



DYNAMIC CONTEXT COMPRESSION OF CGRA FOR APPLICATION SPECIFIC OPTIMIZATION

A. Srilalitha*, C. Paramasivan @ Vignesh* & S. Suthagar**

* Assistant Professor, Department of Electrical and Electronics Engineering,
Dhanalakshmi Srinivasan Engineering College, Perambalur, Tamilnadu, India.

** Assistant Professor, Department of Mechanical Engineering, Dhanalakshmi Srinivasan
Engineering College, Perambalur, Tamilnadu, India.

Abstract:

Most of the coarse-grained reconfigurable 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 re-configuration 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 (2.16%).

Index Terms: Coarse-Grained Reconfigurable Architecture, Configuration Cache, Context Architecture, Embedded System, Low Power.

1. INTRODUCTION:

Application-Specific optimization of embedded system becomes inevitable to satisfy the market demand for designers to meet tighter constraints on cost, performance, and power. On the other hand, the flexibility of a system is also important to accommodate the short time-to-market requirements for embedded systems. To compromise these incompatible demands, coarse-grained reconfigurable architecture (CGRA) has emerged as a suitable solution for embedded systems [1]. CGRA is composed of reconfigurable arithmetic logic unit (ALU)-like data path blocks processing word data and supports dynamic reconfiguration of interconnections among the processing blocks whereas conventional field-programmable gate array (FPGA) has logic blocks handling bit data and require long reconfiguration time with huge routing area overhead. Therefore, CGRA not only boosts the performance by adopting coarse-grained data path array but it can also be reconfigured to adapt different characteristics of class of applications. CGRA has higher performance level than general-purpose processor and wider applicability than application-specified integrated circuit (ASIC).

In spite of the aforementioned 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.

In this paper, we address the power reduction issues in CGRA and provide a framework to achieve this. A new design flow and a new configuration cache structure are presented to reduce power consumption in configuration cache. The power saving is achieved by dynamic context compression in the configuration cache—only required bits of the context words are set to enable and the redundant bits are set to disable. Therefore, the new design flow for CGRA has been proposed to generate architecture specifications that are required for supporting dynamically compressible context architecture without performance degradation. The validation of the proposed approaches is demonstrated through the use of real application benchmarks and gate level simulations. The proposed CGRA architecture has shown to reduce power consumption by up to 39.72% in configuration cache compared to conventional context architecture with negligible area overhead of only 2.16%.

This paper is organized as follows. After mentioning the related work in Section II, we describe CGRA and its context architecture in Section III. In Section IV, we present the motivation of our approach. Then, we describe a new design flow to implement dynamically compressible context architecture with an example in Section V. Finally, we show the experimental results in Section VI and conclude the paper in the Section VII.

2. RELATED WORK:

Many CGRAs have been suggested as summarized in [1]. Most of them comprise of a set of specialized processing elements (PEs) and interconnection fabrics between them and the runtime control of the operation of each PE and the inter-connection provides the reconfigurability. These CGRAs can be classified into two cases: mesh-based reconfigurable array and linear reconfigurable array. Mesh-based reconfigurable arrays arrange their PEs mainly as a rectangular 2-D array with horizontal and vertical connections, which support rich communication resources for efficient parallelism. In the case of linear reconfigurable arrays, they support pipelined execution for stream-based applications with static or dynamic reconfiguration. Morphosys [12] and REMARC [20] are representations of mesh-based architectures. For example, Morphosys consists of Tiny_RISC processor, reconfigurable cell (RC) array, frame buffer, context memory, and direct memory access (DMA) controller. RC array is an 8 × 8 array of ALUs that performs 16-bit operations based on SIMD programming model. REMARC consists of a global control unit and an 88 array of nanoprocessors. A nanoprocessor consists of an ALU, a 16-entry data RAM, an eight-entry register file, data input registers, and data output registers. The configuration for each nanoprocessor is stored in the 32-entry instruction RAM to support MIMD execution model as well as SIMD model. Pirene [19] has linear array structure. Pirene relies on dynamic reconfiguration, allowing the reconfiguration of a PE in each execution cycle. It consists of strips composed of interconnect and PEs with registers and ALUs. The reconfigurable fabric allows the configuration of a pipeline stage in every cycle, while concurrently executing all other stages. The research on low-power CGRA has three different aspects: architecture exploration, code compilation and mapping, and physical implementation. Although the architecture exploration flows that have been suggested in [3] and [4] generate a good instance of CGRA considering area and performance, they do not deal with power consumption. Interconnect architecture explorations have been suggested for low energy [5], [6]. Because CGRA has complex interconnection for performance and

flexibility, power consumption due to interconnection is crucial. The authors in [5] 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 tradeoff between performance and energy. The authors in [6] 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 are exploited mainly for performance [7]–[11]. However, power-conscious configuration cache structure and code mapping have been proposed in [18]. 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 have fabricated Pipe Ranch in a 0.18- μ m process. Their experimental results show that the power consumption is significantly high. Authors describe that the increase in power consumption is due to the dynamic reconfiguration requiring frequent configuration and state memory accesses. Hence, power consumption by dynamic reconfiguration is a serious overhead as compared to other types of IP cores such as ASIC or ASIP.

3. PRELIMINARIES:

A. Coarse-Grained Reconfigurable Architecture

Typically, a CGRA consists of a main processor, reconfigurable array architecture (RAA), and their interface, as shown in Fig. 1. The RAA has identical PEs containing functional units and a few storage units such as ALU, multiplier, shifter, and register file. The data buffer provides operand data to PE array through a high-bandwidth data bus. The configuration cache (or context memory) stores the context words used for configuring the PE array elements. The context register between a PE and a cache element (CE) in configuration cache is used to keep the cache access path from being the critical path of the CGRA.

B. Context Architecture

The configuration cache provides context words to the context register of each PE on a cycle-by-cycle basis. From the context register, these context words configure the PEs. Fig. 2 shows an example of PE structure and context architecture for MorphoSys [12]. Thirty-two 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 Fig. 2(a). Context architecture means organization of context word with several fields to control resources in a PE, as shown in Fig. 2(b). The context architectures of other CGRAs such as [12]–[16], [19], and [20] 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:

A. 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 static RAM (SRAM)

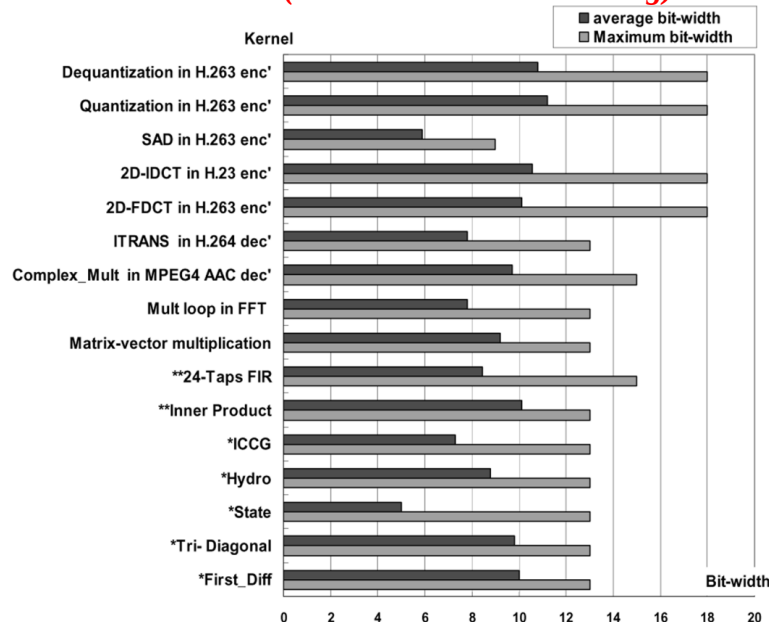


Figure (1): Valid bit width of context words: Livermore loops benchmark

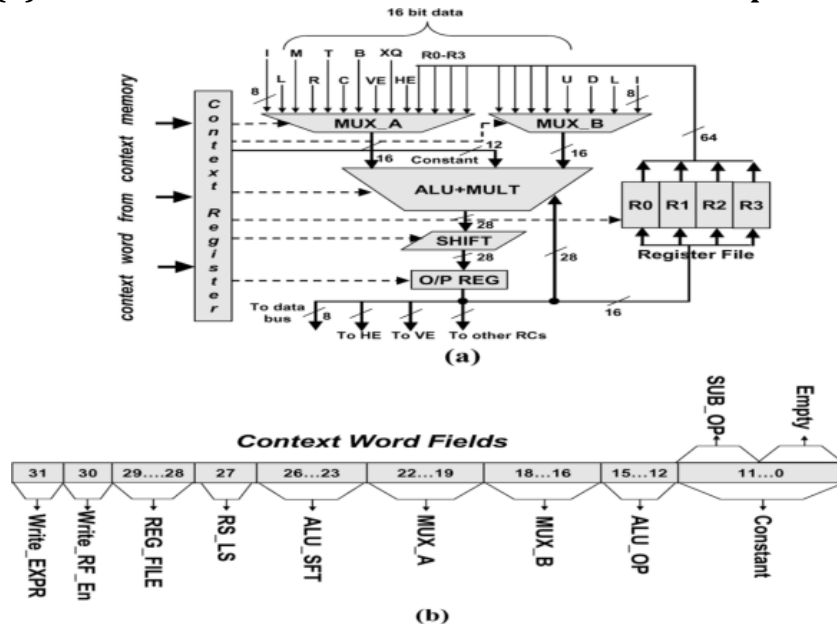


Figure (2): PE structure and context architecture of MorphoSys. (a) PE structure. (b) Context architecture.

read operations in configuration cache. The authors in [19] have fabricated a CGRA (Pipe Rench) in a 0.18- μ m process. Their experimental results show that the power consumption is significant high due to the dynamic reconfiguration requiring frequent configuration memory access. Power breakdown for the CGRA running 2-D-FDCT was proposed in [18] with gate-level implementation at 0.18- μ m 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. In addition, power breakdown for another CGRA (ADRES) running IDCT based on 90-nm technology in also shown in [5]. In this case, the configuration memory spends about 37.22% of the overall power. Therefore, it is explicit that power consumption by configuration cache (memory)

is serious overhead compared to other types of IP cores such as ASIC or ASIP.

B. Valid Bit Width of Context Words

When a kernel is mapped onto CGRA and application gets executed, the usable context fields are limited to types of operations involved due to the kernel executed at runtime. 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 [18] and mapped several kernels onto its PE array in order to maximize the utilization of the context fields. Fig. 3 shows the results for various benchmark kernels and critical loops in real applications. In Fig. 3, average bit width is the average value of valid bit widths of all the executed context words at runtime and the maximum bit width is the maximal valid bit width among all the context words considered at runtime. The statistical result shows that average bit widths vary 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.

C. 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 runtime, it is possible to reduce power consumption in configuration cache. The redundant bits of the context words can be set to disable and make these invalid at runtime. That way, one can achieve low-power implementation of CGRA without performance degradation while context architecture dynamically supports both the cases at runtime: 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 Fig. 4. This design starts from context architecture initialization, which is similar to the architecture specification stage of general CGRA design

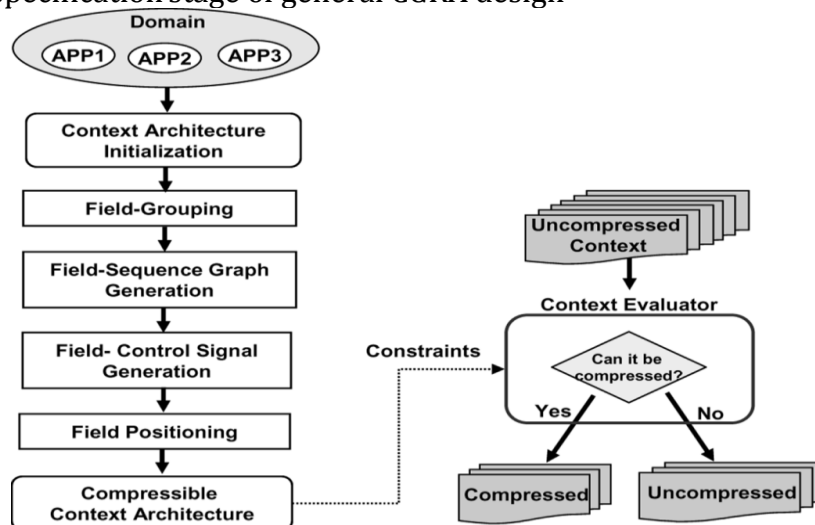


Figure (3): Entire Design Flow

flow given in [3] and [4]. Based on such architecture specifications, PE operations are determined and initial context architecture is defined. From the context initialization, fields are grouped by essentiality of PE operation and dependency with ALU operation to provide some criteria for context compression. A field sequence graph (FSG) is generated to show possible field combinations for PE operation. Then, field control signals are generated to make some field enable or disable when contexts are compressed. Based on former stages, the position of each field is defined and final context architecture is generated. Finally, one can determine whether the initially uncompressed contexts can be compressed or not by context evaluator. From Sections V-A-G, we describe more detailed process for each stage in entire design flow.

A. Context Architecture Initialization

Context architecture in CGRA design depends 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 Fig. 3. It is similar to the representative CGRAs such as MorphoSys [12], REMARC [20], ADRES [3], [5], [15], and PACT_XPP [13]. The PE structure and bit width of each field are shown in Fig. 5. 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 Fig. 5(a), all of the fields are classified by “control” of two cases—“PE” and “context register.” It means that each case is configured by the fields included in that case. Furthermore, Fig. 5(b) 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 context compression.

B. Field Grouping

All of the context fields are grouped into three sets—necessary set, optional set, and unnecessary set. Necessary set includes indispensable fields for all of the PE operations and optional 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 optional and unnecessary fields can be excluded out of context words. In addition, we classify optional 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 V-D. Fig. 6 shows field grouping based on the context initialization presented in Section V-A.

C. Field Sequence Graph Generation

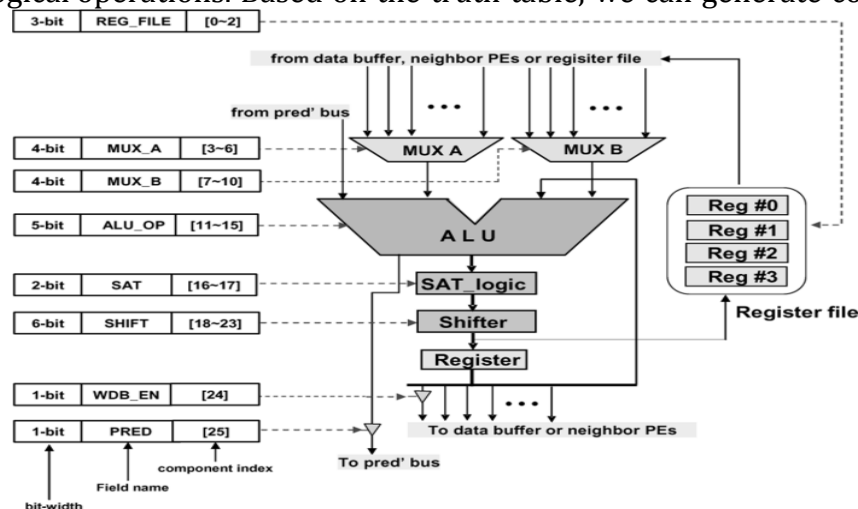
FSG is generated from context architecture initialization and field grouping. FSG is a directed graph composed of necessary and optional fields, and it shows possible field combinations for PE operations based on PE structure. Each vertex of FSG corresponds to a necessary or optional field in field grouping 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, optional fields can be skipped out of vertex tracing to search possible field combinations. Fig. 7 shows an example of FSG from Figs. 5 and 6. Field combinations, sometimes 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 V-E.

D. Generation of Field Control Signal

When contexts are compressed, optional fields are relocated on compressed space and the positions of these fields may be overlapped with each other. Therefore, each optional 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 optional fields in order to make unused fields disable. In this section, control signals generation for optional fields has been described.

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 8(a) shows the truth table manipulated by classifying operations for the example given in Section V-A. MSB (A4) of "ALU_OP" is used for classifying operations according to the number of operands. For example, it is used for the operations with two operands and is used for the operations with one operand. In addition, A_3 A_3 are used for classifying logical operations. Based on the truth table, we can generate control



(a)

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

(b)

Figure (4): Context architecture initialization. (a) PE structure. (b) Context architecture initialization.

signals for two fields with some combinational logic, as shown in Fig. 8(b). We define such a combinational logic as "CTRL BLOCK."

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. Fig. 9(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. Fig. 9(b) shows an updated FSG with modified bit widths of some of the fields.

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 Fig. 10. Then, the optional fields are positioned on the available space between the already occupied context words. For optional 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 criteria.

1) Compressed context words should be able to support all of the ALU-dependent fields.

2) Compressed context words should be able to include at least an ALU-independent field.

E. Notations for Port-Mapping Algorithm:

Notation	Meaning
G_{FCG}	field concurrency graph, $G_{FCG} = (V, E)$: V is a set composed of the optional field set and E is a set composed of edges showing the concurrency between two fields.
G_{MUX}	multiplexer port mapping graph, $G_{MUX} = (V_{MUX}, E_{MUX})$: V_{MUX} is a set composed of input signals and control signals for multiplexers and E_{MUX} is a set composed of weighted edges connecting input data with control signal.
$defV$	subset of V , $defV$ is composed of the fields having their default positions on compressed context word.
$ndefV$	subset of V , $ndefV$ is composed of the fields not having their default positions on compressed context word.
$ctxt[A_i, A_j]$	bit interval from index A_i to index A_j on the uncompressed context word, it is used for showing bit position of a field.
$width[v]$	bit-width of field v .
cmp_lsb	LSB of compressed context word.
$def_pos[field]$	default position of field such as interval type of $ctxt[A_i, A_j]$.
$ctrl_pos[field]$	one bit position of control signal for $field$ such as $ctxt[A_i]$, $ctrl_blk[A_i]$.
$field[i, j]$	component index corresponding to the interval that is from the i_{th} bit position to the j_{th} bit position on $field$.
cmp_ctrl	one-bit signal from cache control unit. '1' means executed context word compressed and '0' means executed context word not compressed.
$pdone[field]$	'1' means positioning firmly done and '0' means positioning not finished.
$mux[field]$	mux (multiplexer) connected with $field$.

F. Context Evaluation:

The context evaluator in Fig. 4 determines whether initially uncompressed

contexts can be compressed or not. This evaluation process can be implemented by checking the fact that following results.

6. EXPERIMENTS AND RESULTS

A. Experimental Setup:

We have implemented entire design flow in Fig. 4 with We have initialized context architecture as the example described in Section V. The implemented design flow generated the specification of dynamically compressible context architecture. For quantitative evaluation, we have designed two CGRAs based on the 8 5 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 [26] with TSMC 0.18-m technology [27]. We have used Design Ware library [26] for the frame buffer and configuration cache. Model Sim [28] and Prime Power [26] tools have been used for gate-level simulation and power estimation. To obtain the power consumption data, we have used the kernels (Fig. 3) for simulation with operation frequency of 70 MHz and typical case of 1.8 V and 27 C. These kernels have been executed with 100 iterations while varying test vectors.

1) Area Cost Evaluation: Table II shows the synthesis results from Design Compiler [26] of proposed architecture and base architecture. It shows 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.

2) Performance Evaluation: 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. It indicates that the dynamic context compression does not cause performance degradation in terms of the critical path delay. In addition, we have applied several kernels in Fig. 3 to the new and base architectures. The execution cycle count of each kernel on proposed architecture does not vary from the base architecture as shown in Table III because the functionality of proposed architecture is same as the base model. It also indicates that the dynamic context compression does not cause performance degradation in terms of the execution cycle count.

Table II Architecture Modification

Component	Area Cost (gate equivalent)		Overhead (%)
	Base Architecture	Proposed Architecture	
Configuration Cache	177170	195508	10.35
Entire Architecture	852861	871261	2.16

$$\text{Overhead (\%)}: \{(\text{Proposed/Base}) - 1\} \times 100$$

3) Context Compression Ratio and Power Evaluation: Table III 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. It also indicates that the dynamic context compression does not cause performance degradation in terms of the execution cycle count. All of the kernels show high

compression ratio to be more than 95%. Furthermore, the comparison of power consumption is shown in Table III³. Compared to the base architecture, it has shown³The power consumptions in Table III are much higher than the results of [18]. This is because SRAM blocks have been synthesized using Design Ware library [26] whereas [18] used Macro Cell library for SRAM implementation. The SRAM blocks of configuration cache have been unavoidably synthesized because TSMC Macro Cell library is not yet available to us. However, the synthesized model is sufficient to show relative power reduction ratio of only configuration cache. We expect that entire power reduction ratio is about 15%–20% based on the result of [18] using Macro Cell library. Power Comparison Between Base Architecture and Proposed Architecture to save up to 39.72% of the power. Four kernels (2-D-FDCT, 2-D-IDCT, Quant, and Dequant) show less reduction in power compared to other kernels. This is because all of the context words for four kernels are not fully compressed—the compression ratios are in the range of 95.12–95.53.

7. CONCLUSION:

Power consumption is very crucial for the CGRA 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.

8. REFERENES:

- [1] H. Reiner, "A decade of reconfigurable computing: A visionary retrospective," in Proc. Des. Autom. Test Eur. Conf., Mar. 2001, pp.642–649.
- [2] H. Reiner, M. Herz, T. Hoffmann, and U. Nageldinger, "KressArray Xplorer: A new CAD environment to optimize reconfigurable datapath array architectures," in Proc. Asia South Pacific Des. Autom. Conf., Jan.2000, pp. 163–168.
- [3] M. Bingfeng, V. Serge, V. Diederik, and L. Rudy, "Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: A case study," in Proc. Des. Autom. Test Eur. Conf., Mar. 2004, pp. 1224–1229.
- [4] B. Nikhil, G. Sumit, D. D. Nikil, and N. Alex, "Analysis of the performance of coarse-grain reconfigurable architectures with different processing element configurations," presented at the Workshop Appl. Specific Process., San Diego, CA, December 2003.
- [5] L. Andy, R. Praveen, and J. Murali, "Energy-aware interconnect exploration of coarse-grained reconfigurable processors," presented at the Workshop Appl. Specific Process., New York, September 2005.
- [6] Z. Hui, W. Marlene, G. Varghese, and R. Jan, "Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs," presented at the VLSI, Washington, DC, Apr. 1999.
- [7] H. Frank, D. Hritam, and T. Jurgen, "Mapping of regular nested loop programs to coarse-grained reconfigurable arrays—Constraints and methodology," in Proc. IEEE Int. Parallel Distrib. Process. Symp., Apr. 2004, p. 148a.
- [8] L. Jong-eun, C. Kiyong, and D. D. Nikil, Mapping loops on coarse-grained reconfigurable architectures using memory operation sharing Center Embedded

Comput. Syst. (CECS), Univ. California Irvine, Irvine, Tech. Rep. 02-34, 2002.

[9] J.-E. Lee, K. Choi, and N. D. Dutt, "An algorithm for mapping loops onto coarse-grained reconfigurable architectures," ACM Sigplan Notices, vol. 38, no. 7, pp. 183–188, Jul. 2003.

[10] J.-E. Lee, K. Choi, and N. D. Dutt, "Compilation approach for coarse grained reconfigurable architectures," IEEE Des. Test Comput., vol. 20, no. 1, pp. 26–33, Jan. 2003.

[11] F. Hanning, H. Dutta, and J. Teich, "Regular mapping for coarse-grained reconfigurable architectures," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process., May 2004, pp. 57–60.