Fused Functional Units for Area-Efficient CGRAs

***Abstract*—To solve the inefficiencies in general-purpose computing architectures, many different solutions have been proposed in the past decade. One of the promising architectures is CGRAs, which comprise functional units and a flexible interconnection network. In this work, we present a framework that can generate area-efficient CGRAs via fused functional unit (FU) design. Our FUs support multi-precision for both fixed- and floating-point numbers. Our evaluation shows that the proposed fused functional units result in a 58%∼60% decrease in FU area and an 11%∼15% decrease across the whole CGRA when integrated with the state-of-the-art VectorCGRA framework [1], [2].**

I. Introduction

Due to the inefficiencies in general-purpose computing [3], domain specific architectures have been the subject of interests for many years. Programmable digital signal processors (DSPs) were first developed in the 1960s [4] and FPGAs followed two decades later [5]. The efficiency gap has been shown to be around 3x between application-specific integrated circuits (ASICs) and DSPs, 25x between ASICs and FPGAs, and 500-1000x between ASICs and CPUs [3], [6]. In addition, GPUs have been shown to be 10–75x less efficient than FPGAs [7], [8]. Early systolic arrays in the 1980s used an array of processing elements (PEs), also known as functional units (FUs) [9], [10], which solution lends itself to reconfigurability. By implementing reconfigurable PEs, the arrays became reconfigurable to execute multiple algorithms. Coarse-grained reconfigurable arrays (CGRAs) appeared soon after as co-processors to accelerate traces, such as PipeRench [11], and standalone designs, like DySER [12] and DynaSpAM [13]. They aim to achieve higher performance and lower area and power consumption than general-purpose chips as well as higher flexibility than ASICs. The benefits are offset by the reduced domain of computation [14], [15] and lower area and energy efficiency, as compared to general-purpose processors and ASICs, respectively.

CGRAs comprise PEs and an interconnection network (switches). PEs are special ALUs capable of executing the same, i.e., homogeneous [11], or a location-specific, i.e., heterogeneous [12], [15]–[18], set of functions. Heterogeneous architectures increase area and energy efficiency over homogeneous ones at the cost of a lower number of attainable configurations. PEs of CGRAs that are homogeneous in composition contain more overhead. Furthermore, not all computation is homogeneous: while neural network applications need an abundance of multiply-and-accumulate (MAC) hardware (neuron, pooling), computer vision algorithms, for example, use a more diverse set of functions.

It has also been established that different domains require different precisions of computation, both for integer and floating-point arithmetic [19]–[21]. Encryption algorithms, depending on the required level of security, multiply variable-length integers. In the floating-point domain, high-end simulations require bit widths beyond that of the quadruple-precision format, while neural networks in embedded applications, for example, trade off accuracy for energy efficiency and settle for lower precisions.

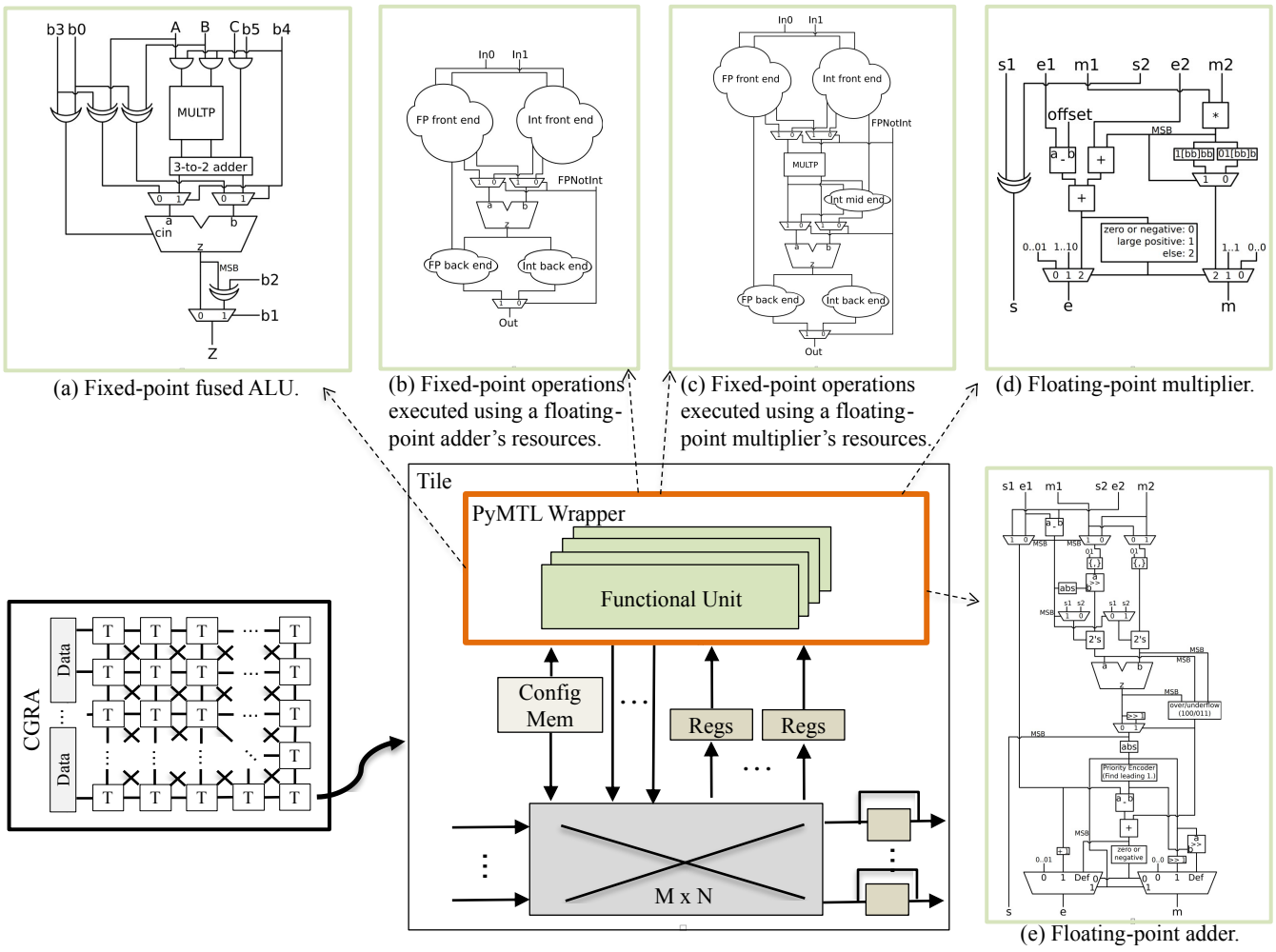
The work presented in this paper focuses on the composition of PEs and switches commonly found in CGRAs. The paper describes the design-space exploration of functional units (FUs) in terms of efficiency vs. configurability, i.e., what functions can be supported at low energy and area overheads. The area- and power-consumption overheads associated with the increasing number of available FU functions are evaluated. The cost of switches in terms of banding and granularity is analyzed as well. The banding number determines from how far vertically a FU input can receive data. For example, a banding number of two means that a FU input is connected to the FUs one and two levels above, as well as to the FU on the same level, and to the FUs one and two levels below. The results and the implications of the findings for future use of FUs and switches in CGRAs are discussed and a case study is presented. The main finding is that some functions have high cost and should be considered only when necessary, while others have marginal cost hence their addition is almost free. Such functions should be added even if not necessary for the specific set of applications, as they increase configurability at a low cost. The following contributions are made throughout the paper:

- Parameterized generator of configurable FUs and a banded switch.
- Evaluation of fine-grained cost of FUs and the switch in terms of banding and granularity.
- A software framework to support auto-integration of functional units with the VectorCGRA framework.

II. Prior work

From a high-level perspective, general-purpose architectures are inefficient for domain-specific tasks mainly due to their large memory footprints and generic granularity. [3] proposed a co-accelerator by defeaturing a general-purpose CPU running an H.264 encoder. The three classes of optimization are the introduction of application-specific instructions, data storage and supply networks, as well as arithmetic blocks. The result is a restricted DSP, which could be used as a co-processor accelerator thanks to its VLIW implementation.

Similarly, [22] observe that the main sources of efficiency in application-specific designs are the specialized compute structures and their connectivity. From there, it can be concluded that an array of multi-granularity processing elements (PEs)



(a) Fixed-point fused ALU.

(b) Fixed-point operations executed using a floating-point adder's resources.

(c) Fixed-point operations executed using a floating-point multiplier's resources.

(d) Floating-point multiplier.

(e) Floating-point adder.

Fig. 1: Parameterizable VectorCGRA architecture and a tile with the PyMTL fused functional unit wrapper. Sub-figures (a)–(e) depict various SystemVerilog RTL functional units that can be wrapped and used by the PyMTL VectorCGRA framework.

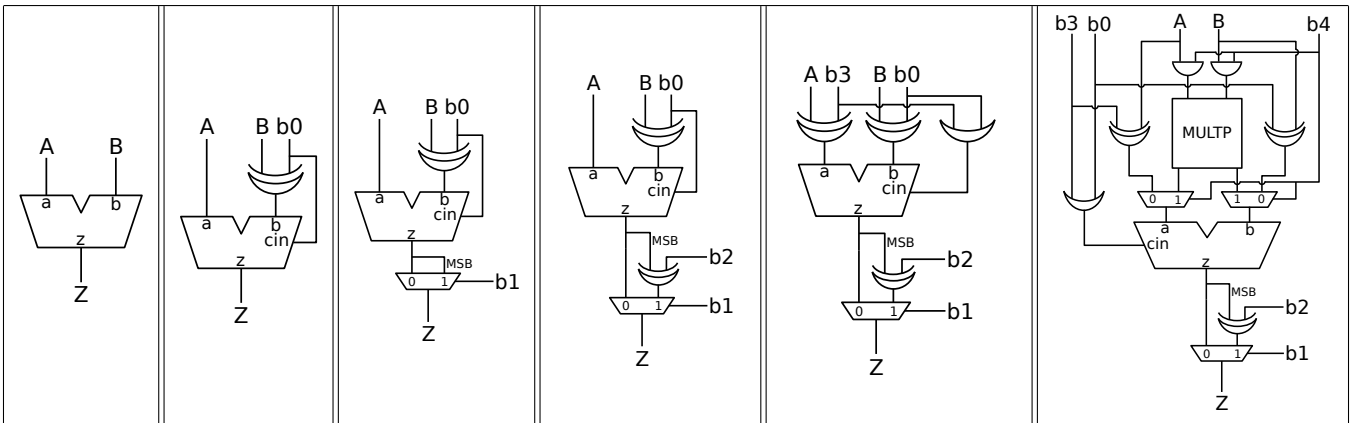
connected by a sparse switch network gets rid of most of the overhead associated with general-purpose computing, and reaches ASIC-like efficiency while maintaining reconfigurability.

CGRAs are configured by selecting each PE's function and connecting these PE functions in a way that implements the desired algorithm. Since CGRA fabrics can contain hundreds of thousands of PEs, thoughtful PE composition is imperative to maintain a reasonable area and power budget. Similarly to this work, Composite Cores [18] advocates for heterogeneous computing. It leverages heterogeneity at a higher level by migrating traces inside the processor core based on their computational intensity. Moving migration into the core reduces switching between tasks of different complexity compared to multi-core designs, such as Arm's big.LITTLE architecture. The motivation, however, is the same behind all three designs; if a set of functions are less computationally intensive than others, mapping them to a new, simple unit created by defeaturing complex resources increases their executions' power and area efficiency.

Besides the nodes of data-flow graphs (mapped to FUs), the other high-level building block of CGRAs is the graph's edges, i.e., the routing between FUs. As design complexity increases, measures have to be taken to mitigate the scaling of the interconnection network's power consumption and area. One study evaluates the advantages of power gating unused switch subnetworks [23]. This solution is applied to many-core processor designs, and would add significant overhead to a CGRA. In the presented domain-specific CGRAs, the objective is to increase energy and area efficiency by implementing resources with a high utilization ratio. For this reason, the effectiveness of banding in switches is evaluated. All connections are expected to be used during program execution, but many connections can be eliminated by clever mapping of the algorithms. The elimination of connections results in a lower interconnect area and power consumption.

Another proposed solution for interconnect scaling in multi-core CPUs is a 16-by-16 network-on-chip capable of connecting 256 processor nodes [24]. The mesh network is a point-to-point connection between the nodes, and increases throughput

TABLE I: Incremental FU designs.



		b0
$A+B$	$A+B$	0
	$A-B$	1

		b0	b1
$A+B$	$A+B$	0	0
	$A-B$	1	0
	$A<B$	1	1

		b0	b1	b2
$A+B$	$A+B$	0	0	?
	$A-B$	1	0	?
	$A<B$	1	1	0
	$A\geq B$	1	1	1

		b0	b1	b2	b3
$A+B$	$A+B$	0	0	?	0
	$A-B$	1	0	?	0
	$A<B$	1	1	0	0
	$A\geq B$	1	1	1	0
	$A>B$	0	1	0	1
	$A\leq B$	0	1	1	1

		b0	b1	b2	b3	b4
$A+B$	$A+B$	0	0	?	0	0
	$A-B$	1	0	?	0	0
	$A<B$	1	1	0	0	0
	$A\geq B$	1	1	1	0	0
	$A>B$	0	1	0	1	0
	$A\leq B$	0	1	1	1	0
	$A\cdot B$	0	0	?	0	1

and energy efficiency by moving global synchronization into the switch network.

A prior solution directly applicable to domain-specific computation is the hierarchical interconnect network in a mobile computing accelerator FPGA [6]. The authors build on the trend of outsourcing specific tasks to fixed-function accelerators in mobile computing. These accelerators are idle most of the time (dark silicon), and are active only when their specific function is being executed. The proposal is to create an FPGA that can be configured to execute all accelerator functions at near-ASIC performance, area-, and energy-efficiency. The switch network is organized in stages and each stage has unique switches. The difference between switches of different stages is that they support different level changes. All in all, they still implement a fully-connected network among the LUTs via two-way routing, but if processing elements are placed between all adjacent switch stages and these switches implement routing in only one direction, a banded switch network is created that is suitable for accelerating any data-flow-like algorithm.

III. Architecture

A. CGRA Architecture Overview

Figure 1 shows the VectorCGRA [1], [2] architecture used to evaluate the fused functional units proposed in this paper. It implements a king-mesh-based coarse-grained reconfigurable array (CGRA). The CGRA comprises tiles which are placed in a rectangular grid. Each tile is connected to its eight neighbors using a crossbar switch. Besides the crossbar, all tiles contain a functional unit (FU) that can be configured to execute a predefined set of functions using the configuration memory. By individually setting every tile's function and the crossbar configurations, the architecture can execute a plethora of algorithms. One of this paper's main contributions is the wrapper located around the FU. The general nature of this wrapper enables switching in and out many different types of IPs or user-designed computational elements.

B. Functional Unit Design

1) Fixed-point functional unit: This section shows the design of a generator that can produce functional units (FUs) at multiple levels of configurability. Table I and Figure 1a show the incremental complexity of the FU designs. The first version does not include any configurability, i.e. comprises a single 'add' operation. The next version implements 'sub' in addition to 'add.' Recognizing that comparison ('lt,' 'lte,' 'gt,' 'gte') operations are based on subtraction, the next step is adding these to the FU. Table II shows how comparison operations can be executed using the already present subtractor (adder) hardware.

TABLE II: Executing comparison functions using a subtractor unit.

Function	Computation	Indicator
A LT B	A - B	1 == MSB
A GTE B	A - B	0 == MSB
A GT B	B - A	1 == MSB
A LTE B	B - A	0 == MSB

'lt' is computed by observing the sign (MSB) of the subtraction's result. 'gte' is the complement of this MSB, while 'lte' and 'gt' can be similarly determined by switching the subtraction's operands. The sixth design adds the multiplication operation. The seventh, final, design adds the multiply-and-accumulate (MAC) operation and is shown in Figure 1a. To keep incremental costs to a minimum, multiplication in both designs is implemented using a partial-product multiplier, whose outputs are added by reusing the original adder unit. The multiply-and-accumulate unit uses a 3-to-2 adder to sum the multiplication's partial products and the addend. The 3-to-2 adder's outputs are summarized using the original adder.

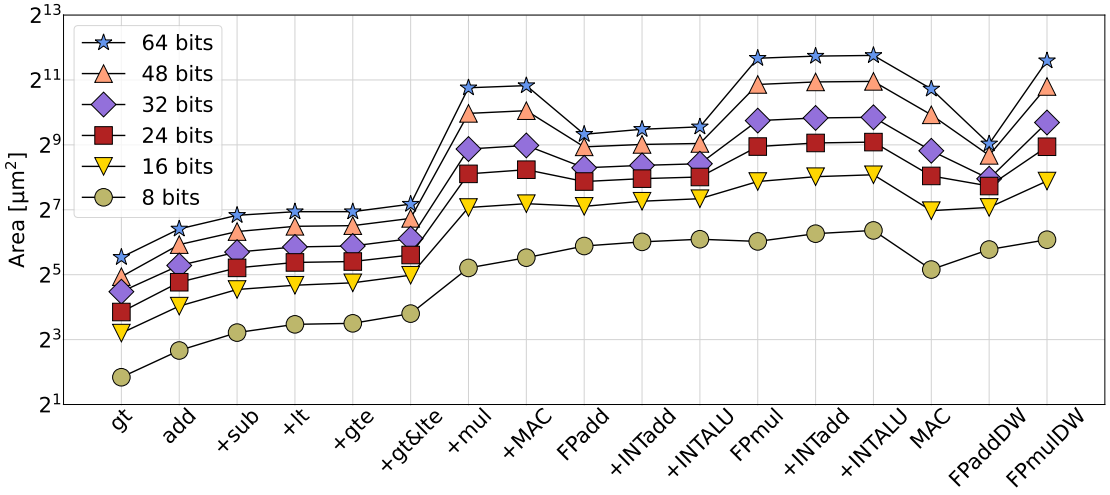


Fig. 2: Area of fixed- and floating-point functional units.

2) Floating-point test units for evaluation: A custom floating-point (FP) adder and multiplier generator are used to evaluate the cost of carrying out fixed-point computations using a floating-point unit's resources. The generator supports custom bit widths for both the exponent and mantissa, therefore not only the IEEE standard precisions (half, single, double, quadruple, octuple), but any other bit width combination can be instantiated. This feature is useful for carrying out experiments, such as running 32-bit fixed-point operations on a custom FP unit that is designed for a 30-bit mantissa. Table III shows the custom bit widths used during the experiments.

TABLE III: Custom floating-point total word lengths, exponent and mantissa bit widths used in the experiments.

WL	Exp	Man	Comment
8	3	4	Quarter-precision (not IEEE).
12	5	6	FP adder executing 8b INT ops.
13	5	7	FP multiplier executing 8b INT ops.
16	5	10	Half-precision (IEEE).
21	6	14	FP adder executing 16b INT ops.
22	6	15	FP multiplier executing 16b INT ops.
24	6	17	Three-quarter-precision (not IEEE).
31	8	22	FP adder executing 24b INT ops.
32	8	23	Single-precision (IEEE), FP multiplier executing 24b INT ops.
39	8	30	FP adder executing 32b INT ops.
40	8	31	FP multiplier executing 32b INT ops.

The units are shown to yield results in the same range as the DesignWare IP blocks, therefore provide a reliable data point for comparisons. Figure 1e and 1d show the custom FP adder and multiplier, respectively. The FP adder's design include fixed-point adders, while the FP multiplier comprises both a fixed-point adder and multiplier.

Figure 1b and 1c show how the previously described fixed-point FU operations can be executed based on these FP-hardware resources. The fixed-point FUs' adder and partial-product multiplier can be shared with those of the FP units. An additional control signal is added to direct the merged fixed-point and FP units' inputs to the fixed-point or FP data path and display the correct result.

C. Switch Design

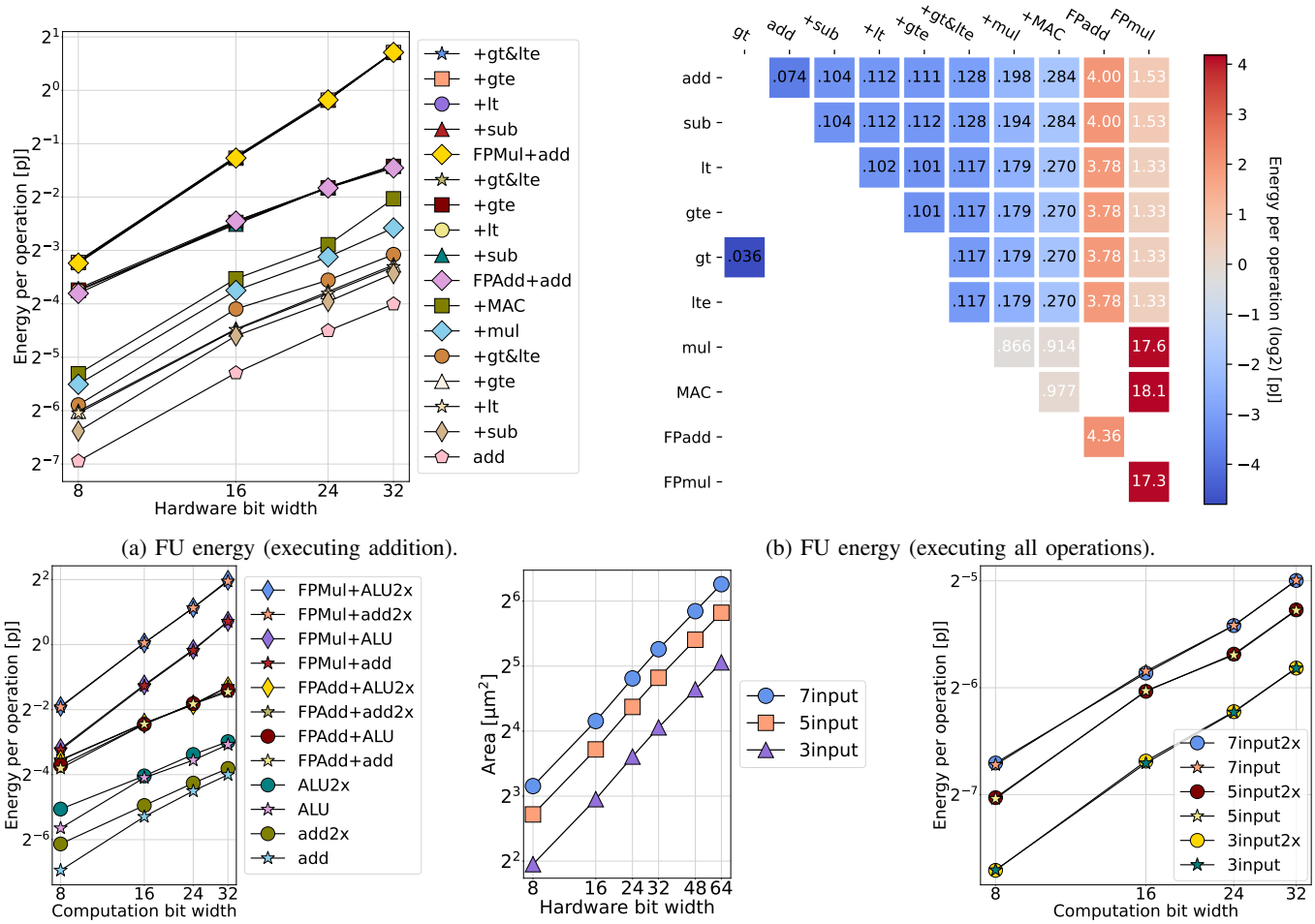
Similarly to the FU, a switch generator is presented to evaluate several switch designs. The switch generator's parameters are input and output count, physical bit width, effective bit width, banding number, and select line encoding. For this evaluation, N-input and $2N$-output switches are chosen, because the majority of FUs have two inputs and one output, and the number of FUs is similar on a particular switch's input and output side.

The physical bit width parameter is the actual bit width of the switch's multiplexers, while the effective bit width is the bit width of the data being routed. Consequently, the effective bit width has to be less than or equal to the physical bit width. The banding number determines from how far vertically a FU input can receive data as explained in Section I.

IV. Methodology

First, we evaluated the fixed- and floating-point functional units. The area, power, and performance results are generated automatically by a Perl script. This allows a large number of designs to be investigated and compared. Based on the predefined configuration parameters, the RTL for the current design is generated. The design is, then, synthesized with a 1-ps clock period, which results in an estimated timing requirement. The time range between 0.85x and 2.5x of the estimated timing requirement is split into twenty equal-length regions. Synthesis is carried out automatically by the Perl script at all twenty clock periods. The resulting twenty data points are manually evaluated to find a good trade-off between area and energy cost.

Other interesting data points are generated by using the post-synthesis netlist, but modifying the original test bench to reduce the bit width of the operands. A scaling factors of 2 is used. For example, a 32-bit adder module is stimulated



(a) FU energy (executing addition).

(b) FU energy (executing all operations).

(c) FU overprovisioning (addition).

(d) Switch area.

(e) Switch energy.

Fig. 3: Area and energy results of the functional units and switch.

with 16-bit data, in addition to the original 32-bit width. These data points are quick to generate since the original full-width design can be used saving the time-consuming synthesis step.

The designs were synthesized and area numbers generated using Synopsys DesignCompiler and a 12-nm commercial PDK. Power numbers were retrieved from Synopsys PrimeTime. Simulations to verify functionality and obtain the VCD switching activity files were run with Synopsys VCS.

We used the previously described VectorCGRA framework to evaluate the efficiency of the fused functional unit. We chose a uniform sixteen-bit granularity throughout the datapath. Since VectorCGRA is implemented in the PyMTL modeling framework, we designed two levels of wrappers around the pure-SystemVerilog functional unit design. The first-level wrapper simply encapsulates the SystemVerilog code to allow compatibility with the PyMTL framework. The second-level wrapper, subsequently, converts the functional unit into a format that makes it compatible with the VectorCGRA framework. Listing 1 shows the generic composition of a PyMTL VectorCGRA wrapper.

Listing 1: PyMTL VectorCGRA wrapper around the PyMTL functional unit.

```
from pymtl3           import *
from ...lib.ifcs      import SendIfcRTL, RecvIfcRTL
from ...lib.opt_type  import *
from ..ALUgenMACRTL   import ALUgenMAC
from ..basic.Fu       import Fu

class ALUgenMACFU( Fu ):
  def construct(..):
    super( ALUgenMACFU, s ).construct(..)

    # RTL component.
    s.fALU = ALUgenMAC()

    s.fALU.op_code //= "fn selection logic code"

    s.fALU.rhs_0  @= s.recv_in[s.in0i].msg.payload
    s.fALU.rhs_1  @= s.recv_in[s.in1i].msg.payload
    s.fALU.rhs_1b @= s.recv_in[s.in2i].msg.payload
    # The wrapped design in SystemVerilog is
    # treated as a blockbox during simulation
    # and synthesis in PyMTL.
    s.send_out[0].msg.payload @= s.fALU.lhs_0
```

V. Evaluation

A. Experimental Setup

We compare six VectorCGRA instances to evaluate the fused functional unit's area efficiency. The baseline designs use a behavioral approach in which the functional units (add, sub, compare, mul, and MAC) are described in PyMTL code. The incremental designs use only the fixed-point fused functional unit for computation. (Both designs implement memory interface units in the left-most column to communicate with the scratchpad memories.) Both the baseline and incremental configurations are realized in two-by-two, four-by-four, and six-by-six VectorCGRA layouts, hence totaling to six data points.

We synthesized the design using Synopsys DesignCompiler and used the Synopsys DesignWare library's partial-product multiplier (DW02_multp) module. We derived the lowest clock period that met the timing requirements of all six designs. We mapped the synthesized netlist to the GlobalFoundries 12-nm cell library. The results of section V-C are produced by Synopsys DesignCompiler using the `compile_ultra -no_autoungroup` command for synthesis, which allows us to report the area of individual instances in the design hierarchy.

B. Functional Unit and Switch Evaluation

1) Area: Figure 2 shows the area cost of increasing the functionality of a functional unit. It is shown that the option of subtraction has some overhead compared to plain addition. The overhead of adding comparison operations is similar, since the adder circuit is not modified and only control blocks need to be added. The exception is the 'gte' operation when 'lt' is already present, since this can be added using a one-bit xor gate (see Table I) which adds negligible cost.

Supporting multiplication results in a big increase, since a multiplier is expensive in terms of area. The already-existing adder can be reused, however, to mitigate the costs. By implementing a partial-product multiplier and adding its outputs using the original adder, the resulting design is one adder smaller compared to the case with a standalone multiplier unit. Once the partial-product multiplier is added, multiply-and-accumulate (MAC) operations can be supported at a negligible cost, as shown in Table I and Figure 1a. A standalone MAC unit's area is shown for reference. The area is dominated by the multiplier, so if a system uses multiple MAC units (e.g., a CNN), complementing those units with integer arithmetic and comparison operators is nearly free, therefore a good way to support reconfigurability at marginal cost.

The custom floating-point adder and multiplier units' are shown to only marginally increase in area when they support integer arithmetic and comparison operations. (The DesignWare floating-point adder and multiplier units are shown for reference.)

Figure 3d shows the area cost of increasing precision and input count of a switch's multiplexer. Area grows linearly both with bit width and input count.

2) Energy: Figure 3a shows the energy-per-operation cost of executing addition on different designs. As far as energy is concerned, executing the same operation is cheapest on the design with no configurability. However, supporting selected additional operations is different in cost. Adding the subtraction option comes at an initial increase, but adding comparison operations comes at a lower cost afterwards.

Executing addition on floating-point units has a high energy cost compared to that of dedicated integer units. However, the cost is nearly identical in all flexibility cases–regardless of what integer operations are supported by the floating-point unit. The floating-point multiplier's energy showcases a steeper increase compared to that of the integer and floating-point adder units, due to the complexity of the design. It is also shown that increasing the bit width results in a linear energy increase in all cases.

Figure 3c shows the energy-per-operation cost of executing addition on designs that either match the precision of the operation, or are twice as wide.

The only significant difference between the over-provisioned and original bit widths is shown for the floating-point multiplier unit. This is caused by the multiplier still switching a large portion of its gates compared to adders and other logic gates when the effective bit width is lower than the actual hardware bit width. The unused blocks can be gated to mitigate this cost when the floating-point unit is used to execute integer operations, but that increases the critical path which is already high in a floating-point unit.

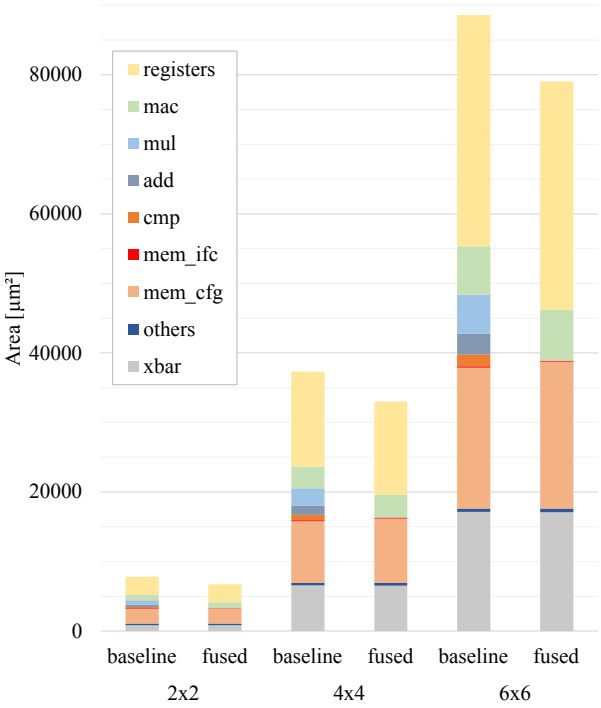
Lastly, Figure 3b shows the energy cost of executing every possible operation on Table I and Figure 1a's designs. The horizontal axis lists the different hardware units. The vertical axis shows what operation is being executed on the selected hardware. The corresponding energy numbers are shown as powers of two. Energy increases as hardware complexity increases. It is also shown that comparison operations cost less energy than addition or subtraction. The biggest increases within the same hardware happen when units that comprise a multiplier (+mul, +MAC, FPmul) execute operations involving multiplication (mul, MAC, FPmul).

Figure 3e shows the energy-per-operation cost of multiplexers at multiple input counts and bit widths. The energy of executing an operation at the same input count and bit width is the same cost in energy when done on a hardware with matching bit width or an overprovisioned one that has a higher bit width than the effective bit width of the executed operation.

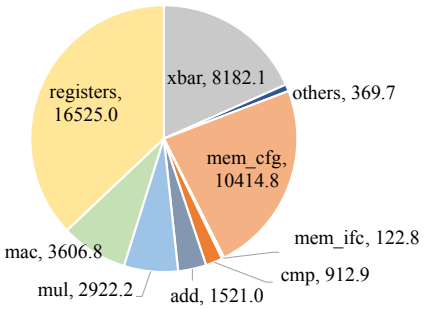
C. VectorCGRA Case Study

Figure 4 shows the area breakdown and comparison across the six different designs described in section IV. All configurations are synthesized at 454 MHz and the GlobalFoundries 12nm technology node.

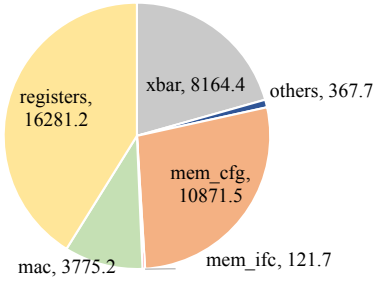
Figure 4a shows the area breakdown by different components for all three VectorCGRA sizes. Figure 4b and 4c show the area of the different components in the baseline and fused VectorCGRA designs. The area numbers are averaged over



(a) 2x2, 4x4, 6x6 VectorCGRA area breakdown.



(b) Baseline VectorCGRA area breakdown.



(c) Fused-FU VectorCGRA area breakdown.

Fig. 4: Area breakdown and comparison of six VectorCGRA instances.

the 2x2, 4x4, and 6x6 designs. Across the three different-sized VectorCGRA configurations, the fused functional unit performs 11–15% better than the baseline instances. Accounting for only the functional units, we see a substantial 58–60% decrease from the baseline to the fused configurations.

As far as scalability is concerned, the total area increases 4.7–4.9-fold from the 2x2 to the 4x4 CGRA, and 11.3–11.8-fold from the 2x2 to the 6x6 CGRA, for the baseline and fused-FU designs, respectively. This is more than the 4-fold and 9-fold increase seen in the number of tiles between the 2x2 and the 4x4 CGRAs, and the 2x2 and the 6x6 designs. The overhead is explained by the relatively higher increase in the crossbar area. Smaller CGRAs are dominated by edge and corner tiles which have significantly less connections than inside tiles. As CGRA dimensions grow, the number of inside tiles starts to dominate, therefore, the increase in area converges to the increase in the number of tiles.

VI. Conclusion

In this paper, we have evaluated the efficiency of multiple fixed- and floating-point functional units and switches. We have concluded that some instructions are possible to support at a negligible cost, therefore they should always be part of the available functions. We have also implemented a methodology to wrap SystemVerilog FUs for use in the PyMTL-based VectorCGRA framework, which resulted in a 58–60% decrease in FU area and an 11–15% decrease across the whole CGRA using our fused units.

References

[1] C. Tan, N. B. Agostini, J. Zhang, M. Minutoli, V. G. Castellana, C. Xie, T. Geng, A. Li, K. Barker, and A. Tumeo, "Opencgra: Democratizing coarse-grained reconfigurable arrays," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2021, pp. 149–155.

[2] C. Tan, D. Patil, A. Tumeo, G. Weisz, S. Reinhardt, and J. Zhang, "Vecpac: A vectorizable and precision-aware cgra," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[3] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 37–47, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1816038.1815968

[4] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 28, no. 1, pp. 7–27, May 2001. [Online]. Available: https://doi.org/10.1023/A:1008155020711

[5] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, March 2015.

[6] F. Yuan, C. C. Wang, T. Yu, and D. Marković, "A multi-granularity fpga with hierarchical interconnects for efficient and flexible mobile computing," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 137–149, Jan 2015.

[7] Y. Li, L. Zichuan, K. Xu, H. Yu, and F. Ren, "A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks," 02 2017.

[8] S. Biookaghazadeh, F. Ren, and M. Zhao, "Are FPGAs Suitable for Edge Computing?" 04 2018.

[9] H. Kung, C. Leiserson, C.-M. U. P. P. D. of COMPUTER SCIENCE., and C.-M. U. C. S. Department, *Systolic Arrays for (VLSI)*, ser. CMU-CS. Carnegie-Mellon University, Department of Computer Science, 1978. [Online]. Available: https://books.google.com/books?id=pAKfHAAACAAJ

[10] R. P. Brent and H. T. Kung, "Systolic vlsi arrays for polynomial gcd computation," *IEEE Transactions on Computers*, vol. C-33, no. 8, pp. 731–736, Aug 1984.

[11] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer, "Piperench: a coprocessor for streaming multimedia acceleration," in *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, May 1999, pp. 28–39.

[12] H. Zijian, C. Xin, and H. Weifeng, "Improved pipeline data flow for dyser-based platform," in *Sixteenth International Symposium on Quality Electronic Design*, March 2015, pp. 135–140.

[13] F. Liu, H. Ahn, S. R. Beard, T. Oh, and D. I. August, "Dynaspam: Dynamic spatial architecture mapping using out of order instruction schedules," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 541–553.

[14] A. Pedram, A. Gerstlauer, R. A. V. D. Geijn, A. Pedram, A. Gerstlauer, and R. A. V. D. Geijn, "Co-design tradeoffs for high-performance, low-power linear algebra architectures," *IEEE Transactions on Computers*, 2012.

[15] C. Zhang, L. Liu, D. Marković, and V. Öwall, "A heterogeneous reconfigurable cell array for mimo signal processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 3, pp. 733–742, March 2015.

[16] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "Aurora: Automated refinement of coarse-grained reconfigurable accelerators," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1388–1393.

[17] G. Mehta, J. Stander, J. Lucas, R. Hoare, B. Hunsaker, and A. Jones, "A low-energy reconfigurable fabric for the supercisc architecture," *J. Low Power Electronics*, vol. 2, pp. 148–164, 08 2006.

[18] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. G. Dreslinski, T. F. Wenisch, and S. Mahlke, "Exploring fine-grained heterogeneity with composite cores," *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 535–547, Feb 2016.

[19] Z. Liu, K. Järvinen, W. Liu, and H. Seo, "Multiprecision multiplication on armv8," in *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*, July 2017, pp. 10–17.

[20] J. v. d. Hoeven, "Multiple precision floating-point arithmetic on simd processors," in *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*, July 2017, pp. 2–9.

[21] C. Tan, T. Tambe, J. Zhang, B. Fang, T. Geng, G.-Y. Wei, D. Brooks, A. Tumeo, G. Gopalakrishnan, and A. Li, "Asap: automatic synthesis of area-efficient and precision-aware cgras," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–13.

[22] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. Horowitz, "Convolution engine: Balancing efficiency and flexibility in specialized computing," *Commun. ACM*, vol. 58, no. 4, pp. 85–93, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2735841

[23] R. Das, S. Narayanasamy, S. K. Satpathy, and R. G. Dreslinski, "Catnap: Energy proportional multiple network-on-chip," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 320–331, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485950

[24] G. Chen, M. A. Anders, H. Kaul, S. K. Satpathy, S. K. Mathew, S. K. Hsu, A. Agarwal, R. K. Krishnamurthy, V. De, and S. Borkar, "A 340 mv-to-0.9 v 20.2 tb/s source-synchronous hybrid packet/circuit-switched 16 x 16 Network-on-Chip in 22 nm Tri-Gate CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 59–67, Jan 2015.