RISC-V based Programming Model of a Computational SRAM Vector Processing Unit

Maha Kooli, Henri-Pierre Charles, Thaddée Bricout, Jean-Philippe Noel, Maria RAMIREZ-CORRALES, Bastien Giraud

Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France

FirstName.LastName@cea.fr

Abstract

Computational SRAM (C-SRAM) is a new computing solution for Near-Memory Computing. It allows to perform operations inside or next to the memory without transferring data over the system bus, leading to significantly reducing energy consuption. Operations are realized on large vectors of data occupying the entire physical row of C-SRAM array, leading to high performance gains. This paper presents the C-SRAM solution as an integrated vector-processing unit to be used by a RISC-V processor as an energy-efficient and high performing co-processor. The proposed programming model of the C-SRAM is based on the system bus of a RISC-V processor.

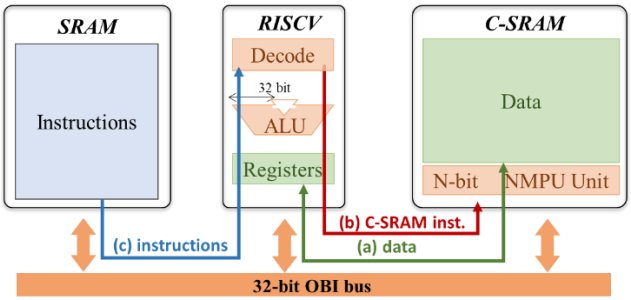
Introduction

The increasing amount of Internet of Things devices and exchanged data motivates the search for energy-efficient computing systems. By separating processing units from memories, traditional Von-Neumann architectures face severe latency and energy issues, limiting the performance of data-intensive applications. Indeed, as processors became faster and memories got denser, a processor/memory performance gap has emerged. To overcome this limitation, Near-Memory-Computing (NMC) is a promising solution since it carries out computations in or as close as possible to the memory.

Computational SRAM (C-SRAM) solution [1-3], allows performing arithmetic, logic and memory operations inside or next to the memory without transferring data over the system bus, which is the most energy-consuming step when computing an operation. Operations are performed on large vectors of data occupying the whole C-SRAM line (not limited by the processor register size such as for existing vector processors). We introduce the C-SRAM solution in this paper as an integrated vector-processing unit to be used and programmed by a RISC-V processor as an energy-efficient and high performing system. We present the programming model of the C-SRAM based on the RISC-V system bus to encode a C-SRAM computing instruction. Thanks to this approach, we are then able to interleave C-SRAM computing instructions and RISC-V computing instructions.

C-SRAM System

C-SRAM system is composed of a RISC-V processor, an SRAM that stores program instructions, and a C-SRAM memory (storing and computing data) connected to a 32-bit system bus, as shown in Fig. 1. The RISC-V core runs the control part of the program, while the C-SRAM runs the main workload. The C-SRAM memory allows different logical, arithmetic and memory operations represented in Table 1.



(a) **Data Transfer:** *32-bit load/store conventional memory access*
(b) **C-SRAM instructions:** *64-bit **STORE** instruction encoding C-SRAM operation & operands*
(c) **Program Code:** *load/store, CPU scalar compute, C-SRAM vector compute, branch, etc*

Fig. 1: C-SRAM System Architecture

C-SRAM RISC-V based Programming Model

In order to integrate the C-SRAM circuit into a conventional system without changing all the system architecture, we develop a communication protocol that defines the interaction between the C-SRAM memory and the RISC-V processor, and manages the transfer of C-SRAM instructions [4].

In order not to change the ISA and the design of the RISC-V, we propose to use the 'STORE' instruction to represent a C-SRAM instruction. We encode the C-SRAM operation and operands inside the registers of the 'STORE' instruction, as shown in Fig. 2. The 'STORE' RISC-V instruction represents then the near-memory C-SRAM operation. When sent to the C-SRAM, the instruction will decoded and then executed by the Near Memory Processing

Unit (NMPU). This solution is flexible and can be easily integrated in existing system architecture, without changing the ISA of the RISC-V processor, which facilitates time-to-market. In addition, thanks to this solution, we can interleave RISC-V instruction and C-SRAM instruction.

Table 1: C-SRAM Instruction Set Architecture.

Category	Width (bits)	Mnemonic	Description
Memory	Line	copy	Copy a line into another
	Line	not	One's complement negation
	8,16,32	bcast	Broadcast 8,16 or 32-bit value to the whole Line
	32,64	hswap	Horizontal 32 or 64-bit word swap
Logical	8,16,32	slli, srli	Shift Left or Right Logical Immediate
	Line	and, nand	Logical AND
	Line	or, nor	Logical OR
	Line	xor, xnor	Logical XOR
Arithmetic	8,16,32	add	Arithmetic Addition
	8,16,32	sub	Arithmetic Subtraction
	8,16,32	cmp	Comparison
	8, 16,32	abs	Arithmetic Absolute value
	8	mul	Arithmetic 8-bit integer multiply
	8	mac	Arithmetic 8-bit integer multiply-accumulate

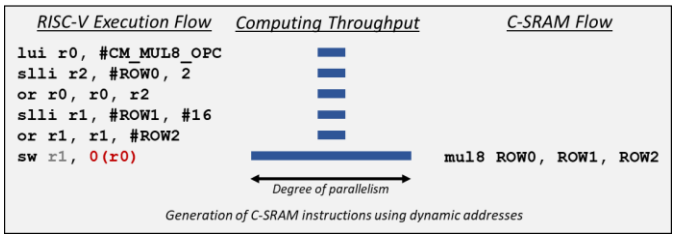


Fig. 2: C-SRAM Programming Model

The proposed programming model is integrated in a 32-bit RISC-V architecture. 32-bit is the minimum size of system bus that permits to interface the C-SRAM as detailed in Fig. 3, where we define three base instruction formats (R/I/U). This permits to classify instructions sent from the CPU to the C-SRAM and decoded by the C-SRAM controller. These formats are defined on concatenated address and data system bus to obtain an ISA aligned on 64 bits (32-bit data bus + 32-bit address bus) in length. Each format refers to a specific set of instructions: R-type groups all instructions where operands address two memory lines; I-type groups all instructions that operates on a 16-bit immediate value and one memory line; and U-type groups all instructions, which applies on a 32-bit immediate value to one memory line. Moreover, the opcode field specifies the operation to perform in the C-SRAM. It is encoded over 8 bits, which allows to perform up to 256 operations. Then, dest, src2 and src1 addresses are relative to the vector addresses of the memory array.

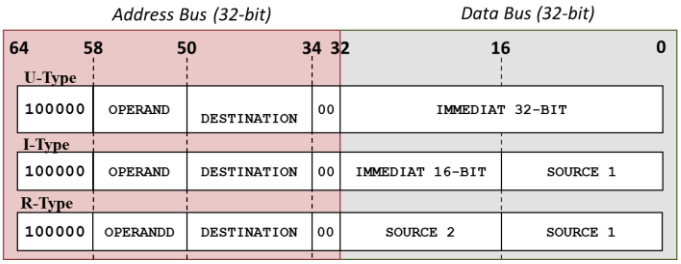


Fig. 3: C-SRAM 64-bit Instruction Formats encoded in 32-bit Address Bus and 32-bit Data Bus

RISC-V based Software Tool Chain

To support the proposed programming model, we have developed a software stack ranging from a QEMU based emulator, a RISC-V C cross-compiler and debugger, a source-to-source compiler used to generate C-SRAM instructions. The RISC-V ecosystem was helpful to go faster and to be flexible. Accessing the software stack of the RISC-V permits to implement the proposed programming model and evaluate different applications in terms of energy and performance gains.

As future works, we propose to integrate our C-SRAM instruction set inside the RISC-V instruction set, thing that is not possible with other hardware platforms. This helps optimizing code generation phase and thus saving execution time.

References

[1] J-P. Noel, et al. 2020. A 35.6 TOPS/W/mm2 3-stage pipelined computational SRAM with adjustable form factor for highly data-centric applications. IEEE ISSCL Letters 3, 286–289.

[2] J-P. Noel et al. 2020. Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing. In IEEE DATE Conference. 1187–1192.

[3] M. Kooli, et al. 2022. Towards a Truly Integrated Vector Processing Unit for Memory-bound Applications Based on a Cost-competitive Computational SRAM Design Solution. J. Emerg. Technol. Comput. Syst. 18, 2, Article 40, 26 pages.

[4] M. Kooli et al. 2018. Smart instruction codes for in-memory computing architectures compatible with standard SRAM interfaces. In IEEE DATE Conference. 1634–1639.