2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP) | 978-1-6654-2701-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASAP52443.2021.00014

Power, Performance and Area Consequences of Multi-Context Support in CGRAs

Vimal Chacko and Jason Anderson
Dept. of Electrical and Computer Engineering, University of Toronto
vimal.chacko@mail.utoronto.ca, janders@ece.utoronto.ca

***Abstract*—A feature associated with coarse-grained reconfigurable arrays (CGRAs) is dynamic reconfigurability, wherein the CGRA supports *multiple contexts*. The multiple contexts form a set of configuration bitstreams that are loaded into the CGRA simultaneously and cycled through according to a schedule. Multi-context allows the CGRA hardware to be time-multiplexed: the logic blocks and interconnect can perform different functions according to the context selected in a given clock cycle. We consider how multi-context may be implemented at the circuit level, and evaluate three circuit implementations from the power, performance and area (PPA) perspectives. Results show that the choice of multi-context circuit implementation has an appreciable impact on the overall CGRA PPA. We also quantify the PPA overhead of the multi-context feature in CGRAs vs. a single-context device.**

***Index Terms*—CGRAs, coarse-grained reconfigurable arrays, multi-context, dynamic reconfiguration, power, performance**

I. Introduction

Coarse-grained reconfigurable arrays (CGRAs) are programmable hardware platforms with large programmable logic blocks and datapath (word-wide) configurable interconnect. A distinguishing feature of both academic and industrial CGRAs (e.g. [1], [2]) is the notion of *multiple contexts*, wherein *multiple* configuration bitstreams are loaded into a CGRA instead of just one. The CGRA cycles through the bitstreams according to a predefined pattern. The behavior of the CGRA logic and interconnect can differ for each context (bitstream). The multi-context feature represents dynamic reconfigurability of the CGRA and permits time-multiplexing the underlying CGRA hardware. In this work, we examine how multi-context should be implemented in hardware, and the trade-offs associated with the implementation alternatives.

Fig. 1 elaborates on the multi-context concept for a 2×2 CGRA. The left side of the figure shows context 0, performing a right-shift and addition on externally-supplied operands and multiplying the results together. The product is stored in a register. The right side shows context 1, which performs a logical `OR` of the product with an external input, performs a logical `AND` on two external inputs, and feeds the `OR` and `AND` outputs to a divider unit. The output of the divider is fed to an external output. Observe that the function of each of the blocks, and their connectivity differs in each context.

While prior CGRAs employ and leverage the multi-context concept, there appears to be no published study on how it might be realized in circuitry, nor does there appear to be a quantitative analysis of the power, performance and area trade-offs associated with the multi-context feature. We designed and evaluated three different circuit alternatives for implementing multi-context support in CGRAs. The first, which we believe to be most intuitive, cycles through contexts using a circular shift register; the second uses a one-hot "context" counter and `AND`/`OR` logic to select configuration bits based on the current context; the third approach uses a binary counter and multiplexer-based logic. The research is conducted using the open-source CGRA-ME (CGRA modelling and exploration) framework [3]. We focus on multi-context CGRA architectures that are fully synthesizable in standard cells. We believe this approach is attractive since such CGRAs can be easily ported across process nodes, or incorporated within SoCs.

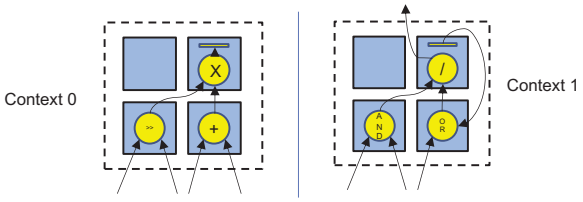


Fig. 1: Multi-context concept.

The contributions of this paper are: 1) We propose three alternative hardware implementations for multi-context support in CGRAs. 2) We integrate automated generation of the proposed hardware implementations into the CGRA-ME framework [3]. 3) The proposed multi-context implementations are evaluated using the HyCUBE CGRA [2]. 4) CGRAs with varied numbers of contexts are implemented with a 65nm standard-cell ASIC flow. 5) A power/performance/area (PPA) study of the multi-context design proposals. 6) We also assess the PPA overhead of incorporating multi-context support into a CGRA relative to a single-context (statically configurable) baseline CGRA.

II. Background and Related Work

CGRAs and Multi-Context: A CGRA with support for N contexts has $N\times$ the number of configuration cells as a single-context CGRA. Most commonly, such contexts are iterated through in a round-robin manner. The rationale is that CGRAs are typically used to accelerate loop bodies, represented as dataflow graphs (DFGs). The use of multiple contexts allows the computations of a DFG to be spread across time, permitting a smaller-size CGRA to be used. The CGRA cycles straight-line (without control flow) through the contexts that implement the loop's DFG, returning to the first context in the cycle after the last context is reached.

MorphySys was one of the earliest CGRAs to support this feature, having 32 contexts [4]. ADRES is a popular CGRA architecture template that supports the same style of dynamic reconfigurability [5]. The recently proposed HyCUBE also supports up to 32 contexts [2]. The interested reader is referred to a recent survey of CGRA architectures [6].

978-1-6654-2701-2/21/$31.00 ©2021 IEEE
DOI 10.1109/ASAP52443.2021.00014

CGRA-ME Framework: CGRA-ME is a software framework for the modelling and evaluation of CGRA architectures. CGRA-ME accepts two inputs: 1) a description of a CGRA architecture, and 2) an application benchmark to map onto the CGRA, represented as a dataflow graph (DFG). CGRA-ME creates an in-memory model of the CGRA and contains architecture-agnostic mapping algorithms [7] to map the application DFG onto the CGRA. The CGRA-ME tool can also generate synthesizable Verilog for the modelled CGRA, and an application bitstream to configure the CGRA for the mapped application. The Verilog and bitstream can be used for verification, or for synthesis into a standard-cell ASIC or as an FPGA overlay.

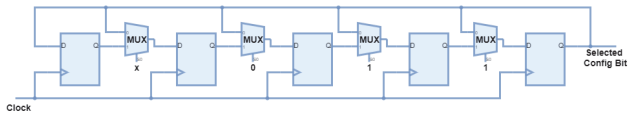
III. Circuit Implementations for Multi-Context

We evaluate three different implementations of round-robin multi-context support, elaborated upon below. The designs considered also support an "unused contexts" scenario, where the underlying CGRA *supports* N contexts, yet the particular application only requires M contexts, where $M \leq N$. The rationale for permitting $M \leq N$ is that different applications may have different context-usage requirements.

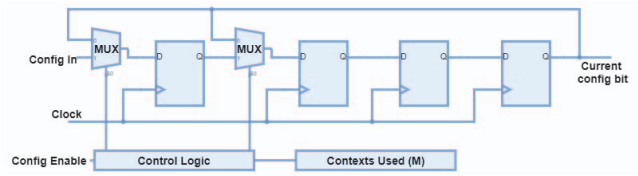
Circular Shift Register: For round-robin context switching, an intuitive design option is to use a circular shift register. Fig. 2a shows an $N = 5$ context simplified example. The rightmost flip-flop outputs the current-context bit. Each cycle, the context bits rotate. Observe that to support scenarios where the number of used contexts, $M < N$, we need the feedback loop to have configurable length, achieved through the 2-to-1 multiplexers, creating a set of tap points for the feedback loop. The select inputs of the 2-to-1 multiplexers are driven from additional configuration cells whose values are set to select the configured length. In this example, there are N-1 2-to-1 multiplexers.

The multiplexers in Fig. 2a contribute to increased area. An optimization is possible if the bitstream generator for the CGRA is able to "repeat" configuration bits in the bitstream. Consider, for example, an 8-context CGRA in a 2-context usage scenario ($N = 8$, $M = 2$). No feedback tap points are required as long as the bitstream for the two used contexts can be repeated 4 times by the bitstream generator. This allows the number of feedback multiplexers to be reduced by more than half – a significant area savings. With the simplified design, $N - 1$ multiplexers are required. In the optimized design, $N/2 - 1$ multiplexers are required.

One circular shift register instance is needed for each configurable structure within the CGRA, e.g. each ALU, interconnect multiplexer, and so on. All such instances will be shifting in tandem, visiting the same context at the same time. Some CGRA structures, such as an ALU, may require a multi-bit-wide opcode indicating the operation to perform in a given context. A circular shift register instance is needed for each bit within the multi-bit opcode. The configuration cells that feed the multiplexer select inputs are shared by *all* circular shift registers within the CGRA.



(a) Simplified example with $N = 5$ and select settings corresponding to $M = 3$.



(b) Design with configuration logic for $N = 4$.

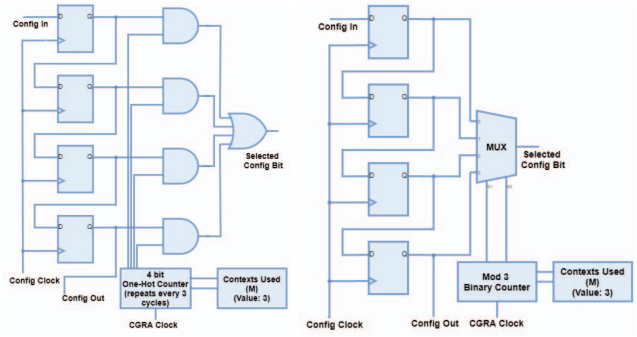
Fig. 2: Circular shift register-based approach.

Fig. 2b gives further detail on the circular shift register design for the $N = 4$ case. When a configuration bitstream is being fed into the CGRA, all the circular shift registers are connected in a serial scan chain. Following configuration, the scan chain is broken, and the feedback loops in each circular shift register are enabled – round-robin context switching commences. The two modes are managed by the control logic box in Fig. 2b.

Two issues are worth mentioning. 1) Because the configuration bits are cycling through the shift register, if a reset occurs, the CGRA may be executing any arbitrary context. If the desired initial context on reset is context 0, it is necessary to clock the circular shift registers some number of times to return to 0. 2) Observe that the configuration clock and system clock are the same signal, increasing the load of the clock network, impacting power.

One-Hot Counter Approach: Identical one-hot counters are located in each core CGRA tile: processing elements (PE), register file (RF), I/O, and memory port. Each of these counters is N bits wide. The counters count in tandem: each counter in each tile outputs the same code in a given cycle. If $N = 4$ and $M = 4$ (all contexts are used), then the counters count through the sequence: 0001, 0010, 0100, 1000, 0001, ... The counters are also capable of working in a "modulo" style if $M < N$. The one-hot codes produced by counters drive `AND`/`OR` logic to select the configuration bit for the current context, as illustrated in Fig. 3a for the case of $N = 4$ and $M = 3$. Unlike the circular shift register style, the values stored in the configuration cells are not changing. For inputting the configuration bitstream, a separate *config clock* is used, and the configuration bits are also connected in a scan chain.

A design decision we considered is: *how many counters should be used?* We first implemented this approach using a single one-hot counter, at the center of the CGRA, feeding the `AND`/`OR` logic needed to select the configuration bits for every CGRA structure throughout the entire CGRA fabric. The outputs from the one-hot counter had extremely high fanout and significantly impacted the overall system performance. We thus opted to include a one-hot counter in each core CGRA tile, which drives only local within-tile circuitry.



(a) Simplified circuit design of one-hot counter approach with $N = 4$ and $M = 3$

(b) Simplified circuit design of binary counter approach with $N = 4$ and $M = 3$

Fig. 3: One-hot and binary counter-based implementations.

Binary Counter Approach: This approach uses binary counters instead of one-hot counters. So, if N is 4 and M is 3, then the counter has a size of $\lceil \log_2(4) \rceil$ bits and the counter will be counting through the sequence: 00, 01, 10, 00, 01, and so on. A multiplexer selects the corresponding configuration bit as shown in Fig. 3b. As in the one-hot counter case, we place one binary counter instance in each core CGRA tile.

IV. Integration into CGRA-ME

We modified the Verilog generator within CGRA-ME framework to be able to generate all three styles of multi-context hardware. CGRA-ME was also modified to accept the number of available contexts, N, as a command-line input, as well as the number of used contexts, M. The mapper within CGRA-ME then maps an input application onto the modelled CGRA with M contexts being used. We also needed to modify the bitstream generator of CGRA-ME to: 1) produce the bitstream in the correct format and ordering, and 2) include the number of used contexts within the bitstream itself. For the former, the bitstream includes the configuration bits for all contexts, stitched together in a manner consistent with the scan-chain of the configuration storage elements in the CGRA hardware. Functional correctness of the multi-context implementations was verified using ModelSim.

We evaluate the three multi-context approaches using two CGRAs with considerably different interconnect architectures: ADRES [5] and HyCUBE [2]. For space reasons, only results for HyCUBE are included in this paper, however, the trends observed were similar. Fig. 4 shows the HyCUBE CGRA and its PE internals. Each FU tile contains a crossbar switch with 6 inputs (from the four neighbor PEs and the ALU output in registered and combinational form) and 6 outputs (to the four neighbor PEs and two ALU operands). In HyCUBE, multiple paths through the crossbar can be used concurrently, and independently of the ALU.

V. Experimental Study

We evaluate the implementations of multi-context support in CGRAs having different array sizes and different numbers of supported contexts. We show results for 4×4 CGRAs

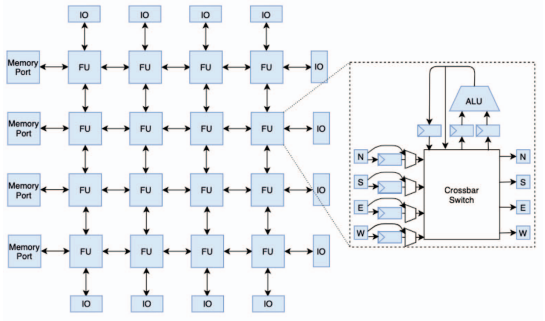


Fig. 4: 4×4 HyCUBE architecture and PE.

with 2, 4, and 8 contexts, as we observed similar trends for 8×8 CGRAs. We implement a single-context version as a comparative baseline.

The Verilog RTL code for all architectures was synthesized using Synopsys Design Compiler targeting TSMC 65nm commercial standard cells. We first synthesized each architecture in an area-driven mode. We found the critical path in the area-optimized design, and reduced it by 40% producing a timing constraint. Synopsys was run again, producing a timing-optimized CGRA. We disabled timing analysis on combinational loops and set false-path constraints on the CGRA reset and the register holding the number of used contexts. Floorplanning, placement, and routing of the architectures considered was done with Cadence Innovus. It was necessary to use floorplanning to make the physical layout match with the logical CGRA layout. Floorplanning was done using a core utilization ratio of 0.7. Circuit parasitics (R, C values) were extracted in SPEF format.

Table I shows the results. For the single-context and each multi-context implementation (left-most column), and for different numbers of contexts (second column), we show area in μm^2, critical path delay assessed in two ways, and power consumption for two benchmarks. From the area perspective, we see that, the circular shift register approach to multi-context causes the area to be ∼1-4% higher than the counter-based designs. From the results, we are able to answer: *what is the area cost of adding multi-context support?* We see that, for example, adding support for 8 contexts to a 4×4 CGRA using one of the counter-based design schemes leads to a ∼36% area increase over a single-context CGRA.

Static timing analysis was done using PrimeTime using the extracted RC values. We analyzed two worst-case path delays: 1) from a register, to a neighboring PE, through its functional unit, and to a destination register, and 2) the longest path through the entire CGRA, which starts at a register, passes through the functional unit of a PE, bypasses many PEs (combinationally), and finally terminates at another register in a destination PE. We include the latter data for completeness, as we do not believe CGRA mapping algorithms would actually result in such long paths being used. Timing columns in Table I give the values for these two path delays.

The circular shift register offers the best critical path delay, which is almost the same as the single-context architecture.

Design	Supported Number of Contexts (N)	HyCUBE				
		Area (μm²)	Timing (ns)		Total Dynamic Power (mW)	
			Critical path between neighboring PEs	Critical path through multiple PEs	Benchmark 1	Benchmark 2
Single Context	1	237492.72	4.27	29.32	2.129	1.426
Circular Shift Register	2	258086.88	4.33	30.09	4.076	3.196
	4	279966.24	4.34	29.58	3.126	2.378
	8	333246.24	4.27	30.31	3.424	3.196
One-hot Counter	2	255531.24	4.48	29.66	2.524	1.699
	4	274906.80	4.45	30.53	2.299	1.588
	8	323399.88	4.47	30.61	1.747	1.536
Binary Counter	2	253273.32	4.55	29.74	2.523	1.686
	4	276914.88	4.62	30.25	2.373	1.647
	8	326180.16	4.69	31.61	1.885	1.648

TABLE I: Area, performance and power results for 4×4 HyCUBE CGRA for different multi-context implementations.

The next best is the one-hot counter scheme with delays 4-5% higher than the single-context architecture. For the binary counter approach, the performance penalty is slightly higher, at ∼7-10%. In a representative, 4×4 CGRA with 4 contexts, the circular shift register approach increases the delay by 2%, the one-hot and binary counter approaches by 4% and 8%, respectively. Similar trends are visible for the longest paths throughout the entire CGRA as well. Comparing the area-delay product using the critical paths between neighboring PEs, the circular shift register offers modestly better area-delay than the counter-based implementations. The counter-based designs have a slightly smaller area than the circular shift register, at the expense of higher delay, with the delay hit slightly exceeding the area win.

Vector-based power analysis was done using Synopsys PrimePower using two benchmarks: benchmark 1 computes a Taylor expansion up to 4 terms; benchmark 2 is a small multiply-accumulate application. ModelSim was used for simulation at uniform 40MHz clock frequency for 1000 cycles. The right columns of Table I give the power results. Dynamic power depends on switching activity and the activity is dependent on the mapping of the application onto the CGRA. So the same mapping is used across the candidate implementations of a particular number of contexts in the experiments. This also means that comparing the power between architectures with different numbers of contexts may not always make sense because of the differences in application mapping.

All multi-context CGRAs consume more power than the single-context baseline as context switching every clock cycle contributes to higher switching activity. The circular shift register implementation has a significantly higher power overhead compared to the counter-based designs ranging from 1.55-to-2.05× the dynamic power of single context CGRA. For the one-hot counter approach, the power overhead ranges from 0.92-to-1.19×, and for the binary counter approach, it ranges from 0.99-to-1.18×. The high dynamic power of the circular shift register design is attributed to shifting the stored configurations in each clock cycle. The circular shift register also has higher system clock capacitance, as the system clock and configuration clock are the same. Regarding energy vs. power, the higher the number of contexts, the greater the number of cycles it takes to execute an application. For example, a 2-context CGRA will accept an input and produce an output every *two* clock cycles (II=2). Multiplying the power by the time it takes to compute will give the energy per computation. The circular shift register approach has a very high energy overhead ranging from ∼4× for 2 contexts to ∼15× for 8 contexts. For the counter-based implementations, it ranges from ∼2.4× for 2 contexts to ∼8× for 8 contexts.

Key Takeaways: 1) The circular shift register implementation of multi-context support offers the best performance; the least area is seen for the binary counter approach. For area-delay product, the circular shift register approach offers a slight advantage. 2) The power overhead of the circular shift register makes it impractical compared to the counter-based approaches. 3) Regarding the area, performance, and power costs of the multi-context feature in CGRAs, for a representative 4×4 HyCUBE CGRA architecture, adding 4 contexts incurs a ∼1.16× area overhead, 4% performance degradation, and ∼9% power overhead when the multi-context support is implemented using the one-hot counter approach.

VI. Conclusions and Future Work

We have explored a number of circuit alternatives for realizing multi-context CGRAs: 1) a circular shift register-based approach; 2) using a one-hot counter for configuration bit selection; and 3) using a binary counter for bit selection. We incorporated multi-context into the HyCUBE [2] CGRA architecture. Results show the approach for implementing multi-context support can significantly impact power, performance, and area (PPA), with the one-hot and binary counter-based approaches offering the best PPA trade-off, primarily due to the high power overhead of the circular shift register approach. We also quantified the PPA overhead of a multi-context CGRA vs. a statically configurable single-context CGRA. A direction for future work is to explore non-round-robin context switching architectures for their benefits and PPA overheads.

References

[1] T. Toi, N. Nakamura, T. Fujii, T. Kitaoka, K. Togawa, K. Furuta, and T. Awashima, "Optimizing time and space multiplexed computation in a dynamically reconfigurable processor," in *IEEE FPT*, 2013, pp. 106–111.

[2] M. Karunaratne, A. K. Mohite, T. Mitra, and L. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *ACM/IEEE DAC*, 2017, pp. 1–6.

[3] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson, "CGRA-ME: A unified framework for CGRA modelling and exploration," in *IEEE ASAP*, 2017, pp. 184–189.

[4] H. Singh, Ming-Hau Lee, Guangming Lu, F. J. Kurdahi, N. Bagherzadeh, and E. 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, 2000.

[5] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," in *FPL*, 2003, pp. 61–70.

[6] B. De Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in *Handbook of signal processing systems*, S. S. Bhattacharyya, Ed. Springer, 2019, pp. 427–472.

[7] M. J. P. Walker and J. H. Anderson, "Generic connectivity-based CGRA mapping via integer linear programming," in *IEEE FCCM*, 2019, pp. 65–73.