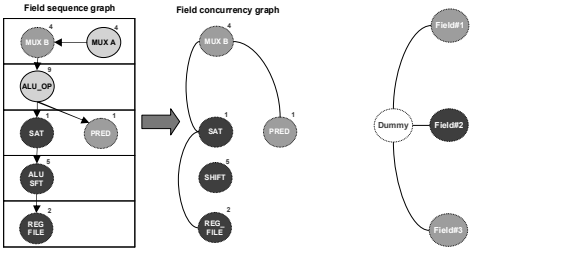
Figure 9. Default field positioning.

To satisfy criterions, we determine the longest field combination showing the maximum bit-width among I and II. The maximum width for satisfying I and II is found to be 18-bit that consists of 'ALU_OP', 'MUX_A', 'MUX_B' and 'PRED'. Therefore, 18-bit is the compressed bit-width. Supplementary fields that are included in the longest field combination are preferentially positioned on the compressed zone near the MSB and other fields are positioned on uncompressed zone near the LSB as Figure 9. After this, the positions of the necessary fields on FSG are firmly determined and the positions of the field control signals are also determined because they are included in 'ALU_OP' as necessary field.

10.2. Field positioning on compressed context word

This stage is for positioning fields on compressed context word to guarantee that all the possible field combinations are not exceeding the compressed bit-width. Therefore, first of

all, all the possible field combinations should be found. This process can be achieved by searching them from FSG and then generating field concurrency graph (FCG) such as Figure 10 (a). The FCG shows the concurrency between the supplementary fields. Therefore the FCG is used for preventing position that is overlapping between the concurrent supplementary fields. An edge between two fields means that the two fields are included in one of the possible field combinations. Even though this example does not show concurrency between more than 2 supplementary fields, such a case can be represented by adding a dummy field connected with the fields as Figure 10 (b).



(a) FCG from FSG (b) FCG with dummy vertex

Figure 10. Field concurrency graph.

Based on a given FCG, the next step is to position the supplementary fields on compressed context word. The positioning means that some supplementary fields have additional positions as well as default positions on uncompressed context words. To select a position among default and additional positions, multiplexers can be used that are composed of multiple position inputs and one feasible position output. Therefore, in this step, the field positioning is a mapping between inputs, outputs and control signals for multiplexers connected with the supplementary fields. Thus, we propose a field positioning and port mapping algorithm for the multiplexers which is outlined in [15].

Input to the algorithm is FCG and the output is multiplexer port mapping graph (PMG) such as in the Figure 11. Each vertex of PMG corresponds to an input or control signal of multiplexer and each edge shows the relationship between control signal and a position that is selected by the weight of the edge from control signals such as 'SAT_EN', 'MUX_B_EN', etc. Then the outputs of multiplexers are connected with the component index defined in Figure 4. Therefore we can implement the multiplexers for the supplementary fields by the PMG.

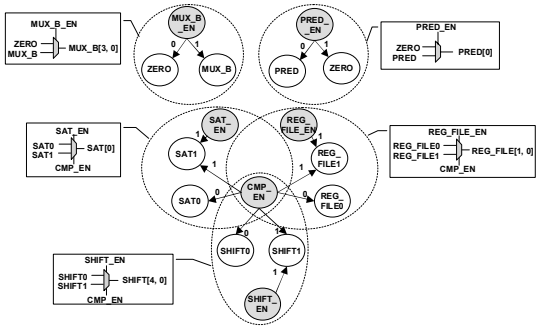
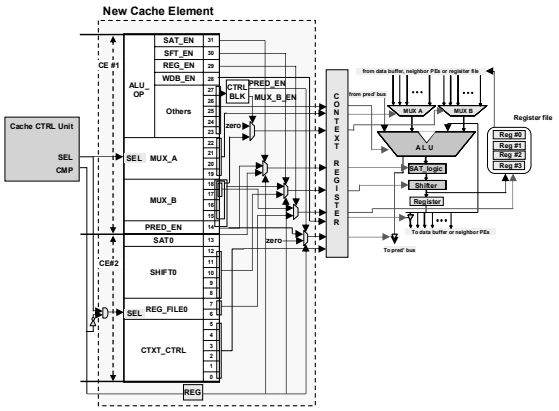


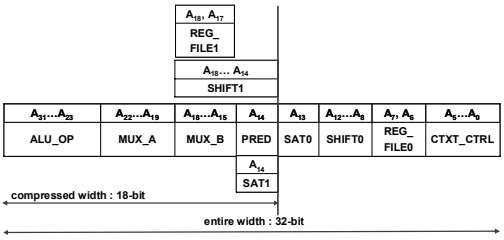
Figure 11. Multiplexer port mapping graph.

11. Compressible context architecture

After the field positioning, we have generated a specification of dynamically compressible context architecture like one in the Figure 12. Figure 12 (b) shows the final field layout of compressible context architecture. 'REG_FILE', 'SHIFT' and 'SAT' have double positions for compressed and uncompressed cases. Figure 12 (a) shows a modified structure between a PE and a cache element (CE). New cache element is composed of CE1 and CE2 and cache control unit provides compression information from port 'CMP' whether executed contexts are compressed or not. CE1 is always selected but CE2 is not selected under compression ('CMP'=1) to remove power consumption in CE2.



(a) Modified structure between a PE and a CE.



(b) Field layout of compressible context architecture.

Figure 12. Compressible context architecture.

12. Context evaluation

The context evaluator in Figure 3 determines whether initially uncompressed contexts can be compressed or not. This evaluation process can be implemented by checking the fact that a given context word is compared with one of the possible field combinations not exceeding compressed bit-width. Using FCG, we can easily check this and generate compressed context words with using position information from PMG.

13. Experiments and Results

13.1. Experimental setup

We have implemented entire design flow in Figure 4 with C++. We have initialized context architecture as the example described in Section 6 ~ 11. The implemented design flow generated the specification of dynamically compressible context architecture. For quantitative evaluation, we have

designed two CGRAs based on the 8x5 reconfigurable array at RT-level with VHDL – one is conventional base CGRA and the other is the proposed CGRA supporting compressible features in context architecture. The architectures have been synthesized using Design Compiler [10] with TSMC 0.18 ㎛ technology [11]. We have used DesignWare library [10] for the frame buffer and configuration cache. ModelSim [12] and PrimePower [10] tools have been used for gate-level simulation and power estimation. To obtain the power consumption data, we have used the kernels (Figure 3) for simulation with operation frequency of 100MHz and typical case of 1.8V Vdd and 27℃.

13.2. Results

The synthesis results show that area cost of new configuration cache including cache control unit, added interconnects and multiplexers has increased by 10.35% but the overall area-overhead is only 2.16 %. Thus, the new configuration cache structure can support dynamic context compression with negligible overheads. In addition, the synthesis results show that the critical path delay of the proposed architecture is same as the base model i.e. 12.87 ns.

To demonstrate the effectiveness of the proposed approach, we have applied several kernels in Figure 2 to the new and base architectures. These kernels were executed with 100 iterations. Table 1 shows context compression ratio for the evaluated kernels. Compression ratio means how many context words can be compressed among entire context words. The execution cycle count of each kernel on proposed architecture does not vary from the base architecture because the functionality of proposed architecture is same as the base model. All of the kernels show high compression ratio to be more than 95 %. Furthermore, the comparison of power consumption is shown in Table 1. Compared to the base architecture, it has shown to save up to 39.72% of the power.

Table 1. Power comparison

Kernels	CMP' Ratio (%)	Configuration cache Power (mW)		Reduced (%)
		Base	proposed	
First_Diff	100	471.47	288.12	38.89
Tri- Diagonal	100	519.22	313.00	39.72
State	100	501.47	309.11	38.36
Hydro	100	386.01	238.27	38.27
ICCG	100	573.16	350.01	38.93
Inner Product	100	364.50	224.56	38.39
24-Taps FIR	100	682.70	418.69	38.67
MVM	100	540.40	333.48	38.29
Mult in FFT	100	460.67	281.11	38.98
Comlex Mult	100	462.90	282.37	39.00
ITRANS	100	547.86	335.00	38.85
2D-FDCT	95.53	586.60	370.00	36.92
2D-IDCT	95.49	579.23	365.65	36.87
SAD	100	478.55	292.00	38.98
Quant	95.12	559.36	354.85	36.56
Dequant	95.23	561.41	355.10	36.75

CMP Ratio : compression ratio = (number of compressed context words/ number of entire context words)×100, **Base**: base architecture, **Proposed:** proposed architecture, **Reduced** : {1-(Proposed/Base)}×100

14. Conclusion

Power consumption is very crucial for the coarse-grained reconfigurable architecture for embedded systems and all reconfigurable architectures have a configuration cache for dynamic reconfiguration, which consumes significant amount of power. In this paper, we introduced new context architecture (dynamically compressible context architecture) with its design flow and configuration cache structure to support it. The proposed dynamically compressible context architecture can save power in configuration cache without performance degradation. Experimental results show that our approach saves much power compared to conventional base model with negligible area overhead. We have reduced the power by up to 39.72% in configuration cache.

15. 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] 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.

[3] 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.

[4] Andy Lambrechts, Praveen Raghavan, Murali Jayapala, "Energy-Aware Interconnect-Exploration of coarse-grained re-configurable processors," in *Proc. of Workshop on Application Specific Processors*, Sept. 2005.

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

[6] 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 applications," *IEEE Trans. on Computers*, vol. 49, no. 5, pp. 465-481, May 2000.

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

[8] 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.

[9] Yoonjin Kim, Ilhyun Park, Kiyoung Choi and Yunheung Paek, "Power-conscious configuration cache structure and code mapping for coarse-grained reconfigurable architec-ture," in *Proc. of Int. Symp. on Low Power Electronics and Design*, Oct. 2006.

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

[11]Taiwan Semiconductor Manufact' Comp.: http://www.tsmc.com

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

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

[14] http://www.ert.rwth-achen.de/Projekte/Tools/DSPSTON

[15] http://students.cs.tamu.edu/ykim/research/iccd_2007.htm