Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing

J.-P. Noel, V. Egloff, M. Kooli, R. Gauchi, J.-M. Portal[2], H.-P. Charles[1], P. Vivet, B. Giraud
Univ. Grenoble Alpes, CEA, LETI, F-38000 Grenoble
[1]Univ. Grenoble Alpes, CEA, LIST, F-38000 Grenoble
[2]Aix-Marseille Univ, Université de Toulon, CNRS, IM2NP, Marseille, France
jean-philippe.noel@cea.fr

***Abstract*—This paper presents a new methodology for automating the Computational SRAM (C-SRAM) design based on off-the-shelf memory compilers and a configurable RTL IP. The main goal is to drastically reduce the development effort compared to a full-custom design, while offering a flexibility of use and a high-yield production. The proposed C-SRAM architecture has been developed to process energy-efficient vector data coupled with a scalar processor, while limiting the data transfer on the system bus. The results obtained by post P&R simulations show that *2RW* and *4RW* C-SRAM configurations using the double pumping technique achieved the highest performance to process vectorized MAC operations compared to the others configurations. Moreover, it has been shown that the impact of the digital wrapper decoding and executing the instructions can be mitigated by increasing the memory cut size to represent less than 10% in area and 20% in power consumption.**

I. Introduction

For edge AI applications processing a large amount of vector data, parallel computing is the straight forward solution. However, due to the transfer of large vector data (up to 512 bits) through the system bus, current commercial vector processors (*e.g.* SIMD) are no longer enough efficient to perform low-energy operations. Indeed, it has been shown that the cost of data transfer between the memories and processing elements is much higher than the operation itself in high-performance computer architectures for advanced technology nodes [1-2]. In this context, In- and Near-Memory Computing (IMC/NMC) architectures defined by a limited data transfer on the system bus seem to be particularly well suited. In the recent literature, several Computational SRAM (C-SRAM) architectures, in which the computing is performed in the bitcell matrix (IMC) and/or at peripheral circuit level (NMC), have been proposed. Most of them, based on silicon results, are full-custom solutions, proposing specific peripheral circuits and/or bitcells complying with logic design rules. This has the advantage of maximizing the energy efficiency of computation, up to several tens of TOPS/W [3-5]. Nevertheless, this design methodology presents two major drawbacks: difficult silicon qualification and low adaptability to specific applications requirements. First, to qualify with a high-yield production a custom-designed C-SRAM IP, specific test circuits (*e.g.* BIST) need to be developed, resulting in major changes to the well-established validation flow. Then, to expand the operating range, several memory sizes need to be qualified, which also requires a major development effort. Finally, to meet EDA tool compatibility requirements in a time-limited design phase, automated generation of C-SRAM views (HDL, physics, timing and power...) need also to be developed. Finally, it is necessary to easily manage a large number of configurations (capacity, form factor, operations...), while guarantying a high-yield production.

In this paper, we propose a new methodology for automating the C-SRAM design based on off-the-shelf SRAM compilers for the storage part and a configurable digital wrapper (RTL IP) for the computing part. The main contributions of the paper are:

- To evaluate the impact, for each selected memory type, the energy efficiency of a Multiply-Accumulate (MAC) operation performed in the digital wrapper.
- To quantify the additional area and energy cost of the computing part (RTL IP).

This work prefigures the algorithm that could be implemented in the automated decision-making system (ADMS) to select the optimal memory type according to the user's constraints (*e.g.* memory capacity, instruction frequency, required operations ...). In order to take full advantage of the parallel computing based on the use of C-SRAM as a vector processing accelerator, the code compiler must also be optimized to execute as many vector-based kernels as possible of a given algorithm. This part is out of the scope of this paper.

The remainder of the paper is organized as follow: Section II demonstrates the benefits of using a C-SRAM as an energy-efficient vector processing accelerator over conventional SIMD processors. Section III describes the proposed methodology for automating the C-SRAM design based on off-the-shelf SRAM compilers and a configurable RTL IP. Section IV presents the simulation results (post P&R) highlighting the correlation between the optimum energy efficiency to perform MAC operations and the number of ports in the selected memory. Finally, Section V summarizes the paper and presents the perspectives of applications.

II. Computational SRAM: an Energy-Efficient Vector Processing Accelerator

The C-SRAM architecture presented in this paper is designed to be implemented with a scalar CPU as an energy-efficient vector processing accelerator thanks to the reduced transfer of vector data on the system bus [6]. To highlight this assumption, Figure 1 provides a comparison between scalar, vector (*e.g.* a SIMD processor) and scalar/vector (based on C-SRAM). Compared to the scalar architecture, the vector architecture proposes parallel

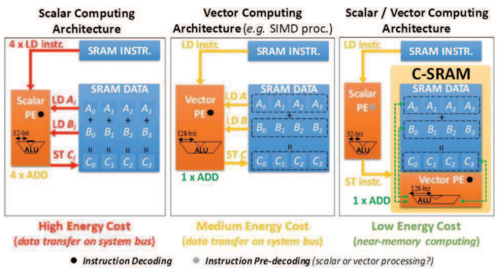

Scalar Computing Architecture
Vector Computing Architecture (e.g. SIMD proc.)
Scalar / Vector Computing Architecture
4 x LD instr.
SRAM INSTR.
SRAM DATA
Scalar PE
Vector PE
C-SRAM
ALU
4 x ADD
1 x ADD
ST instr.
High Energy Cost (data transfer on system bus)
Medium Energy Cost (data transfer on system bus)
Low Energy Cost (near-memory computing)
Instruction Decoding
Instruction Pre-decoding (scalar or vector processing?)

Figure 1 Illustration of scalar (left), vector (middle) and scalar/vector (right) computing architectures. C-SRAM (right case) enables to drastically reduce the data transfer on the system bus (leading to significant energy savings) based on near-memory computing.

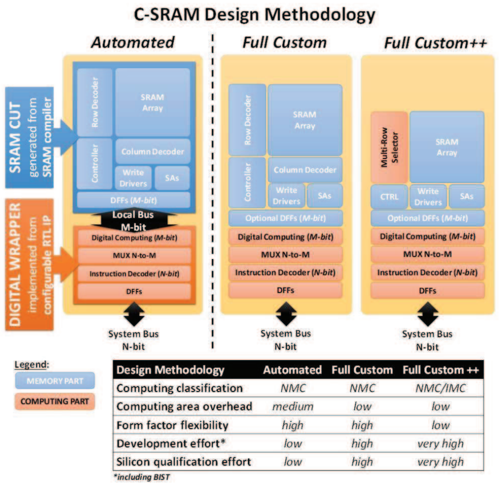
computations on vector data (up to 512-bit for a commercial SIMD processor).

However, it requires data transfers over the system bus towards the registers of the processor. This drawback can be solved by the scalar/vector architecture based on C-SRAM by limiting the data transfer, while keeping parallel computation on vector data. Most of vector operations could be thus directly performed in the C-SRAM (containing the data to be processed) after receiving the instructions from the CPU (up to 64-bit) instead of transferring the data vectors (128-bit or more) to the CPU registers [7]. As for conventional vector processors, the size of C-SRAM operators can be adjusted according to the element length (8-bit, 16-bit, 32-bit ...) of the vector data. The size of the latter are used to define the length of C-SRAM words (128-bit, 256-bit ...).

III. Proposed C-SRAM Design Methodology

A. *SRAM compiler re-use & digital wrapper*

To take advantage of the configurability (size, form factor, power management, ECC ...) and the reliability (silicon qualification) of existing SRAM compilers, while limiting the development efforts, we propose to use a digital wrapper based on a configurable RTL IP (Fig. 2). The wrapper has to decode and execute the C-SRAM instructions sent by the CPU without transferring vector data through the system bus. In addition, this



Design Methodology	Automated	Full Custom	Full Custom ++
Computing classification	*NMC*	*NMC*	*NMC/IMC*
Computing area overhead	*medium*	*low*	*low*
Form factor flexibility	*high*	*high*	*low*
Development effort*	*low*	*high*	*very high*
Silicon qualification effort	*low*	*high*	*very high*

***including BIST*

Figure 2 Proposed design methodology to automate the C-SRAM macro generation w.r.t. full custom design solutions. Automation simplifies the design phase but makes it more difficult to implement analog or pre-computing functions.

wrapper also enables the data vectorization coming from the system bus (up to 64-bit) to the C-SRAM (from 128-bit), and vice-versa. The customizable parameters are the size of the input (depending on the system bus) and the vector data (depending on the application). Then, several operations can be selected from a pre-defined list of a specific Instruction Set Architecture (not described in this paper). Finally, the management of the wrapper pipeline depends on the timing constraints imposed by the frequency difference between the instructions sent by the CPU and the memory accesses.

Another advantage to use existing SRAM compilers and wrap the generated instances with a glue logic is to enable a full flexibility in terms of memory partionning. Due to performance constraints (frequency, power, ...), the memory word length can be limited above the targeted data vector length (*e.g.* 512-bit). In this case, several memory instances with a limited word length

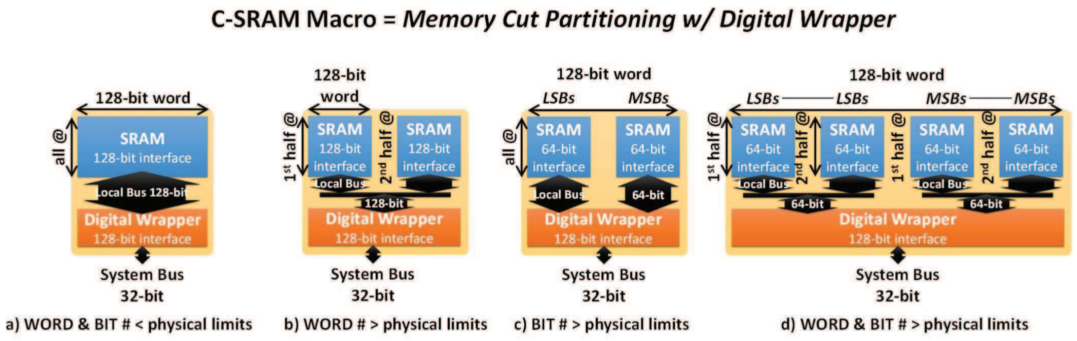

C-SRAM Macro = Memory Cut Partitioning w/ Digital Wrapper
128-bit word
all @
1st half @
2nd half @
LSBs
MSBs
SRAM
128-bit interface
64-bit interface
Local Bus 128-bit
128-bit
64-bit
Digital Wrapper
128-bit interface
System Bus
32-bit
a) WORD & BIT # < physical limits
b) WORD # > physical limits
c) BIT # > physical limits
d) WORD & BIT # > physical limits

Figure 3 128-bit SRAM cut partitioning constrained by the physical limitations: a) word & bit number < physical limits, b) word number > physical limits, c) bit number > physical limits and d) word & bit number > physical limits.

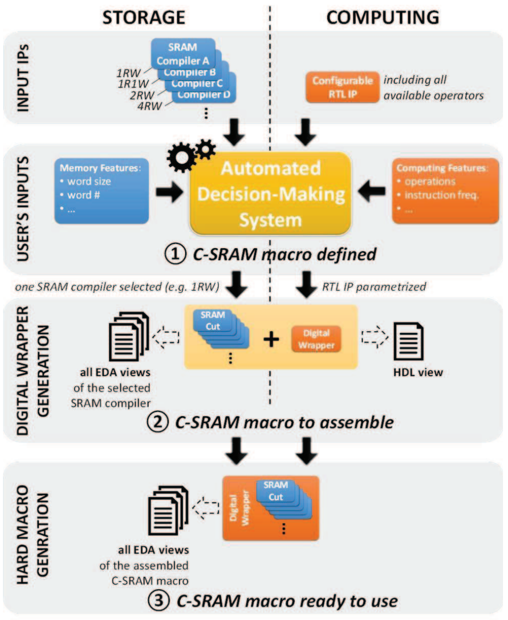


Figure 4 Proposed C-SRAM design flow based on off-the-shelf SRAM compilers and a configurable RTL IP dedicated to decode and execute specific instructions coming from the CPU.

(*e.g.* 32- or 64-bit) can be used to reach the targeted data vector length, while sharing the same digital wrapper to limit area and leakage power penalties [8]. A similar partionning can be done to overcome the limitation of the number of words per SRAM instance, as shown in Fig. 3.

To simplify the development, we propose to divide the C-SRAM design in two parts: storage and computing (Fig. 4). The storage part is based on the use of SRAM compilers to generate the memory cuts. The computing part is based on the use of a configurable RTL IP to implement the digital wrapper. To select the appropriate memory type according to the computing features coming from the user's requests, we propose to use an ADMS. It is based on rules from the results presented in section IV. The algorithm on which it is based is not detailed in this paper.

The outputs of the proposed C-SRAM design flow enable the placement and routing of the digital wrapper with the conventional EDA tools (step 2 in Fig. 4). Nevertheless, the development of this design flow could be pushed one step further to generate an assembled hardware macro with all the EDA views necessary for its implementation (step 3 in Fig. 4).

B. *Multi-port SRAM compiler possibilities based on pushed-rule foundry bitcells*

Three types of SRAM bitcells are systematically density-optimized by the founders: Single-Port (SP), Dual-Port (DP) and Two-Port (TP). These pushed-rule bitcells are optimally designed then qualified on silicon to meet the performance

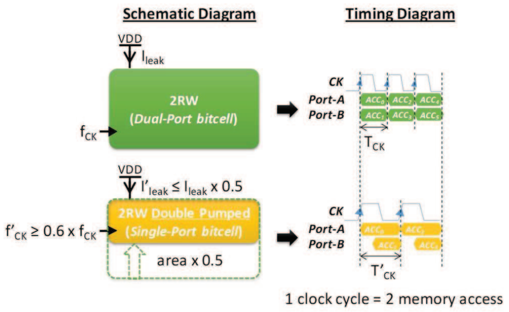


Figure 5 Basic concept of the double pumping technique enabling the port number duplication, while keeping a clock frequency ratio ≥ 0.6 *w.r.t.* the reference solution.

requirements of the process technology. Next, several SRAM compiler types are usually developed, each of them for a specific use. These compilers can be classified by the number of access ports, and more specifically by the number of read / write accesses in a same clock cycle. For example, a *1RW* SRAM compiler means one memory access in one clock cycle, either to write or read a data at a specific address.

To increase the number of synchronous memory accesses, the use of DP or TP bitcells has been generalized. Typically, DP bitcells are used for video processing where multi-access memories (*e.g. 2RW*) optimize the data-parallel processing [9]. While TP bitcells are commonly used as register files where simultaneous read and write accesses (*e.g. 1R1W*) are needed for high-speed computing [10]. In any case, the use of this type of bitcells results in a significant area and leakage overhead (compared to SP-based SRAM). Moreover, to increase the number of ports beyond 2, specific multi-port bitcells are required, further exacerbating this issue. To develop multi-port SRAM compilers without designing specific bitcells or limiting the area and leakage overhead, a circuit design technique, called double pumping, has been adopted by most of the SRAM compiler vendors in advanced technology nodes. This technique consists in artificially duplicating the number of ports by generating a second internal memory clock to start a consecutive memory access without waiting for the end of the first clock.

This is made possible by modifiying decoding and IO circuitry in order to add intermediate sequences of dynamic logic, while reducing the operating frequency (up to ~40%) [11-15]. Nowadays, most of founders and IP vendors propose 2-Port SRAM compilers based on SP bitcells that use this technique to improve both memory density and leakage power consumption at the expense of operating frequency compared to the reference solution (Fig. 5). The more the double pumping technique is optimized, the lower the frequency loss. This technique can also be advantagously used with DP and TP bitcells to achieve up to 4-Port SRAM compilers enabling *4RW*, *2R2W* or *2R2RW* memory accesses [16].

Figure 6 exposes the range of possibilities of SRAM compiler types based on the conventional pushed-rule bitcells (SP, DP and TP) and the use of the double pumping technique for duplicating the number of ports. We will see in the next section

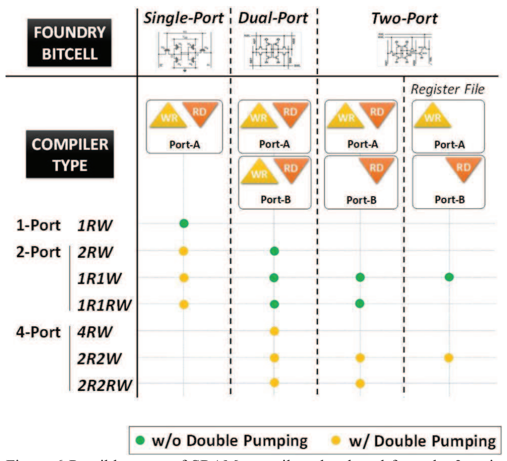


Figure 6 Possible types of SRAM compilers developed from the 3 main pushed-rule foundry bitcells (*Single-Port, Dual-Port and Two-Port*) and the use of the double pumping technique for duplicating the number of port.

the benefits of use 4-Port SRAM compilers, notably those developed from the double pumping technique.

IV. Simulation Results

In this section, we quantify the impact of several SRAM types on the energy efficiency of vector processing. Furthermore, we also quantify the additional area and energy cost of the digital wrapper for several sizes of C-SRAM.

A. *Methodology*

This study is based on the 22nm FD-SOI (22FDX) design platform developed by Globalfoundries. 8-Track standard cell libraries are used (in order to minimize the area overhead) to design the digital wrapper and a set of representative SRAM compilers (all using pushed-rule bitcells described in the previous section) are selected for the storage part of the C-SRAM macros (*1RW*, *2RW*, *2RW-DP* and *4RW-DP*). Simulation results are obtained from post P&R gate-level netlists close to typical conditions (TT/0.8V/85°C).

B. *Testcase: Multiply-Accumulate Operation*

To make the study relevant to edge AI applications, we have decided to base our testcase on a vectorized (16-element of 8-bit) MAC operation. This instruction is executed by pipelining the breakdown operations performed either in the memory (read & write) or in the digital wrapper (multiply & addition). Figure 7 describes the chronogram of the vectorized MAC operation for different memory types of C-SRAM macros. The breakdown operations of this instruction are made of 4 memory accesses (RD A, RD B, RD C, WR Z), 1 instruction decoding, 1 multiplication (A x B) and 1 addition (A x B + C). In this testcase, the operations performed in the digital wrapper are parallelized as much as possible with the memory access clocked at the same frequency ($f_{C\text{-SRAM}}$). The duration of the MAC instruction (latency) can vary between 5 to 6 clock cycles. This variation corresponds to the ability to get A and B operands in the same clock cycle or not, which is the case for *2RW* and *1R1RW* memories (5 clock cycles) but not for *1RW* and *1R1W* memories (6 clock cycles). This figure also describes the chronogram corresponding to the use of a pipeline in the

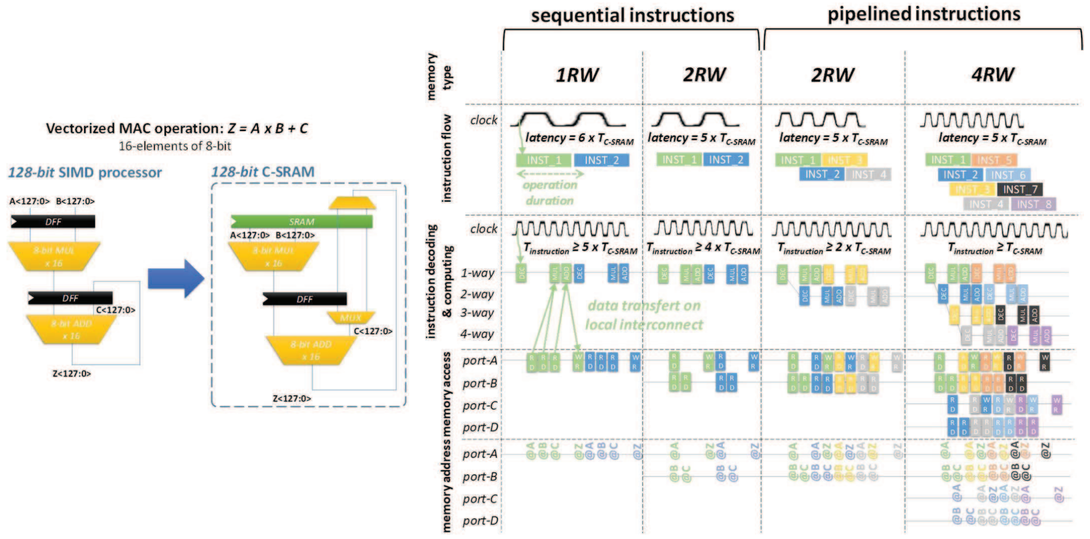


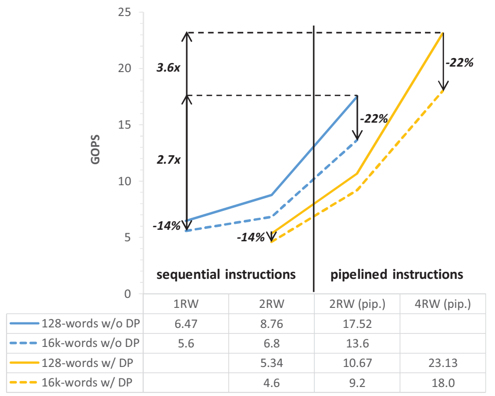
Figure 7 Chronogram of the multiply-accumulate operation ($Z = A * B + C$) performed with various C-SRAM configurations. These configurations depend on the selected SRAM type (*1RW, 1R1W* ...), the frequency ratio between C-SRAM and instruction flow and the way the instructions are performed in the digital wrapper (pipeline enabled or not).

	word length	word #	max. freq. (MHz)	throughput	latency
1RW	128-bit	128	-14% 2021	5-cycles	6-cycles
		16384	1742		
2RW		128	-25% 2190	4-cycles	5-cycles
		4096	1637		
2RW (pip.)		128	2190	2-cycles	
		4096	1637		
2RW-DP		128	-25% 1818	4-cycles	
		4096	1359		
2RW-DP (pip.)		128	1818	2-cycles	
		4096	1359		
4RW-DP (pip.)		128	-22% 1527	1-cycle	
		4096	1189		

extrapolated results

Table 1 Features of selected C-SRAM macros in terms of size (word length and number) and speed (max. operating frequency and throughput/latency of instructions).

digital wrapper, introducing the use of the *4RW* memory. This technique enables to execute successive instructions without waiting for the end of the previous one. Therefore, it is possible to reduce the instruction throughput from 5 cycles to only one. Table 1 describes the different configurations (memory size, maximum operating frequency …) of the C-SRAM macros selected for this study. Note that the results obtained in this section for *2RW-DP* and *4RW-DP* C-SRAM configurations have been extrapolated from simulation of *1RW* and *2RW* C-SRAM configurations based on [11-16]. The maximum operating frequency varies between 1.5GHz to 2.2GHz for *4RW-DP* and *2RW* cuts of 128-word (minimum size) of 128-bit, respectively. The frequency decrease down to 25% for the maximum size of cuts (16k-word for *1RW* and 4k-word for *2RW* and *4RW*). Based on these data, it is possible to get the maximum number of vectorized MAC operations performed by a C-SRAM macro, knowing that 16 MAC operations of 8-bit are performed in parallel (Fig. 8). Duplicating the number of ports of the *1RW* memory and pipelining the instruction execution in the digital wrapper increase the performance up to 2.7x. Furthermore, applying the double pumping technique to the *2RW* memory to get the *4RW* C-SRAM configuration



	1RW	2RW	2RW (pip.)	4RW (pip.)
128-words w/o DP	6.47	8.76	17.52	
16k-words w/o DP	5.6	6.8	13.6	
128-words w/ DP		5.34	10.67	23.13
16k-words w/ DP		4.6	9.2	18.0

***GOPS = maximum C-SRAM frequency** (GHz) **/ throughput** (cycle #) *** 16-operations**

Figure 8 Number of vectorized MAC operations performed by a C-SRAM macro during 1 second (GOPS) for various memory types and sizes (min. and max. macro size).

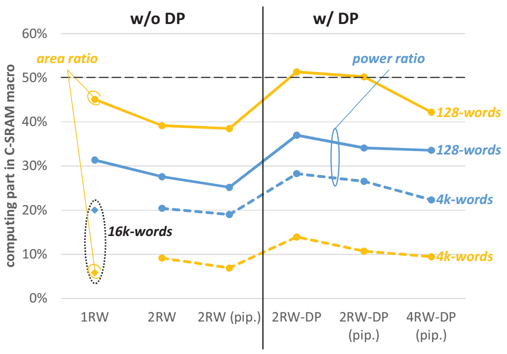


Figure 9 Computing part (area and power ratio) in a C-SRAM macro for various memory types and sizes (min. and max. macro size).

enables to achieve a speedup up to 3.6x. Nevertheless, the use of bigger (128- to 16k-words) and slower macros reduce these performance down to 22%. Figure 9 shows in detail the computing part (included in the digital wrapper) of C-SRAM macros in terms of area and power ratio. As expected, the biggest ratio is achieved for the smallest memories (area: 45% and power: 37%), in particular for those designed with the double pumping technique (area: 51% and power: 37%). The computing part can be reduced by using bigger memory sizes down to 10% and 20% in area and power ratio, respectively. The physical implementation and the memory/computing partitioning are illustrated in Figure 10 for the *1RW* and *2RW* C-SRAM macros representing the smallest and the biggest size allowed by the memory compilers. Figure 11 shows the relative area cost of *2RW*, *2RW-DP* or *4RW-DP* memories compared to a *1RW* memory. As expected, the *2RW* and *4RW* using the

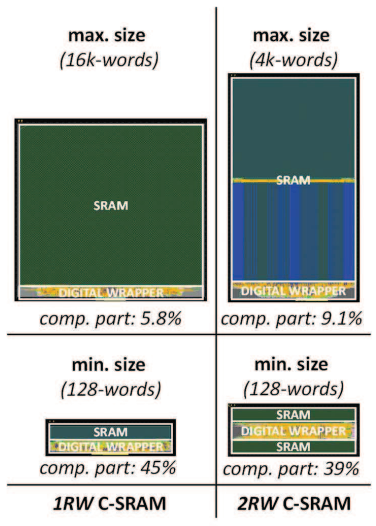
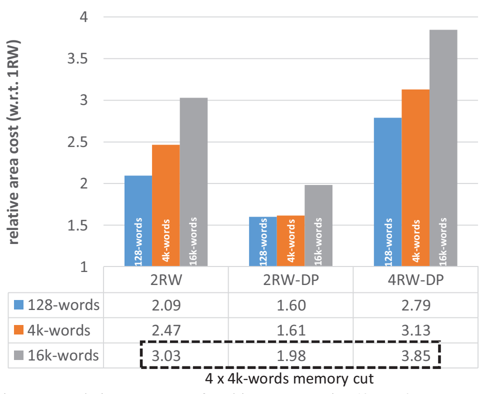


Figure 10 Floorplanning (post P&R) and computing part (%) of *1RW* (left) and *2RW* (right) C-SRAM macros for the minimum and maximum size allowed by the memory compilers selected.



	2RW	2RW-DP	4RW-DP
🟦 128-words	2.09	1.60	2.79
🟧 4k-words	2.47	1.61	3.13
⬜ 16k-words	3.03	1.98	3.85

Figure 11 Relative area cost of multi-port memories (*2RW*, *2RW-DP* and *4RW-DP*) to design C-SRAM macro *w.r.t.* a single-port memory (*1RW*) for various sizes (128- to 16k-words).

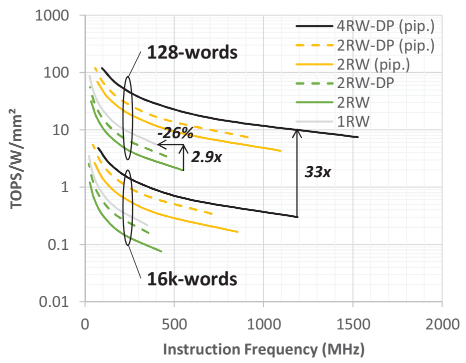


Figure 12 Energy efficiency of vectorized MAC operations per area unit (TOPS/W/mm²) performed by a C-SRAM macro according to the instruction frequency for various memory types and sizes (128- and 16k-words).

double pumping technique enable to limit this cost below 2x (vs. 3x) and 4x, respectively. Figure 12 shows the number of operations (TOPS) for different C-SRAM configurations with a budget of 1W and 1mm² according to the instruction frequency. This illustrates that both the best tradeoff (TOPS/W/mm²) and the highest instruction frequency are achieved by the *4RW-DP* C-SRAM configurations. Regarding the configurations executing sequentially the instructions (*i.e.* w/o pipeline), the *1RW* remains the best solution, with a loss of the instruction frequency of 26% for 2.9x of performance *w.r.t.* *2RW* solutions. In all cases, the performance are drastically reduced (down to 33x) by using the biggest memory size (16k-words).

V. Conclusion

This paper proposed a design methodology for building energy-efficient C-SRAM macros from on-the-shelf memory compilers (based on pushed-rule foundry bitcells) and a configurable RTL IP. The benefit of this approach is to minimize development efforts and production risks using proven design techniques (*i.e.* double pumping) to take advantage of multi-port SRAM compilers (up to 4-Port). It was demonstrated that this C-SRAM configuration is the most energy efficient to perform vectorized MAC operations (16-element of 8-bit) when the instructions sent by the CPU can be pipelined (no data dependencies from one cycle to the next). Otherwise, the *1RW* C-SRAM configuration remains the best choice. It was also demonstrated that increasing the memory capacity significantly reduces the energy efficiency of vectorized MAC operation per unit area (up to 33x). The good tradeoff depends on the user's constraints in terms of area and power budgets as well as system performance (instruction frequency…). The next step in this work is to propose an ADMS to build custom C-SRAMs that match each specific request. This design automation methodology paves the way for the integration of energy-efficient vector processing accelerators into energy-constrained systems using scalar processors (microcontrollers ...).

References

[1] M. Horowitz, "Computing's Energy Problem (and what we can do about it)", *ISSCC*, pp. 10-14, 2014.

[2] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach", 6th edition, 2018

[3] J. Zhang *et al.*, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array", *JSSC*, Vol. 52, No. 4, pp. 915–924, 2017.

[4] Y. Zhang *et al.* "Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security", *JSSC*, Vol. 53, No. 4, pp. 995–1005, 2018.

[5] J. Wang *et al.*, "A Compute SRAM with Bit-Serial Integer/Floating-Point Operations for Programmable In-Memory Vector Acceleration", *ISSCC*, pp. 224–226, 2019.

[6] R. Gauchi *et al.*, "Exploration of a Scalable Vector-based In-Memory Computing Architecture via a System-on-Chip Evaluation Framework", submitted at *DAC*, 2020.

[7] M. Kooli *et al.*, "Smart Instruction Codes for In-Memory Computing Architectures Compatible with Standard SRAM Interfaces", *DATE*, 2018.

[8] R. Gauchi *et al.*, "Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture", *VLSI-SoC*, 2019.

[9] K. Nii *et al.*, "2RW Dual-port SRAM Design Challenges in Advanced Technology Nodes", *IEDM*, pp. 269-272, 2015.

[10] J. P. Kulkarni et al., "5.6 Mb/mm2 1R1W 8T SRAM Arrays Operating Down to 560 mV Utilizing Small-Signal Sensing With Charge Shared Bitline and Asymmetric Sense Amplifier in 14 nm FinFET CMOS Technology", *JSSC*, Vol. 52, No. 1, pp. 229-239, January 2017.

[11] G. S. Ditlow *et al.*, "A 4R2W Register File for a 2.3GHz Wire-Speed POWER Processor with Double-Pumped Write Operation", *ISSCC*, pp. 256-257, 2011.

[12] C.-W. Wu *et al.*, "A Configurable 2-in-1 SRAM Compiler with Constant-Negative-Level Write Driver for Low Vmin in 16nm Fin-FET CMOS", *A-SSCC*, pp. 145-148, 2014.

[13] M. Yabuuchi *et al.*, "A 6.05-Mb/mm² 16-nm FinFET Double Pumping 1W1R 2-port SRAM with 313ps Read Access Time", *VLSI Circuits*, 2016.

[14] Y. Ishii *et al.*, "A 5.92-Mb/mm² 28-nm Pseudo 2-Read/Write Dual-port SRAM using Double Pumping Circuitry", *A-SSCC*, pp. 17-20, 2016.

[15] V. Nautiyal *et al.*, "An Ultra High Density Pseudo Dual-Port SRAM in 16nm FINFET Process for Graphics Processors", *SOCC*, pp. 12-17, 2017.

[16] H. Nguyen *et al.*, "A 7nm Double-Pumped 6R6W Register File for Machine Learning Memory", *VLSI Circuits*, pp. 15-16, 2018.