A Framework for Adding Low-Overhead, Fine-Grained Power Domains to CGRAs

Ankita Nayak, Keyi Zhang, Raj Setaluri, Alex Carsello, Makai Mann, Stephen Richardson,
Rick Bahr, Pat Hanrahan, Mark Horowitz and Priyanka Raina
Stanford University

***Abstract*—To effectively minimize static power for a wide range of applications, power domains for a coarse-grained reconfigurable array (CGRA) need to be finer-grained than a typical ASIC. However, the special isolation logic needed to ensure electrical protection between *off* and *on* domains makes fine-grained power domains area- and timing-inefficient. We propose a novel design of the CGRA routing fabric that intrinsically provides boundary protection. This technique reduces the area overhead of boundary protection between power domains for the CGRA from around 9% to less than 1% and removes the delay from the isolation cells. However, with this design choice, we cannot leverage the conventional UPF-based flow to introduce power domain boundary protection. We create compiler-like passes that iteratively introduce the needed design transformations, and formally verify the passes with satisfiability modulo theories (SMT) methods. These passes also allow us to optimize how we handle test and debug signals through the *off* tiles. We use our framework to insert power domains into an SoC with an ARM Cortex M3 processor and a CGRA with 32×16 processing element (PE) and memory tiles and 4MB secondary memory. Depending on the size of the applications mapped, our CGRA achieves up to an 83% reduction in leakage power and 26% reduction in total power versus a CGRA without multiple power domains, for a range of image processing and machine learning applications.**

***Index Terms*—reconfigurable computing, CGRA, power domains, hardware generators**

I. Introduction

With the surge of edge computing, there is an increasing demand for performing computationally expensive tasks under tight energy constraints. Application-specific integrated circuits (ASICs) are typically used to achieve high performance and energy-efficiency. This efficiency, however, comes at the cost of flexibility. As edge applications today are rapidly evolving, the hardware too must follow the trend, motivating the need for energy-efficient architectures that are configurable.

In the space of configurable architectures, coarse-grained reconfigurable arrays (CGRAs) are a promising alternative to FPGAs; they have a higher energy-efficiency that comes from operating at a word-level granularity in logic and routing. To reduce dynamic power, it is important to be able to use low operating voltage and lower threshold transistors in the functional units, but this requires multiple power domains to effectively manage the leakage power with power gating.

Typically, ASICs are partitioned into multiple power domains after carefully studying the requirements of a fixed set of applications. Since the applications are fixed, these partitions can be coarse-grained. In contrast, to effectively leverage power domains on a reconfigurable fabric, the partitions need to be finer-grained than typical ASICs. However, special isolation logic is needed between power domains to ensure that gates which read signals driven by an *off* region see valid logic values when the driving logic is powered down. A floating node can be at an intermediate voltage causing short-circuit currents that lead to high power dissipation. Adding circuits to clamp these signals incurs additional timing and area penalties proportional to the granularity of the power domains, making finer-grained power domains area- and timing-inefficient.

Some prior work has explored power gating in reconfigurable architectures. Ishihara et al. [1] present an FPGA with lookup table level fine-grained power gating with small overheads. Bsoul et al. [2] modify an FPGA to enable dynamic power gating, in which logic clusters can be selectively powered down at run-time. More recently, Miniskar et al. [3] explore power gating techniques to turn off the unused coarse functional units in a CGRA-based reconfigurable processor. Lopes et al. [4] present a CGRA for biological signal processing that can be partially turned off by power gating. While these works propose various power gating strategies, they do not explicitly address the electrical concerns of having floating outputs from the *off* region and the overheads resulting from introducing boundary protection logic to clamp those signals. We reduce this overhead by designing a CGRA routing fabric that intrinsically provides boundary protection. However, this design choice cannot leverage the conventional Unified Power Format (UPF) based flow to introduce boundary protection. Therefore, we create a framework that incrementally introduces the needed design transformations using compiler-like "passes" and formally verifies them using satisfiability modulo theories (SMT). This framework also makes it easy to experiment with different power domain-related design parameters and to generate the necessary collateral for physical design flow. Finally, we create a place and route tool that lets us efficiently map applications on the resulting CGRA with fine-grained power domains.

To summarize, we make the following contributions:

- Demonstrate boundary protection circuits that leverage existing routing resources in a CGRA, allowing introduction of fine-grained power domains with low overheads.
- Build a framework that inserts the logic required for creating power domains by applying compiler-like passes to a CGRA without power domains, and formally verifies the output of these passes.

978-3-9819263-4-7/DATE20/©2020 EDAA

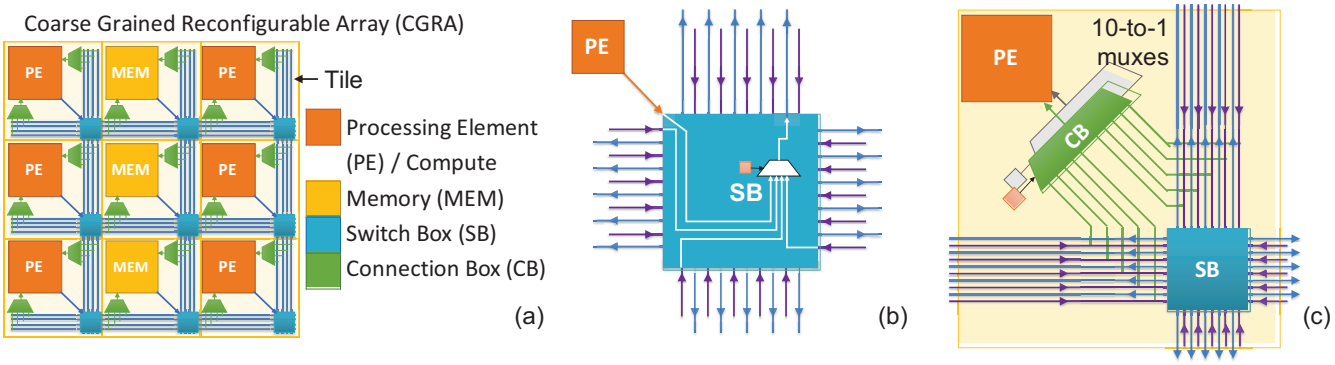

Fig. 1: (a) Our island-style CGRA consists of a 2D array of processing element (PE) tiles and memory (MEM) tiles and an interconnect with horizontal and vertical routing tracks. (b) The switch boxes (SBs) implement connections between any two tiles. Each SB output (blue arrows) on each side has a mux that selects between the PE/MEM output and the (purple) routing tracks coming from the three other sides of the SB. (c) Connection boxes (CBs) select PE/MEM inputs from the routing tracks. Here, green CB selects from inputs tracks coming from the west and the north, and the gray CB from the east and the south.

- Create a tool that performs power-domain-aware mapping, placement and routing of applications on a CGRA, and maximally turns off unused sections of the array.

Section II describes the low overhead boundary protection circuits. Section III describes our framework for inserting and verifying power domains. We use a CGRA as an example to illustrate the features of the framework. Section IV describes the power-domain-aware place and route tool used for mapping different applications on the CGRA. Finally, the chip built using our techniques is described in Section V, and the power savings for different applications are discussed in Section VI.

II. Low-Overhead Boundary Protection

Like an FPGA, our CGRA uses an island-style organization, as described by Brown et al. [5] and shown in Fig. 1. We use this CGRA as our platform to introduce the fine-grained power domains. To provide electrical isolation, typically, AND or OR logic-based isolation cells are inserted on the signals going from *off* to *on* domains as shown in Fig. 2.

To create a CGRA with fine-grained power domains, where one can individually turn on or turn off each tile, the conventional approach would be to place isolation cells on all outputs of each tile. However, the area overhead of this approach would be large, especially if there are a large number of outputs per tile. It can also worsen the timing since the additional isolation logic would be on the critical path. We observe that for reconfigurable hardware architectures, like our CGRA, there are many multiplexers already in the design to allow for flexible routing. Fig. 1(b)-(c) show the multiplexers in the switch boxes (SBs) and the connection boxes (CBs).

To avoid the large overhead from the isolation cells, we re-architect these existing SBs and CBs to incorporate boundary protection logic. As shown in Fig. 1(c), all data inputs coming into a tile go only into SBs and CBs. Instead of isolating all signals leaving the *off* domain, we isolate signals coming into the *on* domain, which serves our case because the newly architected multiplexers ensure that the floating inputs from *off* regions do not propagate into *on* regions.

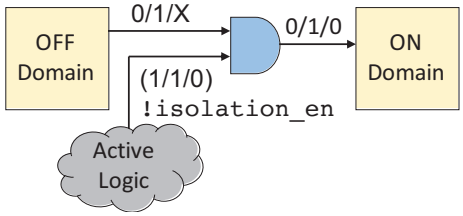

Fig. 2: For an AND isolation cell, when `!isolation_en` is high, the cell is in normal mode of operation; and when the isolation is enabled, the output of the cell is clamped to 0, thus avoiding X-propagation to the ON domain.

To morph SBs and CBs into isolation cells they must fulfill the following requirements:

- In the normal mode of operation, when the entire CGRA is *on*, the SBs and CBs retain their default functionalities;
- When any SB/CB input is X, the incoming X must not propagate through any of the gates that make up these units; although an input might be X, all gate outputs must be 0 or 1 to ensure that no part of the SB/CB circuit dissipates short-circuit power. A corollary to this rule is that the SB/CB outputs always generate a 0/1 value.

Fig. 3 shows how the SB/CB multiplexers are re-architected to perform normal functionality as well as clamping functionality. We break down the multiplexer into three stages: one-hot encoding, clamping and an OR tree. The one-hot encoding of the select input (S) makes sure that only one bit from the encoder output (S') is high at a time, meaning that all other data inputs (D) will be clamped to 0 when not selected. The OR tree processes the output from the clamping logic to create the final output. The resulting SB/CB circuit meets all of the requirements outlined in the previous paragraph. Additionally, during the implementation stage, we ensure that there is no buffer on the path to the SB/CB inputs, else a floating input to the buffer could dissipate power.

Fig. 4 shows the operation of this new SB/CB in the context of a tile. In this example, tiles to the west (W) and the north (N) of the center tile are *off* and tiles to the east (E) and the

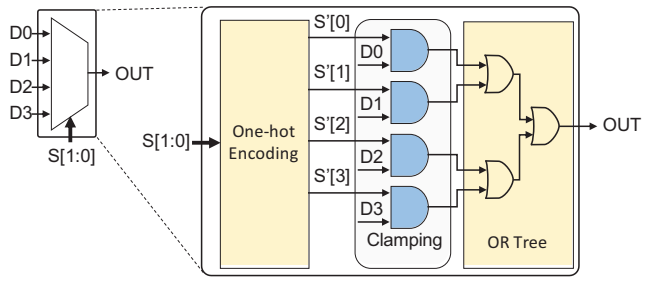

Fig. 3: Isolation logic is embedded in SBs/CBs by breaking down their multiplexers into three stages: one-hot encoding, clamping and an OR tree. This figure shows the re-architected 4-input mux present in the switch box from Fig. 1(b).

south (S) are *on*. The SB multiplexer that sends signals to the south selects between the three other sides - E, W and N, and the PE output. Since one input to the SB is always from the PE, even if all the other sides are *off*, the PE can be programmed to drive a 0/1 value. This fixes the SB. For CBs, however, all inputs can be X, as is the case in our example for the green CB on the right. Therefore, for CBs, we add one more constant input to the multiplexer (0 in our case) as shown in Fig. 4. Thus, by re-architecting the interconnect multiplexers in the CGRA, the isolation logic is naturally embedded in the design. As a result, as shown in Fig. 5, the *on* and *off* power domains can now be flexibly configured on the fabric.

III. Framework for Power Domain Insertion and Verification

We create an open-source framework that incrementally makes the design transformations required for introducing power domains into a base design using compiler-like "passes", and formally verifies the design at each stage. We create our framework using **magma** [6] and **gemstone** [7]. Magma is an open-source domain specific language (DSL) embedded in Python for describing circuits. It is similar to Chisel [8], which is embedded in Scala. Magma abstracts circuits as Python classes, which can be instanced and wired together in a structural manner. Gemstone is a dynamic layer on top of magma that allows for staged generation of circuits. It provides some primitive functions to modify circuits: `add_port(name, type)` and `remove_port(name)` add or remove a port to/from a module definition, and `wire(src, dst)` and `unwire(src, dst)` connect or disconnect ports `src` and `dst`. Together with the ability to replace existing modules and instantiate new modules, these primitive functions allow us to define a "pass," which is any function that traverses the fundamental data-structure (in our case a hierarchy of hardware modules), and adds, removes, replaces, or modifies its elements. To create our framework, we write passes that perform circuit transformations for adding power domains to a base design. This allows us to add power domain related features to any design in a decoupled fashion, without rewriting its original logical description. Since magma and gemstone are embedded in Python, we can use all of the faculties available in Python to modify circuits. This allows us to express these passes very succinctly with a few lines of code. We describe the passes that comprise our framework next.

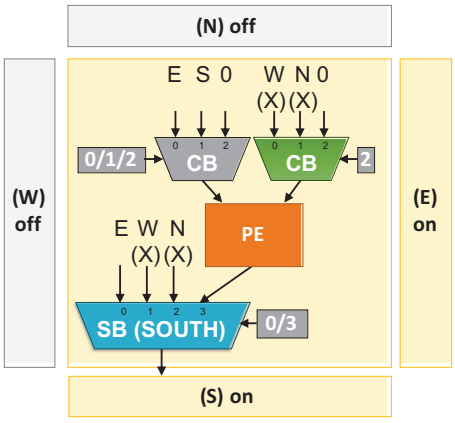

Fig. 4: PE tile with modified CBs and SBs. Gray boxes indicate which signal is selected by the muxes to avoid X-propagation. For simplicity of illustration, all inputs coming from a direction are lumped into one for the CB muxes.

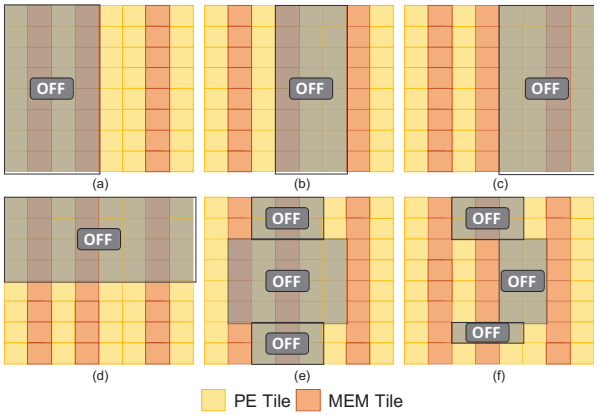

Fig. 5: Configurable power domains: a sampling of six different ways to configure *on* and *off* tiles in our CGRA.

A. Power Switch Insertion Pass

Fig. 6 shows the power switch insertion pass, which instantiates and connects power switches to the configuration register that allows the software to control which tiles are powered on. The addressing for the configuration register and decode logic is also automatically added by this pass, reducing manual effort from the user. If the user desires to add and connect the power switches during physical design using commercial tools, the user can skip adding the switches and just add the configuration logic. For users interested in RTL-level functional verification, this pass will cleanly add and connect the switches at the RTL level.

B. Boundary Protection Pass

This pass replaces the multiplexers in SBs and CBs in each tile with the re-architected multiplexers from Section II, which have inherent boundary protection circuitry. To accomplish

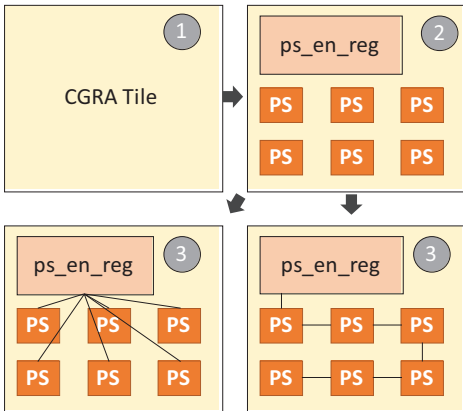

Fig. 6: Power switch insertion pass adds power switches (PS) and a configuration register (ps_en_reg) that controls if the switches are enabled. These can be connected in different styles, such as all-fanout (left), or daisy-chain (right).

```
def boundary_protection_pass(interconnect):
  # Iterate through all tiles in the CGRA
  for (x, y) in interconnect.tile_circuits:
    tile = interconnect.tile_circuits[(x, y)]
    # Find the muxes in CB in the tile
    muxes = find_available_muxes(tile)
    for mux in muxes:
        # Create new mux instance
        pd_mux = PowerDomainMux(mux)
        # Unwire and replace the old instance
        # Wire the new instance
        unwire (cb.ports, mux.ports.in)
        unwire (mux.ports.out, pe.ports.in)
        wire (cb.ports, pd_mux.ports.in)
        wire (pd_mux.ports.out, pe.ports.in)
```

Fig. 7: Since magma and gemstone are embedded in Python, we can express passes succinctly with a few lines of code. This code example shows the boundary protection pass.

this, the pass `unwires` the existing multiplexers and instantiates and `wires` up our modified multiplexers in their place. The code for this pass is shown in Fig. 7. It must be noted that if the user desires, this pass can also be used similarly to perform boundary protection using the conventional AND/OR isolation cells described in Fig. 2.

C. Always-On Buffer Insertion Pass

In our CGRA, global signals like clock, reset and configuration address and data flow from top to bottom *through* the tiles in each column. If the top tiles in any column were *off*, this would turn off the global signals to the bottom tiles. This is the case in Fig. 5 (d)-(f). To solve this issue, we treat the global signals as *always-on* feed-through nets, so that even when a top tile is *off*, the bottom tiles can receive the global signals. This pass inserts always-on buffers on these feed-through nets using a process similar to the boundary protection pass.

D. Debug Signal Isolation Pass

There are several test and debug signals in our CGRA that are affected by the introduction of power domains. For example, as shown in Fig. 8, the CGRA contains circuitry to read out any configuration register in any tile. Each tile receives a configuration address as described in the previous pass, which it decodes locally to check if it must put out the value of one of its configuration registers on the `read_config_data` bus. If no register in a tile is selected, it puts out a zero. All `read_config_data` signals are OR'ed together using an OR chain. If any tile is *off* it will send an X into the OR chain, which might corrupt the configuration read-out. The debug signal isolation pass handles this scenario. When the tile is *on*, the transformed circuit on the right behaves as OR logic, but when the tile is *off*, it blocks X-propagation from the *off* tile while allowing the upstream debug signal to propagate.

By successively applying these passes we are able to introduce the transformations required for configurable power domains in the CGRA. It must be noted that these passes are not specific to our CGRA and can be leveraged by other designs. It is also very easy to add design-specific passes in our framework using the basic circuit modification primitives.

E. SMT-based Formal Verification

Our pass-based solution introduces a new concern: how do we verify whether the transformations made by a pass are correct? Conventional approaches insert the isolation cells in a design using a UPF-based flow [9], and commercial tools like Conformal Low-Power Verification [10] are used to ensure that signals expected to cross boundaries from *off* to *on* domains are isolated. However, since we are using our custom boundary protection circuits from Section II, we cannot use the commercial tools for verifying boundary protection.

To address this problem, we utilize an SMT-based formal hardware verification tool, CoSA [11], to prove that the transformations are correct. We use SMT-based verification to exploit word-level structural information of the design, as opposed to more traditional satisfiability (SAT) solving approaches which operate on bit-level netlists.

Our goal is to formally verify that no X-propagation occurs in the chip. To achieve this, we must encode our condition as a property for CoSA to check. The first step is to formalize the concept of X-propagation. Formal tools do not have a notion of X values, so we cannot directly encode the property "there does not exist X-propagation." An X value on an input simply means there are no assumptions on the possible input values in the formal tool. However, there is no easy way to state that X values have not reached the output. Fortunately, our boundary protection drives signals with a known, constant value. Thus, our verification condition is to ensure that when the tile is *off*, for all possible inputs:

- the top-level outputs have a known constant value;
- internal signals beyond the first level of the SB and CB circuit have been masked to a known constant value;
- the PE generates some known constant value.

With this formal definition of our desired property, we were able to encode all the verification conditions described in Section II in CoSA's property language. The resulting

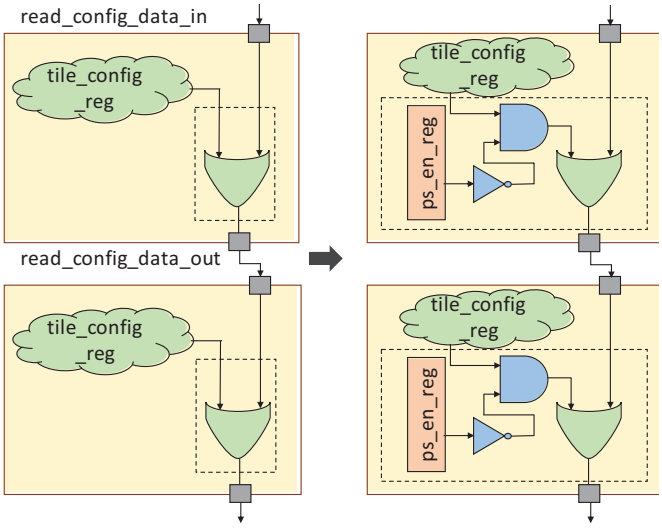

Fig. 8: Debug signal isolation pass. When the tile is *on* (`ps_en_reg=0`), the circuit is in normal mode of operation (OR logic). When the tile is *off* (`ps_en_reg=1`), the circuit blocks the X-propagation from the *off* tile while allowing configuration data to propagate. (Gates inside the dotted box as always on.)

verification problems were straightforward and CoSA was able to prove all of them in only a few seconds.

F. End-to-End Flow

Fig. 9 shows our end-to-end flow for adding power domains. We start with a base CGRA built in magma with some user-defined configuration. We then perform power domain passes on this base CGRA to create a CGRA with fine-grained power domains. The passes produce the modified CGRA RTL, which is now power domain aware, *and* the necessary collateral for the downstream physical design flow. The design transformations to generate the modified CGRA RTL take just a few minutes to run. We then use commercial EDA tools to perform synthesis, P&R and IR analysis. After P&R, we review the power-performance-area (PPA) and IR results. If the metrics don't meet our PPA and IR budget, we modify the power domain configurations to generate a new instance of the power-domain-aware CGRA.

IV. Power Domain Aware Place and Route

We built a tool chain to perform efficient static power-domain-aware mapping, placement, and routing (PnR) of different applications on a CGRA. Our fine-grained power domains give the PnR tool the freedom to choose any tile topology (to select which tiles are on, like in Fig. 5) it needs to map an application and minimize the number of *on* tiles. Prior to placement, we map the operations in the application onto PE and memory tiles, packing multiple operations into a single tile when possible. Since our routing is embedded into the tiles, the number of required tiles after mapping is the lower bound on the number of *on* tiles. We adjust the wire

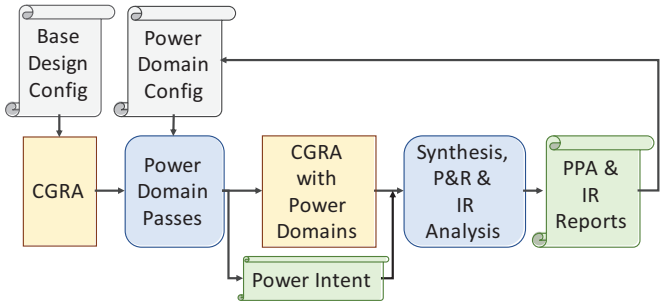

Fig. 9: Our end-to-end flow for adding power domains.

cost functions during PnR to make utilizing the wire tracks from an unused tile more expensive, than using existing tile tracks. For example, we dynamically reduce the cost of using a wiring track in the routing graph if another track in the same tile has been used for routing. These modified wiring costs decrease the number of tiles being used. The part of the fabric that is not used is turned *off* and the SBs and CBs on the fabric are appropriately configured so that no floating inputs from *off* tiles are consumed by the *on* tiles.

V. Chip Implementation

We used our framework to insert power domains into an SoC with an ARM Cortex M3 processor and a CGRA with 32×16 PE and memory tiles and 4MB secondary memory. We chose the appropriate power switch count, size, and connection style that meets our IR budget using the standard power analysis tools. Each tile is internally partitioned into two power domains, one of which can be turned *off*—this partition contains most of the logic—while the other one stays *always-on*. The *always-on* domain is required for power switch configuration logic, debug circuits, and buffers for global signals. Since our goal is not to substitute what is already supported in commercial tools, we use a UPF-based flow to do the tile-level power-domain-aware physical design. We perform power-aware functional verification at gate-level to ensure: 1) when the chip is reset, the tiles turn *on*; 2) power switches turn *on* and *off* as expected through the configuration logic; and lastly 3) when the tile is *off*, the global signals are still *on*. Finally, we take care of multi-VDD constraints for NWELL connections due to the presence of multiple power domains in the design to generate the DRC and LVS clean design in TSMC 16 nm technology, shown in Fig. 10.

VI. Results

Table I shows the area overhead for the CGRA with 1) no power domains (no boundary protection); 2) power domains with conventional isolation cell-based boundary protection; and 3) power domains with our technique of circuit transformation-based boundary protection. The entire design uses mid-Vt cells except the power management cells like power switches and always-on buffers which use high-Vt cells. We avoid usage of the leakiest variants of the cells to avoid exaggerated leakage power improvements from shutting down the tiles. As shown in Table I, our power domain boundary

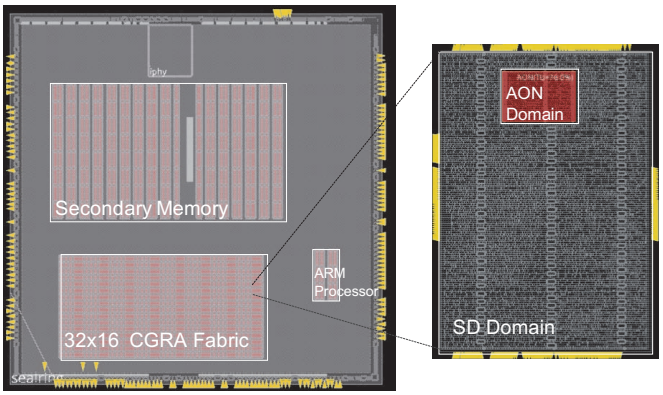

Fig. 10: Layout of the SoC is on the left. It includes the processor, secondary memory, and a 32×16 CGRA. Layout of the switchable PE tile is on the right with power switches, always-on logic, and circuits for boundary protection.

protection technique adds less than 1% area overhead while the conventional technique adds an overhead of 9.2% for the PE tile and 5.7% for the memory tile. It is important to note that the area overhead for the conventional technique is variable and depends on CGRA design parameters such as the tile's I/O count and bit-width of the data buses, since that determines the number of isolation cells that need to be inserted. Since the boundary protection logic is embedded in our design, it is not impacted by the tile parameters and the overhead is fairly constant for different CGRA parameters. We add just enough power switches (4-6% overhead) to the designs such that the IR budgets are met. The operating frequency for the CGRA built with conventional boundary protection is 475 MHz. With our technique, the isolation cells on the critical paths are removed, so we can further push the frequency to over 500 MHz. When a tile is turned *off*, the leakage power saving for the PE tile and memory tile is $22\times$ and $46\times$ respectively. The power switch leakage contributes only less 2% to the PE tile leakage and 1% to the memory tile. Most of the leakage when the tile is *off* is from the buffers added to keep the global signals *on* when the tile is *off*.

We ran a range of image processing and machine learning applications on the CGRA. Conv 1×2 is a 2D convolution that blurs two pixels horizontally. Conv 3×3 is a 2D convolution with a 3×3 kernel. Gaussian is a convolution that blurs an image. Demosaic creates an RGB image from raw pixels. Cascade has two 3×3 convolutions performed back-to-back. Harris is a corner detector. Camera pipe performs three stages of operations transforming raw images to RGB images. DNN conv is a multi-channel convolution used in convolutional neural networks. We use our power-domain-aware place and route tool described in Section IV to statically map these applications on the CGRA. The leakage and total power savings for the described applications are shown in Table II.

VII. Conclusion

We show that in FPGA and CGRA designs it is possible to embed the needed power-domain isolation circuits into the programmable routing fabric. This approach makes the area overhead of fine-grained power domains essentially zero, but it cannot completely leverage the current UPF power flows. We address this issue by using a set of Python-based tools to first add the needed power domains and then modify the routing network. These transformations are done after the "logical" design. This separation of concerns makes the logical design and the power domain transformations more reusable for future designs. The added validation, to ensure proper isolation is inserted, is performed formally using an SMT solver. Creating these tools makes low-overhead fine-grained CGRAs possible.

TABLE I: Area overhead of conventional vs. our technique.

Boundary Protection Techniques	*PE*	*Memory*
No Power Domains (PD)	1x	1x
PDs with Conventional Isolation Technique	+9.2%	+5.7%
PDs with Our Technique	+0.73%	+0.64%

TABLE II: Power savings for applications mapped on a 32×16 CGRA with 384 PE tiles and 128 memory tiles.

Applications	% Tiles Utilized		Total Power Reduction	Leakage Power Reduction
	PE	Memory		
Conv 1×2	17.7%	9.3%	26.2%	82.7%
Conv 3×3	31.3%	29.7%	12.2%	66.9%
Gaussian	35.1%	31.3%	10.8%	64.0%
Demosiac	35.9%	33.6%	10.2%	62.8%
Cascade	39.3%	37.5%	8.9%	59.2%
DNN conv	55.2%	53.0%	4.9%	44.0%
Harris	58.8%	54.7%	4.4%	41.3%
Camera pipe	90.0%	85.0%	0.8%	11.5%

References

[1] S. Ishihara, M. Hariyama, and M. Kameyama, "A low-power fpga based on autonomous fine-grain power gating," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 8, pp. 1394–1406, 2010.

[2] A. A. Bsoul and S. J. Wilton, "An fpga architecture supporting dynamically controlled power gating," in *2010 International Conference on Field-Programmable Technology*. IEEE, 2010, pp. 1–8.

[3] N. R. Miniskar, R. R. Patil, R. N. Gadde, Y.-c. R. Cho, S. Kim, and S. H. Lee, "Intra mode power saving methodology for cgra-based reconfigurable processor architectures," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 714–717.

[4] J. Lopes, D. Sousa, and J. C. Ferreira, "Evaluation of cgra architecture for real-time processing of biological signals on wearable devices," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–7.

[5] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-programmable gate arrays*. Springer Science & Business Media, 2012, vol. 180.

[6] P. Hanrahan. Magma. [Online]. Available: https://github.com/phanrahan/magma,2019

[7] R. Setaluri. Gemstone. [Online]. Available: https://github.com/StanfordAHA/gemstone,2019

[8] Chisel. Chisel. [Online]. Available: https://www.chisel-lang.org

[9] V. Gourisetty, H. Mahmoodi, V. Melikyan, E. Babayan, R. Goldman, K. Holcomb, and T. Wood, "Low power design flow based on unified power format and synopsys tool chain," in *2013 3rd Interdisciplinary Engineering Design Education Conference*. IEEE, 2013, pp. 28–31.

[10] Cadence, "Conformal low power," https://www.cadence.com/content/cadence-www/global/enUS/home/tools/digital-design-and-signoff/low-power-validation/conformal-low-power.html, 2019.

[11] C. Mattarei, M. Mann, C. Barrett, R. G. Daly, D. Huff, and P. Hanrahan, "Cosa: Integrated verification for agile hardware design," in *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2018, pp. 1–5.