CONVOLVE

Seamless design of smart edge processors

GRANT AGREEMENT NUMBER: 101070374

Deliverable D2.1

Report on the Roadmap

Title of the deliverable	Report on the Roadmap
WP contributing to the deliverable	WP 2
Task contributing to the deliverable	Task 2.1
Dissemination level	RE - Restricted to a group specified by the consortium
Due submission date	01/05/2023
Actual submission date	11/05/2023
Author(s)	Andre Guntoro (BOSCH) Anteneh Gebregiorgis (TUD) Guilherme Pereira Paim (KUL) Gianna Paulin (ETHZ) Jim Garside (MAN) Henk Corporaal, Manil Dev Gomony (TUE)
Internal reviewers	Frank Gürkaynak (ETHZ) Tobias Piechowiak (GAN)

Document Version	Date	Change
V0.1	10.03.2023	Initial document
V0.2	24.04.2023	Draft for review
V0.3	28.04.2023	Draft for incorporating feedback from internal review
V0.4	29.04.2023	Polishing document and minor changes Incorporate standard approaches on image-based perception
V0.9	29.04.2023	PDF ready for submission
V1.0	11.05.2023	Final PDF

Table of Contents

Deliverable Summary

The deployment of machine learning is growing exponentially thanks to vast availability of the data, the compute resources for training, and the advances in training algorithms, along with the achievable performance/accuracy improvement on more complex tasks. In the last past years, various approaches to speed-up Deep Neural Networks (DNN) inferences were developed, ranging from GPU-based (Graphic Processing Unit) solutions which also target embedded devices (such as NVIDIA Jetson), FPGA (Field Programmable Gate Array) frameworks with/including their HLS (High-Level Synthesis) approaches, to dedicated ASICs (Application-Specific Integrated Circuit).

Addressing rapid growing compute requirements and complexity posed by DNNs is challenging, especially on designing hardware accelerators that can fit future trends. In this deliverable, we define the micro-architectural building blocks that are served as building blocks that compose DNNs / CNNs (Convolution Neural Network), viz. the layer types employed in WP1. This method allows us to break down the hardware requirements for our targeted accelerators and can improve flexibility of our accelerators to cope with future demands.

1. Introduction

Emerging and rapid growing Deep Learning (DL) have shown exceptional performance in addressing complex tasks, such as image classification or object detection, and the rapid growth in these methodologies has allowed the accuracy to improve continuously thanks to the more advances in training methods. Whilst the accuracy of such tasks is continuously improving, thanks to the more complex network topologies that have been developed and deployed, it imposes challenges in supporting such inference efficiently on resource-constrained edge devices. Traditional computing platforms such as CPUs and GPUs are not sufficient to infer such networks in constrained systems due to limited power budgets, limited area/volume of the systems, real-time and/or latency requirement, etc. Therefore, for such systems, dedicated hardware accelerators are designed and deployed. Nonetheless, dedicated hardware accelerators have their own limitation to cope with different network topologies and are typically optimized towards specific tasks with specific networks. Therefore, we need to evaluate the network topologies in detail to improve the flexibility of hardware accelerators and utilize this information in devising hardware architecture for such accelerators.

This document "D2.1 Report on the Roadmap " is a deliverable of the Work package No. 2 "Self-configurable modular ULP accelerator blocks", task T2.1 "Roadmap for energy-efficient, reconfigurable, and self-healing functional units " under the task lead of Bosch, sets out the "Report on the roadmap" including challenges in Chapter 2, landscape of hardware accelerators in Chapter 3, our target accelerators in Chapter 4, the basic building block decomposition in Chapter 5, initial standard interface requirements as basis for T2.2 and WP6 in Chapter 6, as well as the summary in Chapter 7.

2. Challenges

The compute and memory demands on inferring neural networks pose challenges on resource-constrained embedded systems. The development of network topologies is mainly driven by the task complexity, and it mainly uses GPUs as basis of computing brain for training the networks, validating the accuracy of the deployed networks, and latter inferring the networks. This is a reasonable approach as most of the pipelines are deployed in GPUs, including optimization techniques such as compressing the networks, quantizing the weights and the activations, etc. to make such networks smaller and embedded-friendly. Nonetheless, deploying such networks on embedded devices requires rethinking of the computing architecture to address the challenges in resource-constraint embedded devices, such as limited power budget, latency, security, etc. In this chapter, we discuss the common challenges when inferring neural networks in such systems.

2.1. Energy Efficiency

One important indicator to evaluate the performance of DL accelerators is their energy efficiency. This is typically visualized as in Figure 1, where on one axis peak power is used as measurement and on the other axis its peak performance in terms of operations per second these accelerators are capable to deliver. The relation of these parameters is at most interesting as it gives its rough performance estimation, and this value is well-known accepted in the community. Thus, we can draw baselines to divide and categorize the efficiency and the limitation of these accelerators, which gives us normalized Ops/W as efficiency indicator. With state-of-the-art hardware architectures that are mainly based on digital CMOS design, 1-10 TOPs/W are typically expected. Some optimization techniques such as data representation (e.g., low bit precision), exploiting sparsity on both weights and activations, data re-use are the key ingredients to achieve better efficiency.

We can also cluster target application segments based on their power consumption based on this visualization. E.g., sub-milliwatt for very-low power system, followed by embedded systems that are targeting around 1-10 watts, autonomous, local data centre, and finally farm infrastructure. Such clustering method varies and is subject to change depending on the definition of system on such clusters itself.

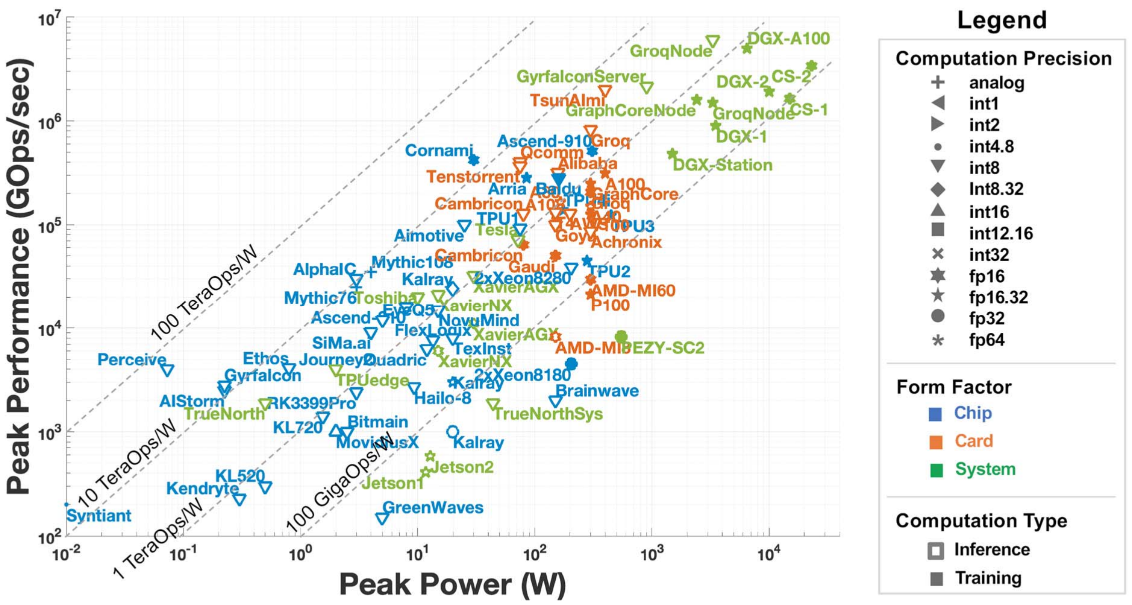
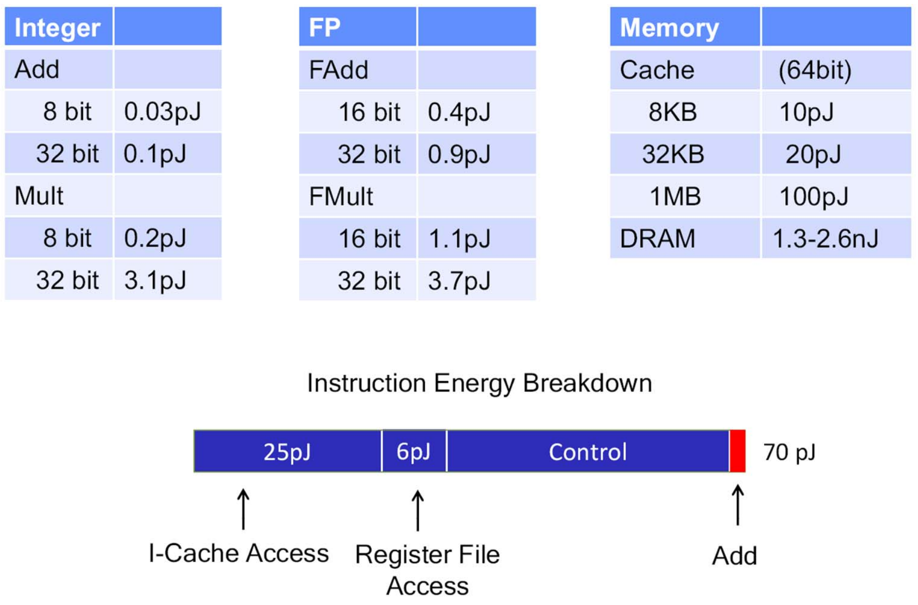


Figure 1. Peak Performance vs. Peak Power of Deep Learning Accelerators and Processors [1].

When we breakdown on the energy consumed on accelerating DL in details [2], we notice that most of the energy consumption is driven by moving data from one point (such as main memory, DRAM) to the accelerator and vice versa. Even for high-precision floating-point FMAC operation, the memory-to-compute relation is more than 2 orders of magnitude, depending on DRAM access pattern/behaviour, technology, etc.

This also means that data locality and data reuse will help to reduce energy consumption of the system. Once we master to exploit these challenges, the emerging memory option to execute the compute in the memory will further allow us to minimize this energy consumption.



Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

Memory	
Cache	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

Instruction Energy Breakdown
25pJ
6pJ
Control
70 pJ
I-Cache Access
Register File Access
Add

FIGURE 2. BREAKDOWN OF ENERGY COST FOR VARIOUS OPERATIONS IN 45NM [2].

2.2. Reconfigurability and Flexibility

As defined in [3], compute system flexibility refers to the invariance of a system's normalized Compute system design means making trade-offs. One important trade-off is between energy-efficiency and application support flexibility. It is a well-known fact that high flexibility and energy-efficiency are not possible to achieve simultaneously, i.e., the higher the flexibility the better the support for diverse applications, but the lower the energy-efficiency.

Actually, compute architecture flexibility, as a measurable quantity, is not well defined. As mentioned in [3], compute system flexibility refers to the *invariance* of a system's normalized performance, energy efficiency, area efficiency (or other secondary metric), *to change of the application*. Therefore, the key goal should be: introducing sufficient architectural reconfigurability such that the normalized system performance, energy- and area-efficiency are hardly affected. However, this requires in-depth understanding of the reconfigurability requirements based on careful analysis of the different applications.

In CONVOLVE, we aim to add architectural support for neural networks of both the ANN (Artificial Neural Network) and SNN (Spiking Neural Network) types. This already is a challenging flexibility requirement as SNNs are event-driven and operates based on dynamic binary spiking inputs as function of time compared to ANNs whose input are more static and often frame-based (i.e., whole input frames are processed, one at a time). In order to stay flexible, we will base this support on a reconfigurable architecture template, like a CGRA (Coarse Grain Reconfigurable Array) and/or a flexible many-core architecture. Their main difference is that CGRAs typically contain a set of function units, like Adders, Multipliers, and Load-Store units, as building blocks for constructing a processing array, while a many-core is based on connecting many (specialized) instruction-set processors, like RISC cores, using e.g., a mesh NoC (Network-on-Chip). So, their difference mainly concerns the granularity and programmability of the used building blocks. In both cases the building blocks can be extensively specialized for ANNs and SNNs, or other application domains. Intermediate solutions, using e.g., multi-core CGRAs are also possible. More on this in Chapter 4.

We foresee many challenges, including:

1. What is the right hardware template architecture for building accelerators?

2. Can we combine support for SNNs and ANNs in the same instantiation of the templated architecture, or do we need separate instantiations? This is far from clear, since ANNs use typically frame-based processing, while SNNs are event-based in nature.

3. What is the degree of reconfiguration we need? Clearly there is a trade-off between static and dynamic reconfiguration options, where static is more energy efficient while dynamic can easier adapt to dynamic workloads. The latter becomes important since DL computing becomes more dynamic, with adaptable networks, early exits, etc.

4. What support is given for going really ultra-low power, and which power management options are available?

5. Find a suitable programming model and accompanying highly optimizing compiler for application acceleration while exploiting all hardware features of the template.

These challenges will be addressed within the CONVOLVE project.

2.3. Data Representation

Compared to CPU and GPU approaches which can utilize higher precision formats such as 32-bit single-precision floating-point to infer CNNs, lower-precision is favourable on resource-constrained embedded systems due to lower area overhead, lower energy consumption, and higher achievable operating frequency.

The early hardware implementations to infer CNNs utilized 16-bit fixed-point arithmetic, such as DianNao [4] and its derivatives [5][6]. The first generation of Eyeriss [7] also implemented 16-bit fixed-point arithmetic while the second generation Eyeriss v2 [8] is optimized for 8-bit for activations and weights. This approach is also followed by TPU (Tensor Processing Unit) developed by Google, that is designed for a high volume of low-precision 8-bit computation. On the other end, for very specific tasks such as low-resolution classification, arithmetic optimization down to binary was also investigated and developed, such as BNN [9]. In [10], we have studied and experimented, that deploying networks with lower precision (e.g., less than 8-bit) would require post-training or fine-tuning of the networks to recover the accuracy loss and depending on the task complexity, some percentage of quality loss must be taken into account. The key message here is that we need to define our reference data representation that works well among the applications that we are targeting in CONVOLVE project. This exploration will be carried out in conjunction with use-case optimization in WP1 and algorithm development in WP4.

In addition to arithmetic precision, model compression through pruning can considerably improve performance. However, random sparse computation can be complex to handle in hardware. Structured sparsity can help with a known sparsity pattern making the hardware design easier. Though the accuracy of the models with structured sparsity needs to be evaluated, they can benefit from sparsity-aware training. Some surveyed platforms have used sparsity (unstructured and structured), showing improved performance and should be a future trend for hardware design.

2.4. Compute-in-Memory

Computing-In-Memory (CIM) architectures, where computational tasks are performed within the memory itself, eliminate unnecessary data movement. Consequently, these architectures address the memory bottleneck issue as well as provide higher data parallelism. CIM can be an analog or digital CIM [11]. In digital CIM, the SRAM bit cells are used to store the weights and the input value is used to activate one row at a time and multiplication and addition operations are performed very close to the bitcell as shown in Figure 3. In analog CIM, multiple memory rows containing the weights are activated simultaneously, and the resulting current from each row is summed up together to produce an output voltage that is converted to digital domain using an Analog-to-Digital Converter (ADC) resulting in the sum-of-product value of the weight and input. Memristive based CIM uses similar principles of current summation as in analog CIM based on SRAM except that the memristive elements are programmed with a transconductance value corresponding to the weight value. Despite very high energy efficiency compared to conventional compute architectures, CIM faces several device technology and design challenges. This section highlights the key non-ideality and design challenges, such as non-ideality, data-conversion overhead, application mapping, data-flow reconfigurability, cross-layer modeling, and design automation, that hinder the widespread deployment of CIM architectures.

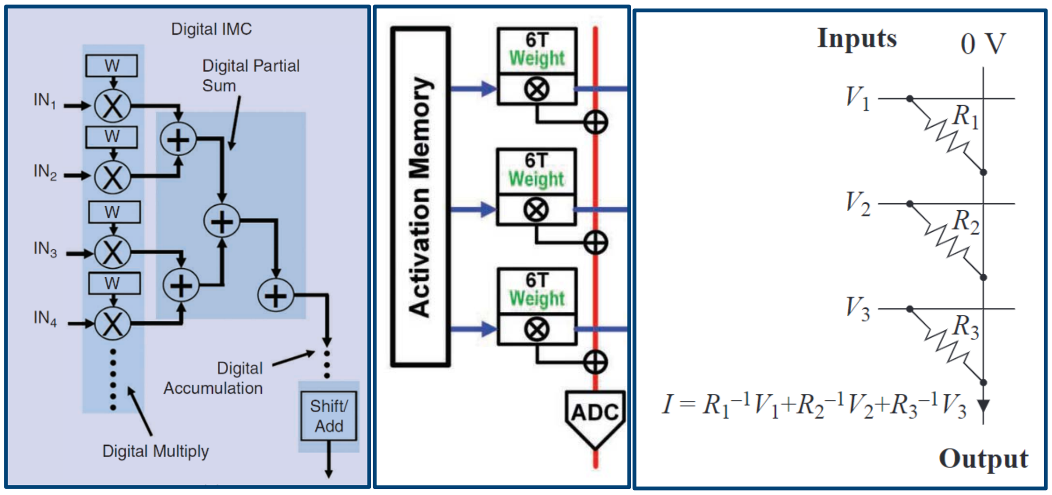


FIGURE 3. DIGITAL CIM, ANALOG CIM AND CURRENT SUMMATION OPERATION IN A MEMRISTOR BASED CIM.

2.4.1. CIM Design Challenges

Non-ideality: Below are some challenges specific to non-ideality posed by CIM architectures.

1. Variation: Variation is the deviation of the resistance value of the memristor after programming from the expected resistance value, which can lead to incorrect computations [12]. Variation happens mainly due to fabrication imperfections and the stochastic nature of underlying physics. Additionally, traditional CMOS process, voltage and temperature (PVT) variations further impact the computational correctness.

2. Wire Parasitic: Due to the erroneous outputs [13]. For instance, in logic operations, the reference and input signals reaching the sensing circuits (for e.g., sense amplifier) suffer from delay mismatch caused by different critical paths. Additionally, the wordline (input voltages shown in Figure 5) degrades along the path reaching farther columns that degrades the associated current output.

3. Non-Zero G_{min} Error: In the digital domain, multiplying any non-zero input with a zero weight must result in a zero output. However, when such computation is mapped to memristors a non-zero output current is produced when a non-zero input voltage is applied to a memristor with G_{min} conductance which represents digital zero. This phenomenon is known as non-zero G_{min} error which causes a functional mismatch between the expected digital value and the actual memristive computation result [14].

4. Endurance: Memristors suffer from limited endurance due to the destructive nature of the programming operations. For instance, in RRAMs, the material assumes presence and absence of conductive ions by forming and rupturing the conductive filament during

a write operation. However, continuous write operations gradually degrade the ON/OFF resistance ratio of the devices, eventually leading to endurance failure.

5. Device Degradation: Due to stress and ageing, CMOS periphery and memristors in CIM suffer from device degradation [15]. These phenomena are aggravated by high voltage of operation and temperature.
6. Read Disturb: Read disturb is a phenomenon where the data stored in the cell is flipped by the read operation.
7. Conductance Drift: The conductance states of the memristors tend to drift with time and can eventually lead to unwanted bit-flips [39].

Data conversion overhead: Since SRAM based analog and memristive CIM uses fundamentally analog operations, interfaces for connecting digital and analog parts are required and represent a critical part of the design. Digital-to-Analog Converters (DACs) and ADCs are crucial components to handle the CIM inputs and outputs, respectively. Conversions performed by ADCs are very critical and challenging due to (1) Analog signals have low noise margin and hence, can lead to erroneous output; (2) Analog computation heavily relies on memristors and CMOS selectors strength (e.g., for 1T1R), therefore these variations induce variation in output current; (3) Quantization error in ADCs increases as the number of activation levels increases for higher. Moreover, ADCs usually occupy significantly large areas and consume significant power when compared with the total area of the CIM. Figure 4 shows that the ADC alone typically dominates CIM die area (>90%) and power consumption (>65%); this highlights the challenge ADC design poses for CIM.

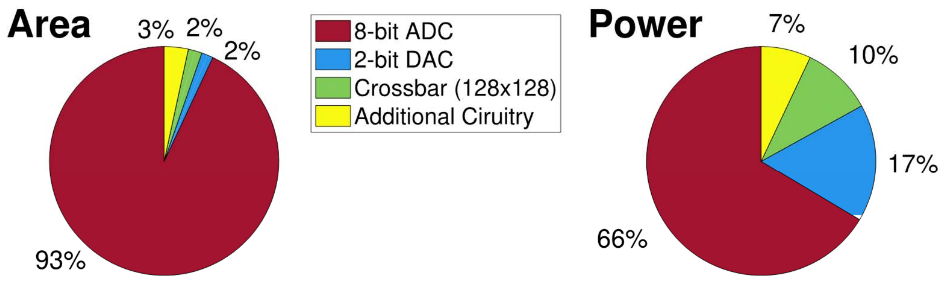


FIGURE 4. AREA AND POWER CONSUMPTION SHARE OF CIM DESIGN BLOCKS.

Optimal Application Mapping: Existing CIM research mostly focuses on circuit-level optimizations to maximize the macro efficiency and its *peak performance*. Yet, in reality the system efficiency also strongly depends on the mapping efficiency. This mapping efficiency quantifies how well a particular workload can exploit the peak computational resources and memory resources of a DL processor. This performance metric strongly depends on both the system architecture as well as the targeted workload, requiring a holistic system-level view of the complete compute system, its memory hierarchy, its dataflow, and its mapping strategies.

Dataflow Reconfigurability: A single dataflow cannot support the efficient mapping of a wide variety of workloads and neural network layer types. Supporting multiple dataflows can increase the real efficiency at the system level due to the better mapping capabilities. However, higher dataflow flexibility also tends to increase the CIM circuit-level overhead. Therefore, it is a challenge for CIM to find the optimal dataflow flexibility trade-off.

Cross-layer Modelling: To maximize the mapping efficiency, an all-encompassing system model is required to be capable of grasp at the same time low level circuit effects, as well as system consequences and mapping influences. A reusable, configurable, and standardized modelling framework/methodology is required for driving the circuit-level CIM design decisions though system-level explorations.

Design Automation: Current CIM design methodologies are extremely time intensive, due to their relying on custom design. Due to the time-to-market (TTM) and engineering hours, this precludes pushing forward its industrial adoption. More automated design of such architecture can push market adoption and stimulate rapid innovation.

3. Landscape of Hardware Accelerators

There are many ways of designing computation engines. The figure below shows several compute architecture options with a rough indication of their flexibility versus performance trade-off (see also Section 2.2).

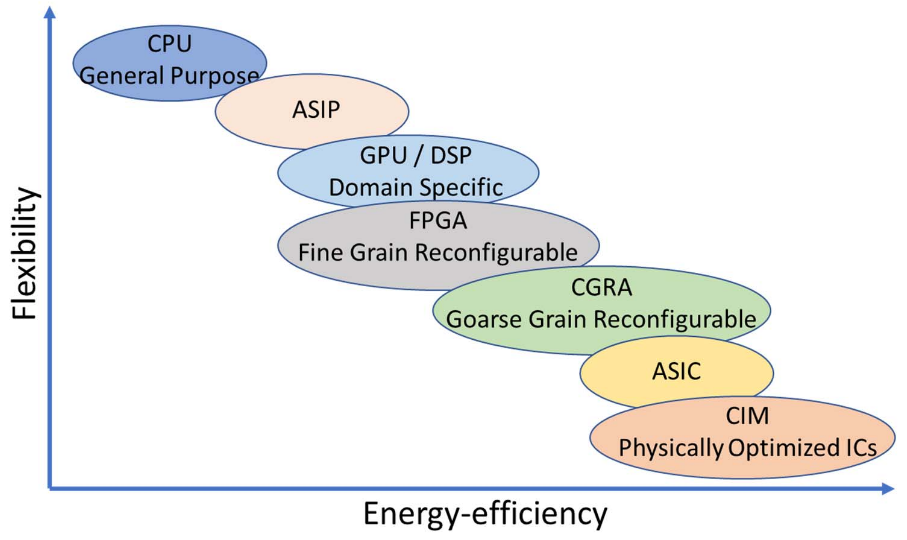


FIGURE 5 LANDSCAPE OF HARDWARE ACCELERATORS.

CPUs, like the RISC-V (see Section 4.1), are quite generic, and therefore flexible (see [16] for a quantitative definition on flexibility). However, their energy-efficiency (measured e.g. in pJ/operation) is very low. Specializing pays of e.g., GPUs and DSPs (Digital Signal Processors) are tuned for signal and graphic processing domains. GPUs are also used a lot for Deep Learning, since they contain a massive amount of MAC (Multiply-Accumulate) units. FPGAs are different. They are based on millions of logic blocks, which can be flexible (programmable) connected, and where each block can be configured to mimic a couple of gates. One can argue that they are very flexible, but their efficiency is only good if there is a right match with the application fine grain level of granularity. We expect CGRAs (Coarse Grain Reconfigurable Array; Section 4.2) to be far more efficient when the granularity of operations is coarser. Still, they are quite flexible since they can be freely programmed and their building blocks (like Multiply-Add units) reconfigurable connected. ASICs can be extremely efficient but are hardly programmable. In best cases we can configure a couple of parameters. CIM architectures go even a step further. They typically change the physical memory generators by adding support within the memory cell periphery, typically in the memory read-out circuitry (but it could also be in the wordline selection circuitry). Both digital and analog variants of these CIMs exist. They can potentially be extremely energy efficient.

However, the more specialized the architecture and the implementation, the lower its programmability and/or reconfigurability, and therefore its flexibility. In the CONVOLVE project we research and implement a compromise. We start with a very flexible platform (see WP6), containing one or more RISC cores (likely of the RISC-V type, see Section 4.1). This platform will be specialized by adding several accelerators of the CGRA type (Section 4.2) ,and even more specialized ones, like SRAM and RRAM based CIM (Section 4.3.2 and 4.3.3) engines. By embedding these accelerators in a generic SoC (System-on-Chip) template we aim at getting the best of both worlds: sufficient flexibility, while being highly energy-efficient.

4. Targeted Hardware Accelerators

In CONVOLVE project, we address the hardware challenges in deploying machine learning on edge devices where the systems are resource-constrained in many aspects such as system size or dimension, power budget, privacy, etc. Here, we will be working with different hardware architectures, ranging from flexible RISC-V software-centric accelerator to dedicated hardware accelerators with CGRA structure and CIM paradigm.

4.1. RISC-V

As previously mentioned, one of the main challenges in designing hardware accelerators to infer DNNs is flexibility. The algorithms in machine learning evolve rapidly, which make dedicated hardware accelerators that are specifically designed for certain domains less efficient. Software-centric approaches based on RISC-V offer greater flexibility that can greatly complement the highly efficient hardware accelerators that we address in the CONVOLVE project. Here, along with hardware-centric accelerators that are addressed in next sections, we devise the improvement that will be carried on RISC-V.

As power and die size play also important role on edge devices, we selected Snitch [17] core as RISC-V subsystem implementation, as depicted in Figure 6.

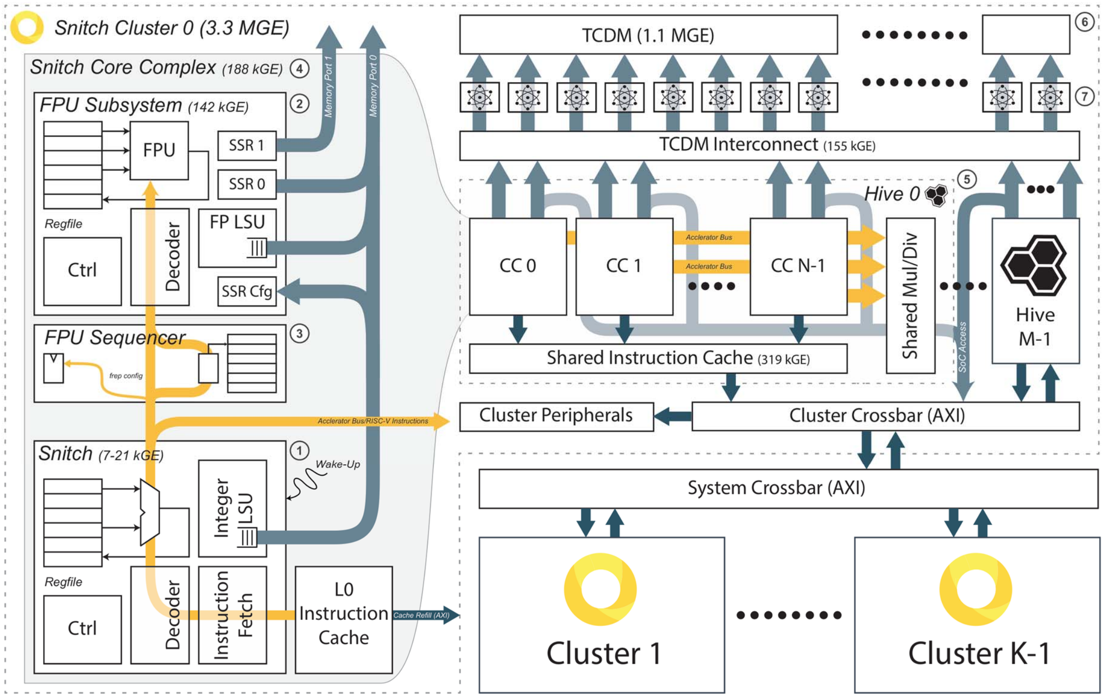


FIGURE 6. OVERVIEW OF ENTIRE RISC-V SNITCH SYSTEM [17].

Snitch core is a single-stage out-of-order variant of the RISC-V processor architecture, that is designed to provide enhanced security and privacy features for use in Internet of Things (IoT) devices and other applications where security is critical. The Snitch core implements several security features to protect against a variety of threats such as side-channel attacks, information leakage, and tampering. Additionally, the Snitch core is also configurable to enable performance and efficiency improvement, such as supporting dynamic voltage and frequency scaling, which can help to reduce power consumption and extend the battery life of edge devices.

Depending on the design configuration, Snitch core occupies only between 9 kGE (Gate-Equivalent) and 21 kGE with all mandatory integer based (RV32I) specification. The complex subsystem is already designed to enable multiple instances of cores to improve parallelization which is needed to infer CNNs in real-time. Additionally, FPU (Floating-Point Unit) is already integrated in the core subsystem to enable us to run complex networks such as LSTM (Long Short-Term Memory) and Transformer networks.

Within the course of WP2, we will include more features on Snitch core to further enhance its flexibility and performance when inferring CNNs, such as:

1. Design space exploration to support lower-precision of floating-point format such as 8-bit floating-point. Here, two *standards* are proposed: fp<1,5,2> and fp<1,4,3>. The first number denotes the number of bits used for sign, the second for exponent, and the latter for mantissa.

2. In addition to floating-point, we plan to investigate alternative format such as posit, where in certain extent it outperforms 8-bit floating-point accuracy with the cost of area.

3. Design space exploration to extend the Snitch core ISA to improve parallelization inside the Snitch core itself, such as sub-word permutations.

4. Coupling digital-based CIM into the Snitch core pipeline to further reduce energy consumption on Matrix-Vector-Multiplication (MVM) operations, which are the main ingredients of CNNs.

4.2. CGRA

CGRAs (Coarse Grain Reconfigurable Arrays) consists of a homogeneous or heterogeneous grid of programmable PEs (Processing Element) and a static reconfigurable network that can pass data to neighbouring PEs [18]. An example is shown in Figure 7.

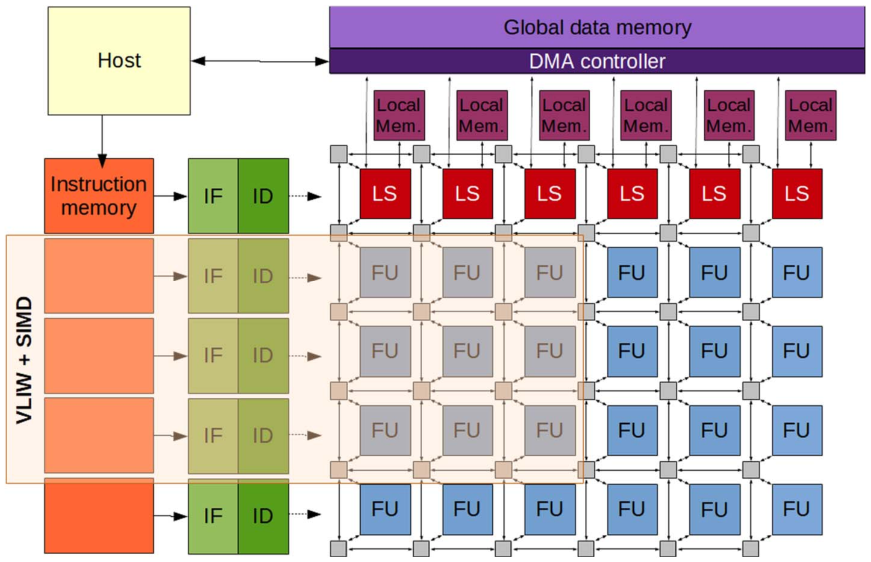


FIGURE 7. AN EXAMPLE OF CGRA CONTAINING 6 LOAD-STORE UNITS AND MANY OTHER FUNCTION UNITS (FUS).

In the example a PE is a function unit, like a MAC, Decoding, Register File, or Load-Store unit. Units can also be more complex and coarse grain. These CGRAs can be flexible configured to support one or more processors, where each processor supports multiple issue slots, and issue slots can contain vector (SIMD) units.

There are also CGRAs where each PE essentially operates is a small independent RISC core, and has its own instruction memory and decoding stage, ALU and register file. Some PEs have a load-store unit to access the memory that is shared with the host core. On top of exploiting Instruction-Level-Parallelism (ILP) inside each PE. Its instruction network allows the CGRA to combine multiple PEs into a virtual SIMD-unit to exploit data-level parallelism (DLP).

CGRAs have several advantages:

1. They are highly parallel (containing many units), supporting parallelism at various levels (data-instruction-/operation-, pipeline, and often even task-level).
2. They have a high area efficiency, due to its coarseness (as compared to e.g., FPGAs).
3. Energy efficiency is also good, due to spatial computation using static configuration of the communication; also, the computation itself can often be statically mapped.
4. Finally, they are very flexible, supporting all kinds of computation (unless they are very specialized for a certain domain). This wide applicability gives them the potential advantage of high-volume production with resulting low cost.

CGRAs also have major challenges; to mention a few important ones:

1. Typically, its switching fabric is expensive. Here we can learn from FPGA interconnect, with short and long wires, and by avoiding full crossbar switchboxes. Another way to address this issue is by designing the PEs coarser granular, i.e. PE is capable of performing larger kernels. This may also be beneficial to efficiently support SNNs in addition to ANNs in the same hardware platform.
2. Another challenge is to determine the right architecture for CGRAs. Some recent efforts focus on automated design-space-exploration (DSE) to find a good CGRA architecture for a given set of applications [19]. We will use the DSE framework described in D6.1 for performing the design-space exploration. Note, that a CGRA may even include CIM units, as detailed in Section 4.3.
3. Perhaps the biggest challenge in CGRA research has been to find a suitable programming model and accompanying highly optimizing compiler for application acceleration while exploiting all hardware features of the CGRA. One well-explored, although restricted, approach is to map a static dataflow graph to the CGRA, while executing the application control flow on a tightly coupled host processor [20]. Another promising approach is to model the CGRA as an exposed data-path architecture (EDPA) to reuse existing compiler developments for VLIW (Very Long Instruction Word) / TTA (Transport Triggered Architecture) processors [21].

All above areas will be explored in the CONVOLVE project.

4.3. Compute-in-Memory

4.3.1. SRAM-based Digital CIM

The state-of-the-art SRAM based CIM proposed in [22] offers high energy and area efficiency will be used as the baseline architecture. The high-level architecture of CIM macro shown in Figure 8 uses fully digital 12-T SRAM, without sense amplifires and write drivers on bit lines, reducing the energy consumption for read and write operations. The size of the CIM macro is 64kb and consists of 64 1kb MAC arrays and each 1kb MAC array supports 64 4bx4b multiplication and accumulation, with 64 4b input and 14b output. Each MAC array consists of SRAM bitcell array, bitwise multipliers (NOR), 6 stages adder tree, bit shifter and accumulator. During a MAC operation, every clock cycle a single bit input value is multiplied with 4b weights stored in SRAM cells. Hence, total 4 cycles are required to complete 4b*4b multiplication. Partial sums of multiplications are added together by 6 stages adder tree.

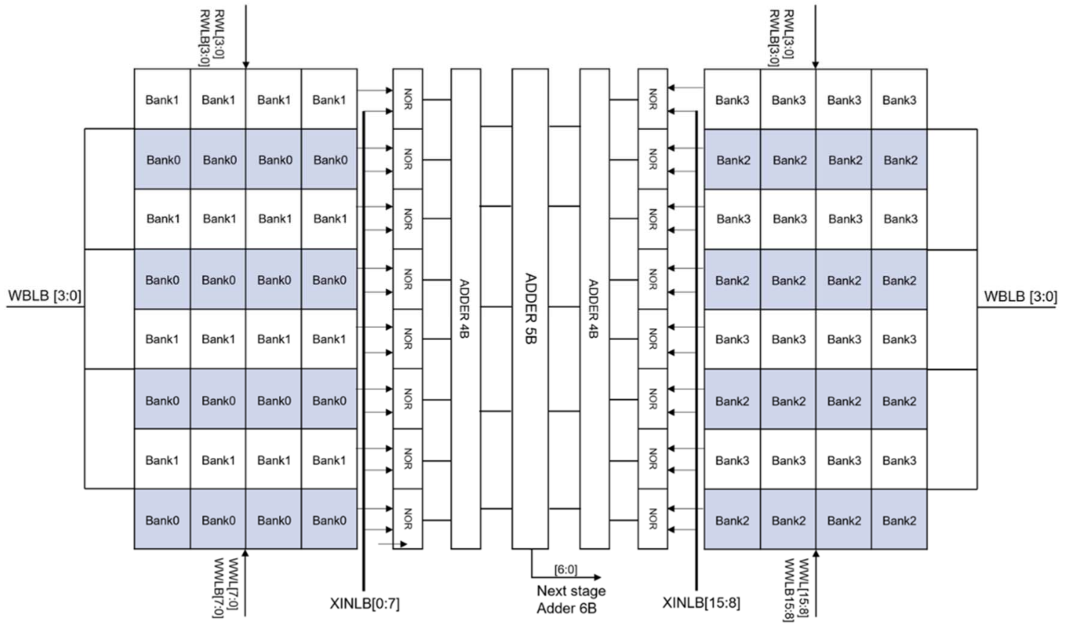


FIGURE 8. SRAM-BASED DIGITAL CIM ARCHITECTURE

In the above digital CIM architecture, the majority of the power is consumed in the adder tree and bitwise multiplication operations. Hence, we will focus on reducing the energy consumption of partial sum additions with novel techniques. One approach to be explored is the use of smaller look-up-tables to minimize activity in the adder tree similar to the technique proposed in [23]. However, keeping the area overhead of the LUT size to the minimal would be the challenge and we will use custom designed LUT cells. Another approach is to optimize the adder tree by utilizing energy and area efficient adders in the adder three hierarchy. In addition, we will explore the possibility of using body-biasing to reduce the leakage of the rows that are not being assessed. This needs to be done without adding the overhead of additional power supplies. Finally, we will also explore a digital-on-top design flow for quickly designing large SRAM based digital CIM and integrate with custom digital logic. This will include designing custom SRAM bit cells with integrated bitwise compute capabilities that will be instantiated as a library component along with the technology vendor provided standard cells in a digital-on-top design flow. Since the digital-on-top flow allows faster design space exploration, we will explore different architecture configurations (i.e. different CIM macro sizes) to achieve the highest Power Performance Area (PPA).

4.3.2. CGRA using SRAM-based Digital CIM

The selection of the supported dataflows at the circuit-level (macros) can dictate the efficient support of different ML (Machine Learning) layers on these hardware accelerators at the system-level. Therefore, we plan to develop a coarse-grained reconfigurable array (CGRA; see Section 4.2), where each compute grain is built from small SRAM-based CIM accelerator blocks. With this architecture, we aim to achieve higher system-level efficiency in combination with dataflow flexibility. In this accelerator, we plan to optimize both the design time parameters such as the number of input/output channels (ICH and OCH in Figure 9), as well as the supported dataflow and its runtime management. We plan to explore the flexibility-cost trade-off between the mapping performance benefits and the reconfigurability hardware overhead of a SRAM-based Digital CGRA. The key idea is to model this accelerator and explore it in the ZigZag framework, extended in WP6, for searching the optimal trade-off between hardware efficiency and flexibility considering several workloads as benchmark. Moreover, to speed up development, we also aim to build an automated design flow for this accelerator. By defining a parametrized accelerator configuration at the macro level, our flow is expected to generate the corresponding hardware layout with the assistance of EDA tools.

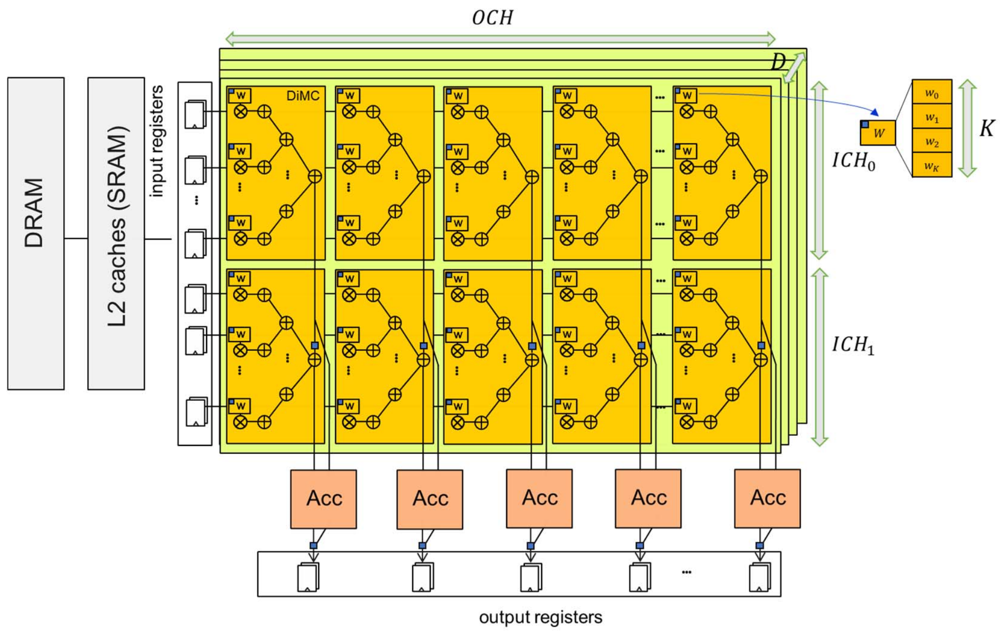


FIGURE 9. ILLUSTRATION OF A COARSE GRAIN RECONFIGURABLE ARRAY OF SRAM-BASED CIM ACCELERATOR. THERE ARE MANY DIFFERENT DATAFLOWS THAT CAN BE SUPPORTED BY RECONFIGURING THE CONNECTIONS BETWEEN THE MACROS. [THIS FIGURE WILL BE REVISED]

We plan to find the balance between circuit-level efficiency and reconfigurability to support the ever-changing set of different workloads. This entails a platform with the following characteristics:

- Flexible accelerator dataflow: A single dataflow cannot efficiently support the diverse types of layers; therefore, reconfigurable architectures such as DIANA [24] and TinyVers [25] that support multiple dataflows should be the ideal choice. However, careful trade-off analysis should be undertaken for reconfigurability vs. hardware overhead. This requires a design space exploration for the co-design of hardware and software.

- Flexible arithmetic precision: From the several performance plots, a strong correlation between precision and energy or power consumption can be observed. Therefore, finding the most optimal precision and training the models with a quantization-aware methodology to achieve good accuracy metrics remains a primary target. Support for variable precision computation can be beneficial to provide flexibility to support different workloads. Moreover, depending on the models, asymmetric quantization of weights and activation might be more efficient; however, it can increase the complexity of the hardware.

- Fine grained power management: Several of the existing NN algorithms have so much sparsity in their computations with no relevant information during the classification. These irrelevant computations (i.e., null operand or neutral MAC operations) can be optimized by turning-off the CIM datapath with a fine-grained management to achieve ultra-low power and ultra-low energy operations.

4.3.3. RRAM-based Analog CIM

RRAM-based CIM accelerator block will be developed in CONVOLVE to realize the key objectives of the project. The targeted RRAM-based CIM block has two main architectural units: (1) Memory array commonly known as crossbar array unit and periphery unit. The crossbar array stores the data and can perform any logic or arithmetic operation. Similarly, the periphery unit converts input/output data formats between analog and digital. Moreover, the periphery unit can also be used to perform basic logical and arithmetic operations.

4. *Crossbar array:* Different applications such as neuromorphic computing use primitive computational units such as multiply and accumulate (MAC) extensively in order to perform matrix-matrix multiplication (MMM). Such primitive units can be easily mapped into a RRAM-based crossbar array and perform their operation e.g., Vector Matrix Multiplication (VMM) in the CIM crossbar. The VMM is performed by applying a voltage vector V = V_j (where j ∈ {1, n}) to a RRAM-crossbar matrix of conductance values G = G_{ij} (where i ∈ {1, m}, j ∈ {1, n}) as shown in Figure 10. At any instance, each column performs a MAC operation, with the output current vector I, in which each element is $I_i = \Sigma V_j x G_{ij}$. Note that all m MAC operations are performed with O(1) time complexity.

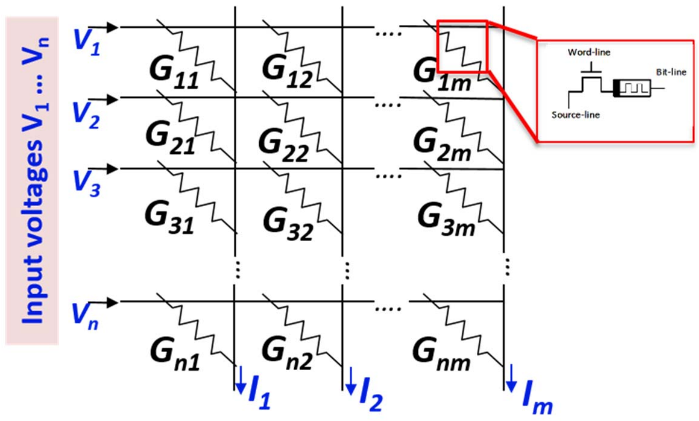


FIGURE 10 CROSSBAR STRUCTURE FOR RRAM-BASED CIM ACCELERATOR

5. *Periphery:* A RRAM-based CIM accelerator needs peripheral circuits to accommodate analog-based computing. For example, the following is needed to perform VMM operation in CIM: 1) Row-decoder becomes complex as it involves enabling several rows in parallel. Also, 1-bit row or word-line drivers need to be replaced by digital-to-analog converters (DACs) that convert multi-bit VMM operands into an array of analog voltages. 2) Column

periphery circuits performing read operations need to be replaced by analog-to-digital converters (ADCs). 3) Control block needs to deal with complex instructions such as handling intricacies of multi-operand VMM operations.

Both crossbar and peripheral units of the targeted RRAM-based CIM accelerator need to support different optimization knobs which can be used to tune some parameters in order to perform trade-off at runtime. Moreover, the RRAM-based CIM accelerator block will support proper interface that is supported by the DSE framework that will be developed in the CONVOLVE project.

5. Building Blocks in Neural Networks

The accelerators mentioned in the previous section have to support various neural network building blocks. Independent from the development of the use-cases in WP1, we survey and benchmark various topologies of neural networks that are commonly deployed and implemented. We break down the network topologies into building blocks such as layer types to give more insight details of the underlying operations needed to map such layer into hardware. Table 1 lists various options commonly used in Deep Neural Networks. Here, we categorize the building blocks into three categories: (1) standard operations – these building blocks are commonly used in various network topologies, and development of such layers are driven by the algorithms to improve e.g. accuracy of the networks; (2) efficient operations – these building blocks are structured to have some hardware awareness during inference, i.e. typically the efficiency is measured by minimizing the operation counts or the parameter size; and lastly (3) optimization techniques – here we deal with further optimizations such as synapse pruning techniques, quantization, compression, and transformation. Such optimizations require hardware support to enable efficient inference.

In the next sections, we highlight the required building block support for the different use-cases.

TABLE 1. COMMON BUILDING BLOCKS IN NEURAL NETWORKS.

Standard Operations	Efficient Operations	Optimization Techniques
1x1, 3x3, 5x5 or larger Convolutions	Bottleneck Structure	Weight Pruning (fully connected layer)
Parallel Convolutional layers	Depth-wise Separable Convolutions	Weight Pruning (convolutional layer)

ReLU, Leaky ReLU, Clip ReLU, tanh, Sigmoid Activations	Separable Convolutions	Fixed-point Quantization (activation / weights)
Average- / Max-Pooling	Densely Connected Convolutional Layers	Binary/Ternary Quantization
Concatenation	Group Convolutions	Power-of-Two Quantization
Max-out	Channel Shuffle	Nonuniform-Interval Few-Bit Quantization
Fully Connected Layers	Dilated Convolutions	Mixed Bit-Width Quantization
Up-sampling	Batch Normalization	Huffman Coding
Skip Connections	Feature Map Normalization	Run Length Coding
Residual Connections		Frequency Domain Methods
Two-staged Approach with ROI-Pooling		Matrix Factorization
Plain RNN		Approximate Units
Recurrent Layers except Plain RNN		Transform-Domains
Non-Maximum Suppression		Structured Sparse Weights
Padding		Canonical Signed Digit
Transpose Convolution		

5.1. Use-Case Noise Suppression

Audio data can be regarded as mostly time series data so in general, recurrent architectures/layers are highly relevant in this field. However, there are variations on the usage of those in combination with other layers. A full list of desirable of layers/activations/abstractions is given by:

TABLE 2. REQUIREMENT BUILDING BLOCKS FOR THE USE-CASE NOISE SUPPRESSION.

Layer	Activation	Other
1d convolution	ReLu	FFT
2d convolution	tanh	Mel – transform
GRU	sigmoidal	Upsampling
LSTM	linear	Downsampling
Dense		Reshape
Transpose 1d convolution		Expand dimensions
Transpose 2d convolution		

A more general statement could be that all PyTorch supported layers and abstractions (*Other* in the table) should be supported.

5.2. Use-Case Acoustic Scene Analysis

The use case of acoustic scene analysis also builds on time series data. Hence, recurrent layers as well as common feature extraction methods are key components of the proposed solutions (Table 3). Ideally, up to 1000 units should be supported for GRU (Gated Recurrent Unit), LSTM (Long Short Term Memory) as well as Dense layers.

TABLE 3 REQUIREMENT BUILDING BLOCKS FOR THE USE-CASE OF ACOUSTIC SCENE ANALYSIS.

	Type	Comments
Layer	1d Convolution	Raw audio-based processing
	GRU	Encoding time event in the neurons
	LSTM	Encoding time event in the neurons
	Dense	Classifier read-out
	SNN	Event-based neurons, e.g. LIF
Activation	ReLU	Commonly used activations
	Tanh	Gated activation to enforce stability

	Sigmoid	Gated activation to enforce stability
Feature extraction	FFT	Pre-processing time-series into frequency domain
	Mel-transform	Pre-processing time-series into time-frequency resolution
Other	Reshape	Restructuring neurons
	Move dimensions	Align shape of conv and recurrent/dense layers

In the scope of the CONVOLVE project, we would like to investigate SNNs for the use-case of acoustic scene analysis. Here, SNN layers of comparable size and composed of e.g., leaky integrate-and-fire (LIF) neurons could serve as a starting point. Similar to the existing solutions, the implementation of recurrent connections should also be possible for SNNs.

Apart from the handling of audio data based on conventional feature extraction methods, the processing of raw audio signals with neural networks is an interesting direction of current research. To draw on recent successes, 1d convolutions could be targeted as well.

5.3. Use-Case Image-based Perception

Compared to acoustic use-cases detailed in previous sections, image-based perceptions such as image classification or object detection exploit standard network topologies. Thanks to the major research contributions in this field, a lot of different network topologies are already developed, targeting and optimizing certain aspects of the tasks such as accuracy, latency, or even compute demand.

Some examples of neural networks for image-based perception are:

- MobileNet and its derivatives are designed to minimize the number of compute operation.
- SqueezeNet and its derivatives are designed to minimize the memory size and parameter counts.
- ResNet and its derivatives are designed to improve accuracy by adding residual connections.

- Yolo and its derivatives are designed to speed-up object detection by exploiting single-stage approach compared to more complex and more accurate two-stage approaches such as FPNs (Feature Pyramid Network) and R-CNN (Region-based CNN) and its derivatives.

Therefore, the building blocks that are needed to support inferring such networks are already covered at the beginning of this chapter.

5.4. Specific Requirement on Spiking Neural Network Accelerator Components

Components to relieve processor(s) of load and consequently save energy are envisaged in several architectural areas; many of the principles are derived from previous experience with the ARM-based SpiNNaker devices A principal consideration is to address the volume of data movement, primarily by reducing the size of the synaptic weight operands. These operands quickly become numerous, requiring 'backing store' (SDRAM) and their transfer for processing represents a significant energy cost. However, they can be quite low resolution so are amenable to a degree of compression. A single value might be stored in (say) 8 (or even fewer) bits.

It is intended to assess two possible ways to handle such variables. The first is to implement hardware functional units which could process such data types directly. Processing could then be performed on SIMD vectors within a RISC-V by an extension of the ISA; similar extensions have been introduced to earlier ISA for comparable purposes such as 'multimedia'. The processing could be done from standard registers by enhancing an existing datapath or with a wider coprocessor in a manner similar to x86 MMX/SSE or ARM NEON.

A complementary approach is to unpack and translate data representation during the data fetch -- and reverse this process if data is being rewritten. This can be done during and by the DMA transfer; the packed data can be expanded as it is cached in a local memory. Programmable flexibility of the formats would be included.

An issue with short-format representation is the loss of resolution. This is typically not a serious issue but becomes important when a network is learning, since it is desirable to apply many, repeated, very small changes as the network converges to a stable state. A compromise here is to apply larger changes stochastically (probabilistically) which can have a closely equivalent effect without the need for large variables. This requires a ready supply of (pseudo) random values; the generation and application of these is much cheaper with dedicated hardware.

Lastly, for the purposes of robustness, continuous learning/adaptation and, probably, to assist Dynamic Neural Network (DyNN) configuration it is desirable to keep some short-/medium-term records of past activity so that weight adjustments can be made when a more systemic outcome is known. The exact needs here are still to be determined but there must exist an equivalent in biology, it is necessarily fairly simple — although perhaps massively parallel — and it is probably a 'leaky integrator' similar to the spiking neurons but over a longer timescale. One type of support hardware for this type of function could also support the neuron firing state evolution. A semi-autonomous (configurable) unit working close to a memory buffer could provide this without costly software intervention.

6. Initial Standardize Interface

The accelerators designed in WP2 will implement standardized interfaces to ensure a compositional integration into the system-on-chip (SoC) template. This standardization enables a rapid change of the heterogeneous configuration of the SoC. The SoC template and the interfaces are specified in the Deliverable D6.1 of WP6. The next section will give a short summary on the interfaces.

6.1. SoC Template and Accelerator Integration Levels

The SoC template consists of a host infrastructure domain that includes a RISC-V host, main memory, and peripherals. It is connected to a group of L2-accelerators through high-speed on-chip interconnect such as network-on-chip (NoC) or AMBA AXI. These L2-accelerators may consist of similar or different types of accelerators. A sample L2-accelerator can be found in the compute cluster depicted on the right side of Figure 11. Within this accelerator, general-purpose RISC-V cores can be enhanced with L0-accelerators or share TCDM with an array of L1-accelerators. Each accelerator has its own gate-able clock and reset domain at both the L1 and L2 levels.

We outline three distinct levels of accelerator integration within the SoC template. Each of them has different interface requirements:

- L0: RISC-V co-processor
- L1: Tightly coupled accelerator
- L2: Loosely coupled AXI- accelerator

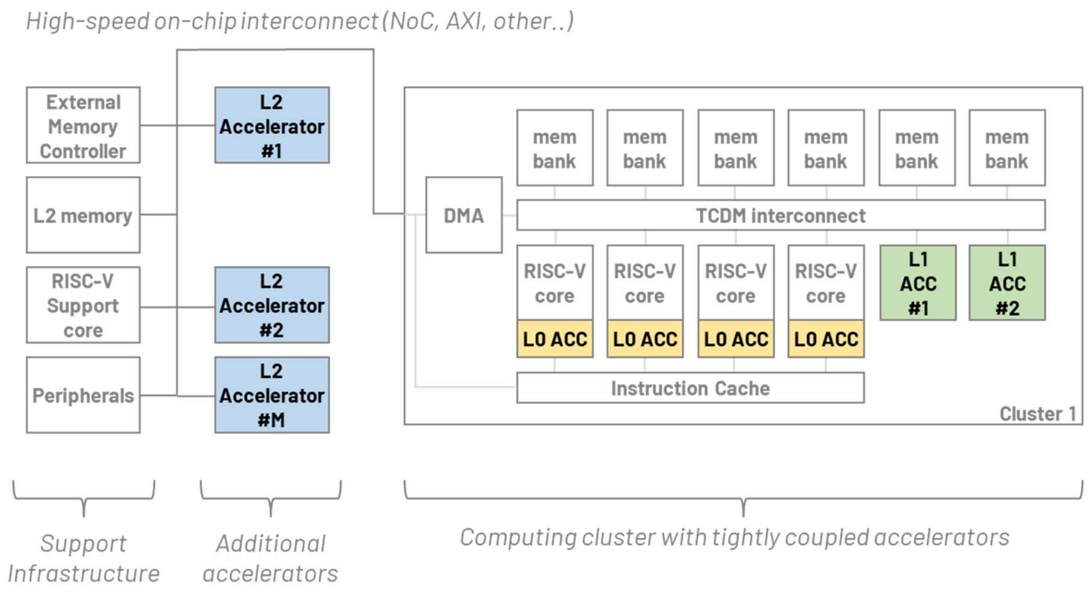


FIGURE 11. OVERVIEW OF THE HIGH-LEVEL SOC TEMPLATE WITH THREE ACCELERATOR INTEGRATION LEVELS.

6.2. Standard Interfaces

The accelerators described in section 4 can make use of the different interfaces provided by our platform. All protocols for these interfaces, as mentioned below, are further described in the Deliverable D6.1.

6.2.1. L0-Accelerator: Core-V-X Interface

L0-accelerators are specialized Co-Processors that work closely with a RISC-V core. They enable customized instruction set architecture (ISA) extensions to accelerate specific workloads such as the neural network building blocks described in Section 5. The core offloads any instructions that itself cannot process to the L0-accelerator. L0-Accelerators can be attached to the RISC-V PEs via the Core-V-X interface[1]

6.2.2. L1-Accelerator: TCDM/HCI Data Interface & PERIPH/REGBUS Interface

L1 accelerators follow the interfaces supported by the Hardware Processing Engines (HWPEs)[2]. Each L1 accelerator has two interfaces, one for streaming in data and one to

[1] https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/en/latest/
[2] https://hwpe-doc.readthedocs.io/

configure new jobs. The data interface can implement a *master* of the TCDM or HCI protocol and the control interface can implement either a *slave* of the PERIPH or REGBUS protocol.

6.2.3. L2-Accelerator: AXI4 Interface

The L2 accelerators employ AXI4 for both data management and control, following ARM's AXI4 specification. An accelerator will have two AXI ports, a single slave port allowing incoming requests for control, and a single master port allowing outgoing requests for data movement. Please note that these most likely will have different ID widths for proper interconnect design.

7. Summary

This document details the challenges in designing hardware accelerators for Deep Learning, which will be used as baseline for our hardware development in WP2. Prior to defining our planned works in designing various types of accelerators, we cluster different types of existing hardware accelerators and architectures to give us some insights what we want to target in the CONVOLVE project. Two emerging memory options SRAM and RRAM to reduce data movements by allowing compute unit implemented in the memory block are detailed as our CIM preference on edge devices. Coupled with break-down of neural networks into building blocks, we will use this information to refine our hardware designs and architectures so that our targeted systems are efficient to handle such workload. Finally, by having standardize interface, generation of SoC that composes of different hardware accelerators will be speed-up. This also allows us to broaden our design-space exploration of infrastructure and accelerator to further optimize our targeted systems in term of power consumption, die size, latency, security, etc.

The table below summarizes the different hardware building blocks that will be developed in WP2 with the involved partners:

TABLE 4 SUMMARY OF DIFFERENT HARDWARE BUILDING BLOCKS.

ULP block	Partner to contribute
RISC-V	BOS
CGRA	TUE
SRAM-based Digital CiM	TUD, TUE, BOS
Coarser CGRA using SRAM-based Digital CIM	KUL, BOS
RRAM-based (CiM)	TUD
Initial Standardize Interface	ETH

Bibliography

[1] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi and J. Kepner, "AI Accelerator Survey and Trends," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)(2021)*.

[2] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2014.

[3] Shihua Huang, Luc Waeijen, and Henk Corporaal, "How Flexible is Your Computing System?," *ACM Trans. Embed. Comput. Syst.*, 2022.

[4] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," *Association for Computing Machinery*, 2014.

[5] Y. Chen et.al,, "DaDianNao: A Machine-Learning Supercomputer," in *IEEE/ACM International Symposium on Microarchitecture*, 2014.

[6] D. Liu et.al.,, "PuDianNao: A Polyvalent Machine Learning AcceleratorD. Liu et al.," *Association for Computing Machinery*, 2015.

[7] Y.-H. Chen, J. Emer and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow," *IEEE Journal of Solid-State Circuits*, 2017.

[8] Y.-H. Chen, J. Emer and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[9] M. Cournariaux et. al.,, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," in *NeurIPS*, 2017.

[10] S. Vogel, A. Guntoro and G. Ascheid, "Self-Supervised Quantization of Pre-Trained Neural Networks for Multiplierless Acceleration," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[11] J. -s. Seo et al., ""Digital Versus Analog Artificial Intelligence Accelerators: Advances, trends, and emerging designs,"," *IEEE Solid-State Circuits Magazine, vol. 14, no. 3, pp. 65-79,* , Summer 2022.

[12] J.-H. Lee et.al,, "Exploring cycle-to-cycle and device-todevice variation tolerance in MLC storage-based neural," in *TED*, 2019.

[13] Y. Jeong et.al,, "Parasitic effect analysis in memristorarray-based neuromorphic systems," in *Nanotech*, 2018.

[14] P. Chen et.al, , "Technological Benchmark of Analog," in *IDT*, 2019.

[15] M. Fieback et.al,, "Defects, fault modeling, and test development framework for rrams," in *JETC*, 2022.

[16] Shihua Huang, Luc Waeijen, Henk Corporaal, "How Flexible is Your Computing System?," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 4, pp. 1-41, 2022.

[17] F. Zaruba, F. Schuiki, T. Hoefler and L. Benini, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," in *IEEE Transactions on Computers*, 2021.

[18] Corporaal, H., & de Bruin, E., "What is a CGRA?," *ACM/SIGDA e-newsletter*, 2023.

[19] C. Tan and et.al, "AURORA: Automated Refinement of Coarse-Grained Reconfigurable Accelerators," in *DATE*, 2021.

[20] J. Weng, S. Liu, D. Kupsh and T. Nowatzki, "Unifying Spatial Accelerator Compilation With Idiomatic and Modular Transformations," in *IEEE Micro*, 2022.

[21] Kanishkan Vadivel, Roel Jordans, Sander Stujik, Henk Corporaal, Pekka Jääskeläinen, and Heikki Kultala, "Towards Efficient Code Generation for Exposed Datapath Architectures," in *SCOPES*, 2019.

[22] e. H. Fujiwara, "A 5-nm 254-tops/w 221-tops/mm2 fully-digital computing-in-memory macro supporting wide-range dynamic-voltage frequency scaling and simultaneous mac and write operations," in *ISSCC*, 2022.

[23] e. C.-F. Lee, "A 12nm 121-tops/w 41.6-tops/mm2 all digital full precision sram-based compute-in-memory with configurable bit-width for ai edge applications," in *VLSI Technology and Circuits*, 2022.

[24] K. Ueyoshi et al., "DIANA: An End-to-End Energy-Efficient Digital and ANAlog Hybrid Neural Network SoC," *International Solid- State Circuits Conference (ISSCC)*, 2022.

[25] V. Jain, S. Giraldo, J. D. Roose, B. Boons, L. Mei and M. Verhelst, "TinyVers: A 0.8-17 TOPS/W, 1.7 µW-20 mW, Tiny Versatile System-on-chip with State-Retentive eMRAM for Machine Learning Inference at the Extreme Edge," *IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2022.

[26] A. O. El-Rayis., "Reconfigurable Architectures for the Next Generation of Mobile Device Telecommunications Systems. Ph. D. Dissertation," 2014.