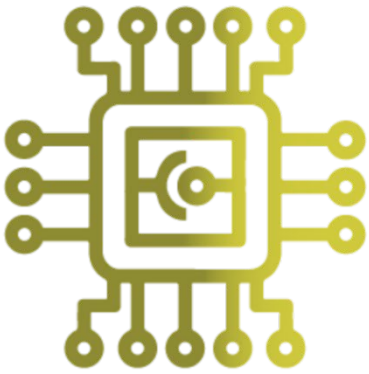

CONVOLVE

Seamless design of smart edge processors

GRANT AGREEMENT NUMBER: 101070374

Deliverable D2.3

Intermediate report on the accelerator simulator

Title of the deliverable	Intermediate report on the accelerator simulator
WP contributing to the deliverable	WP 2
Task contributing to the deliverable	Task 2.3
Dissemination level	RE - Restricted to a group specified by the consortium
Due submission date	30/04/2024
Actual submission date	25/06/2024
Author(s)	Bas Ahn [TUE] Rick Luiken [TUE] Manil Dev Gomony [TUE] Ahmet Turan Erozan [BOS] Andre Guntoro [BOS] Anteneh Gebregiorgis [TUD] Yash Biyani [TUD] Georgios Alexandris [ICCS] Sotirios Xydis [ICCS]
Internal reviewers	Friedemann Zenke [FMI] EIGUREN ARZA Gaizka [THA]

Document Version	Date	Change
V0.1	25.02.2024	Initial document
V0.2	25.04.2024	Integrated contributions from partners
V1.0	22.05.2024	First version for internal review
V2.0	24.06.2024	Final version for submission after addressing the comments from reviewers

Summary

Contents

Deliverable Summary

In WP2, several hardware accelerators supporting machine learning using Deep Neural Networks (DNN) are developed and implemented. This deliverable provides an intermediate report on the activities carries out under Task *T2.3: Simulator design and model development for targeted accelerators*. The goal of this task is to develop tunable performance models for each hardware accelerator block at different levels of abstraction such that they can be used to perform rapid Design Space Exploration (DSE) and functional verification. In this report, first we provide a general motivation for the need of tunable accelerator simulator models. Then we present the tunable simulation models of different hardware accelerators at different levels of abstraction. In addition to the tunable accelerator models, this reports presents the progress of the accelerators developed in the project. Finally we conclude the document with a summary.

1. Introduction

Achieving Ultra Low Power (ULP) consumption in the accelerator blocks requires extensive DSE of the different architectural parameters such as memory hierarchy, memory sizes, number and type of processing units, interconnection scheme, scheduling etc. Since the design-space is large, we need fast and accurate way to explore the design space and identify the optimal architectural parameters. In CONVOLVE, we use two different frameworks for fast DSE of hardware accelerators. One is the *ZigZag* (developed in WP6) , shown in Figure 1, is an analytical modeling framework designed to optimize dataflow and explore single-core architectures for DNN (Deep Neural Network) and SNN (Spiking Neural Network) inference. ZigZag aims to identify the most efficient spatial and temporal mappings to minimize costs such as energy consumption, latency, and EDP (Energy-Delay Product). The analysis conducted by ZigZag is structured in a layer-wise manner, allowing for the breakdown of workloads into distinct layers for individual performance assessment. Refer to Deliverable D6.3 for a more detailed explanation of ZigZag framework. Note that the *SigSag* is an extension of ZigZag tool to support SNNs.

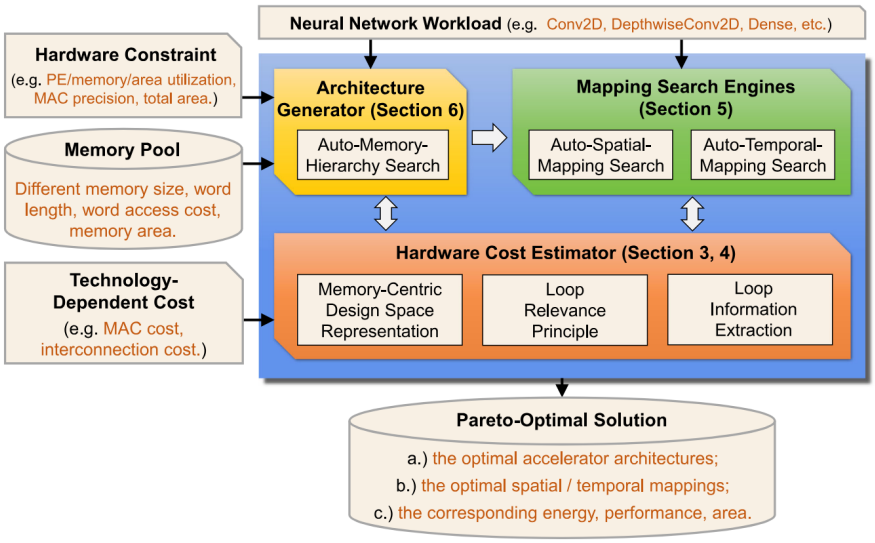

Figure 1: Overview of the ZigZag DSE framework. ZigZag consists of an analytical model called Hardware Cost Estimator that uses energy-performance-models of hardware building blocks for estimating the cost, Mapping Search Engines that tries various mapping strategies on an architecture, and an Architecture Generator that explores different memory hierarchy and generates possible accelerator architectures automatically [1].

The second framework is GVSoC, which is a configurable, fast and accurate system-level simulator for RISC-V based accelerators as shown in Figure 2. GVSoC takes the IP models (of core, memories, DMAs etc) described in a high-level language (C/C++), and the architecture parameters (e.g. interconnect bandwidth/latency) configuration via JSON file. It uses a set of python generators to instantiate all components of the specified platform. The modular structure of GVSoC allows compilation of high level models of the IP blocks and build a specific platform without recompiling the simulator enabling fast DSE. More detailed explanation of GVSoC is provided in Deliverable 6.3.

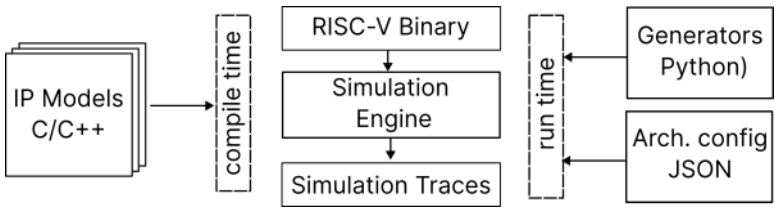

Figure 2: Overview of the GVSoC simulation framework [2]. Main components of GVSoC include the Architecture config in JSON files, the Python generators instantiate the components and C++ models of the IP blocks.

Both ZigZag and GVSoC requires the high-level tunable performance models of hardware IP blocks as input to perform fast DSE. In addition, low-level models of the accelerators are required for functional verification. Hence, the goal of the task T2.3 is to develop models of targeted hardware accelerators at different levels of abstraction so that their architectural DSE can be explored using ZigZag or GVSoC and functionally verified. Table 1 below describes

the different ULP accelerators indicating the different models being developed and their current progress. The rest of the deliverable is structured as follows: Section 2 of this deliverable first describes the progress of the digital accelerators and their tunable performance models of digital accelerators, and then the compute-in-memory (CIM) based accelerators in Section 3.

TABLE 1: ULP ACCELERATORS AND THE PROGRESS OF DIFFERENT MODELS BEING DEVELOPED IN CONVOLVE.

ULP accelerator	ZigZag model available	GVSoC model available	RTL/Circuit model available	Section #	Lead partner
RISC-V Instruction-set extensions	-	Yes	Yes	2.1	BOS
Approximate CGRA	Yes	-	Yes	2.2	ICCS
2-D-Array GeMM Accelerator	Yes	-	Yes	2.3	KUL
Near memory ANN/SNN acceleration	No	-	Yes	2.4	MAN
Dynamic SNN accelerator	Yes	-	Yes	2.5	TUE
Adder-tree Free SRAM-based CIM architecture	No	-	Yes	3.1	TUD
Fibbinary multi-bit multiplier Approximate DCIM architecture	No	-	Yes	3.2	TUE
RRAM-based analog CIM accelerator with self-healing	No	-	Yes	3.3	TUD

2. Digital accelerators and their performance models

Within WP2, five different digital accelerators are developed. 1) RISC-V Instruction-set extensions, 2) Approximate CGRA, 3) Digital 2D-array GeMM accelerator, 4) Near-memory ANN accelerator, 5) Dynamic SNN accelerator. This section presents the tunable performance models of the five different digital accelerators.

2.1. RISC-V Instruction-set extensions

The baseline for performance model and platform used in this project involves simulating the *Snitch* core using GVSoC, depicted in Figure 2, a high-level event-based simulator specifically designed for RISC-V cores [40]. Snitch, a highly configurable, single-stage, single-issue, in-order RISC-V core, is tuned for simplicity and minimal area footprint. GVSoC provides a versatile and scalable environment for system-level simulations, allowing for the accurate evaluation of the Snitch core's performance. By utilizing GVSoC, the Snitch core can be effectively simulated, enabling comprehensive analysis and optimization across various

applications. This combination of Snitch and GVSoC forms a robust baseline, offering the flexibility and accuracy required for performance modeling and evaluation.
The ISA extensions for Snitch can play a crucial role in efficiently supporting various functionalities. For digital signal processing (DSP), Snitch's ISA extensions provide specialized instructions and optimizations that enhance the core's ability to perform complex mathematical operations, such as filtering, wavelet transforms, and FFT. These extensions enable Snitch to efficiently handle DSP tasks. Additionally, Snitch's ISA extensions can be utilized to support spectrum-based operations, such as Mel-Transform, which allows for the extraction of important frequency components and accurate representation of spectral information. The flexibility of Snitch's ISA extensions also enables the implementation of pre-processing filter banks, facilitating efficient decomposition and analysis of acoustic signals. Furthermore, Snitch's ISA extensions can be leveraged for GCC-PHAT optimization [3], enabling precise acoustic localization through correlation operations. By leveraging these ISA extensions, Snitch becomes a versatile platform capable of efficiently supporting a wide range of functionalities in audio processing and acoustic analysis.

In contrast to typical AI workloads that often utilize reduced bit widths to reduce power consumption, signal processing tasks such as filtering, wavelet transforms, and FFT require a different approach. These applications necessitate higher output precision to preserve signal integrity and maintain a high signal-to-noise ratio (SNR). Therefore, in DSP, data types like block floating-point are often preferred. Block floating-point allows for increased precision in representing and manipulating signals, ensuring accurate results, and minimizing the loss of valuable information during processing. By employing higher output precision data types, DSP algorithms can effectively handle complex signal processing tasks while maintaining the desired level of accuracy and fidelity. With the help of GVSoC with Snitch cores as target RISC-V, it gives us opportunities to assess various configurations to be tuned for our demands.

Spectrum-based supports, like Mel-Transform, are essential in neural network approaches for acoustic perception. Initially, the Fast Fourier Transform (FFT) is applied with a Hanning window to reduce spectral leakage. However, due to the unique characteristics of acoustic signals, the important frequency components are extracted using Mel-Transformation. This mapping to the Mel scale emphasizes perceptually relevant frequency bands, aligning the spectrum representation with human auditory perception. Incorporating Mel-Transform enables more effective analysis and processing of acoustic signals in neural network-based acoustic perception systems. We plan to evaluate various options to incorporate Mel-Transform in Snitch cores and with the help of GVSoC, derive some KPIs for even broader acoustic use cases.

Additionally, to ensure better reconstructability of the acoustic signal, orthogonal time-based filter banks are utilized instead of spectrum-based approaches. These filter banks decompose the signal into sub-bands, preventing overburdening of subsequent processing stages. By incorporating pre-processing filter banks, the acoustic signal is efficiently represented and analyzed, preserving essential characteristics while reducing redundancy. This enhances the overall performance and accuracy of acoustic signal processing algorithms in various applications.

Further, acoustic localization can be achieved through various methods, including CNN-based approaches and signal correlation techniques like GCC-PHAT. GCC-PHAT involves correlation operations to estimate time delays between audio signals from multiple microphones, enabling accurate localization. Additionally, the output of GCC-PHAT can be used as input to CNN models, enhancing their performance in acoustic localization tasks. This combination of signal correlation and CNN-based approaches optimizes GCC-PHAT for precise and robust acoustic localization in diverse scenarios.

Last performance assessment with fast model on Snitch cores is to investigate the support of Recurrent Neural Networks (RNN). To reduce latency in full spectrum-based CNN approaches, RNNs are employed. By inputting small FFT windows into the RNN and utilizing overlapping windows, the prediction latency of RNNs can be significantly reduced. This combination enables real-time and efficient processing of acoustic signals, facilitating faster analysis in various applications.

Suitable RISC-V ISA extensions for aforementioned algorithms will be investigated to improve the efficiency of RISC-V cores processing targeted workloads and implemented within GVSoC to properly simulate and validate the improvement gained by these ISA extensions. To implement and integrate the ISA extensions, the hardware-related properties of the ISA extensions will be extracted and modeled in corresponding parts of GVSoC simulator.

2.2. Approximate CGRA

As introduced in Deliverable D2.2, we will use Coarse Grained Reconfigurable Array (CGRA) [1] for developing flexible approximate CGRA accelerators. The need for approximation in hardware components has been a major research field for energy efficient design of hardware components. In the deliverable D2.2 we presented a methodology that evaluates state-of-the-art [31][32][33] approximate techniques in multiplication by using behavioural simulation with Verilator and high-level inference of DenseNet NN. By using the Mean Squared Error output of the layers and in combination with the Power and Area needs of each multiplier, we generated the Pareto Optimal solutions for our task. This analysis concluded that the DRUM[32] and ROUP [31] multipliers are a good fit for convolution NNs. In the results there also included multipliers from the EvoApprox8b [33] library which were discarded due to reduced modularity. With those designs in mind, we continued with the integration of those components into the Blocks CGRA [1] environment. This integration introduced three different types of tiles, the **Static**, **Dynamic** and **MAC**.

Static Approximate Tiles

Static CGRA Tiles function like the prebuilt MUL Tiles, apart from the approximation unit added where the accurate "vanilla" adder used to be.

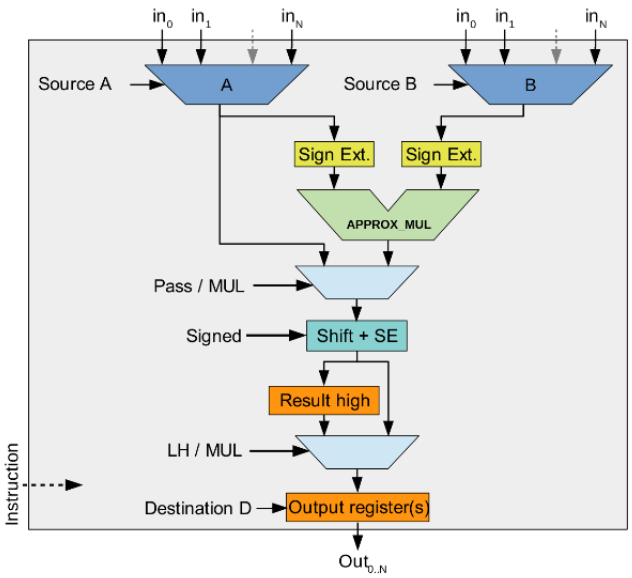

Figure 3: Static Approximate Tile.

Figure 3 presents the stock implementation of the MUL tile and of our Static Approximate Tile. Because of the similarity of the RTL description the ID of this Tile reuses the ISA of the MUL Tile, providing the same decoding of the instructions to be executed.

Dynamic AxC Tiles

Dynamic Tiles are Multiplication Tiles with different multiplication approximation parameters in the same block. This means that in each instruction fetch a different approximation can be selected which makes this tile bigger than a Static Tile (and that includes an impact in the Area/Power factors) but with the run-time approximation change functionality. At RTL level, these tiles function similarly to the Static Tile in the multiplication process but with extra hardware and control signals.

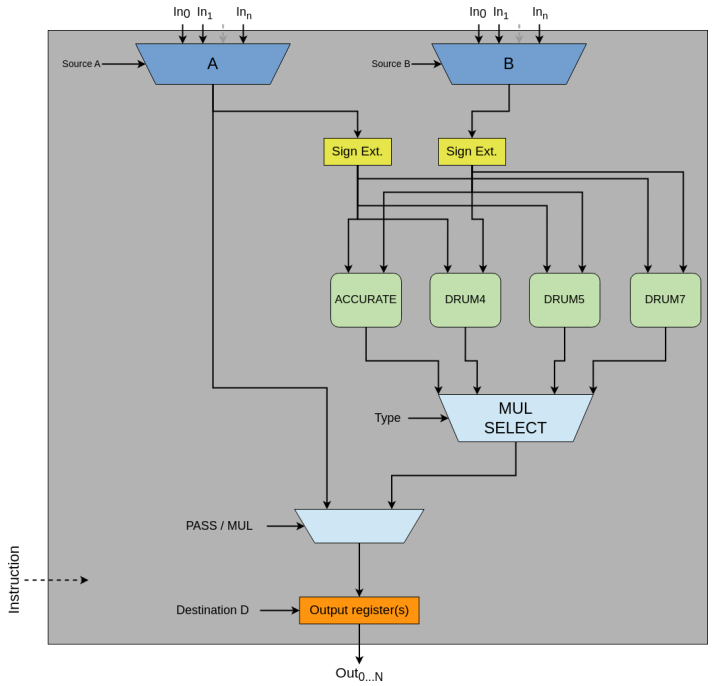

Figure 4: Dynamic Approximate Multiplication Tile (DRUMk).

Figure 4 presents an RTL representation of a Dynamic Tile. With the control signal **Type** we can control the type of the multiplication (Accurate/Approximate) as well as the k parameter of the DRUM multiplier. The ID tile of this implementation uses extra bits in the ISA without extending the decoded size (it is still 16-bit as used in the MUL Tile) to implement the extra control. So, the ISA was extended to include the approximator selection with the commands presented in the Table 2 below.

TABLE 2: DESCRIPTION OF ISA.

Name	Instruction	Function
mul_drum4	0000000_**01**_10_D_BB_AA	Select DRUM4 Multiplier
mul_drum5	0000000_**10**_10_D_BB_AA	Select DRUM5 Multiplier
mul_drum7	0000000_**11**_10_D_BB_AA	Select DRUM7 Multiplier
mul_acc	0000000_**00**_10_D_BB_AA	Select Accurate Multiplier
pass	0000000_00_01_D_??_AA	Pass the PORTA value to Out
nop	0000000_00_00_?_??_??	No Operation (Out = 0)

In Table 1, we see the different approximation options along with the Opcode. The highlighted bits (in blue) are the approximation control bits, the AA and BB bits are the input port selector bits, and the D bit is the output selector bit. Finally, in this implementation hardware of the different DRUM [32] multipliers are reused for Area efficiency. More specifically, the P Encoders and the Barrel Shifters used in this multiplier's implementation are common among the different k parameter changes.

MAC Unit

As an extension to the Dynamic Multiplication Unit is the MAC Unit which in addition to the multiplication selection it also accumulates the output back to the adder.

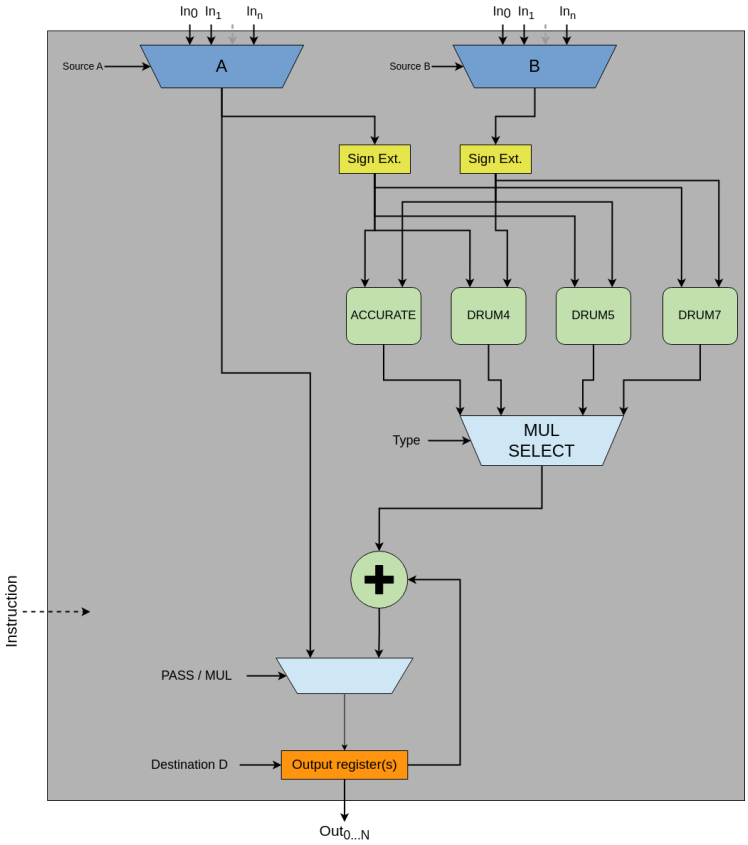

Figure 5: Approximate MAC Unit Tile.

Figure 5 shows a basic RTL of the Approximate MAC Unit Tile. At the ISA level the commands and decoded instructions are the same as the Dynamic Multiplication Tile except the naming of the actual performing commands (for example the mul_drum4 command is now mac_drum4).

Functional Units (Tiles) to CGRA Integration

All the tiles described above are integrated to the designed architectures by implementing the RTL of the approximate and ID units adding the top level wrapping I/O for each unit.

TABLE 3: CGRA TILE I/O.

I/O	Bitwidth	Description
iClk	1	System Clock
iReset	1	System Reset
iInputs	NUM_INPUTS*DATA_WIDTH	Serialized Input Data
iDecodedInstruction	16 (our case)	Decoded Opcode
State Control	1 (multiple signals)	Optional State Control of CGRA
oOutputs	NUM_OUTPUTS*DATA_WIDTH*2	Serialized Outputs
oStateOut	1	Optional State Control of CGRA

Power/Area Lookup Table

Due to the experimentation and development constraints of the CGRA environment we concluded that the complete R-Blocks ZigZag integration will be added in a later stage during the tapeout planning. However, the need to have an estimation of the Energy and Area needs of our approximate components is still imminent. For this purpose, in this section, a **Lookup Table** will be presented, including the Energy and Area estimations of all the Tiles described above, which were synthesized by the **Synopsys Design Compiler** and the library **GSCLIB45nm**. This Lookup Table will be used as input to ZigZag during DSE.

In the analysis all the Tiles are configured with **16-bit Data Input** and **32-bit Data Output**.

TABLE 4: POWER, AREA FOR DIFFERENT UNITS.

Tile	Frequency (MHz)	Power (mW)	Area
Accurate MUL	300	2.44	4934
	500	4.05	
	700	5.68	
DRUM4	300	0,80	1956
	500	1,32	
	700	1,85	
DRUM5	300	0.93	2135
	500	1.54	
	700	2.17	

DRUM6	300	1.09	2373
	500	1.81	
	700	2.54	
DRUM7	300	1.29	2523
	500	2.14	
	700	3	
Dynamic MUL	300	3.71	6321
	500	6.17	
	700	8.68	
MAC	300	3.89	7025
	500	7.5	
	700	10.56	

In Table 4, we can see the Power and Area estimations of our tiles. From that point of view we can clearly observe that the DRUM approximate components are more efficient both in power and in area sections, which is expected. The most important aspect of this table is the comparison between the Accurate Multiplier and the provided Dynamic/MAC tiles. In those tiles we can see that the Dynamic module draws less than 2x Power and Area compared to the stock MUL tile provided by the Blocks CGRA. That data motivates us to further investigate the integration extensions.

ZigZag Evaluation of Multiplication Units

For exploring the memory needs and the number of multiplications in our design, we must rely on two critical constraints, the actual computation logic and the memory components. In terms of computational logic the most part of our research focused on the multiplication handling. So for now the architecture will be described as a grid of MAC units, which in the actual CGRA implementation will be represented as MUL-ALU tile combinations. For the memory implementation, we are currently proposing an architecture with three Loca l Memory (LM) tiles which will handle **input**, **weights** and **output** feature maps accordingly for the given NN model, in order to have a well defined address space for each feature map, for better presenting and handling the results. To have a more close-to-real scenario for our estimation we used a compute intensive intermediate layer of the YOLOv6 NN which called ERBlock_4, a schematic of this block is shown in Figure 6.

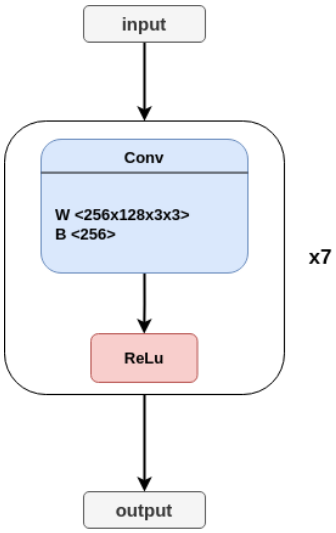

Figure 6: YOLOv6 intermediate layer ERBlock_4.

Due to ZigZag limitations in evaluating the "ReLu" layers, we could only model the Convolution layers of the mentioned block. Still, this estimation is enough for our exploration, because all our testing processes handled so far were also on Convolution based procedures. The test results were based on three different memory size (each one of the three local memories will have the same size) : 3KB, 4KB, 8KB and three different MAC unit grid sizes: 4x4, 5x5, 6x6, we run the exploration as mentioned and the results produced are shown in Figure 7.

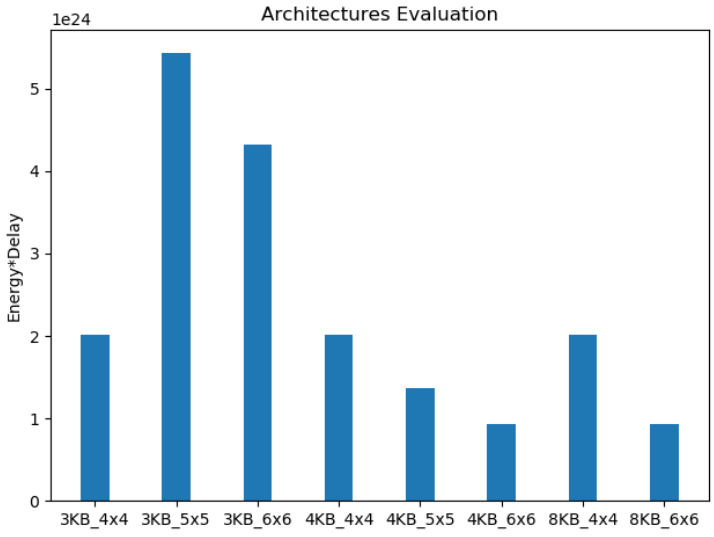

Figure 7: ZigZag Architecture Exploration.

From the results presented in Figure 7, the best architecture for implementing the specified ERBlock_4 of the YOLOv6 NN is using a grid of 6x6 MAC units and three local memory banks, with 4KB of capacity each.

Proposed Architecture

For the proposed CGRA architecture we are going to generate a more aligned version of the MAC units (MUL + ALU) with the given Local Memory blocks, which comes to generating a 4x4 grid of MAC units with the implementation of the 3x4KB Local Memory tiles. A schematic of the proposed architecture and communication principles can be found in Figure 8.

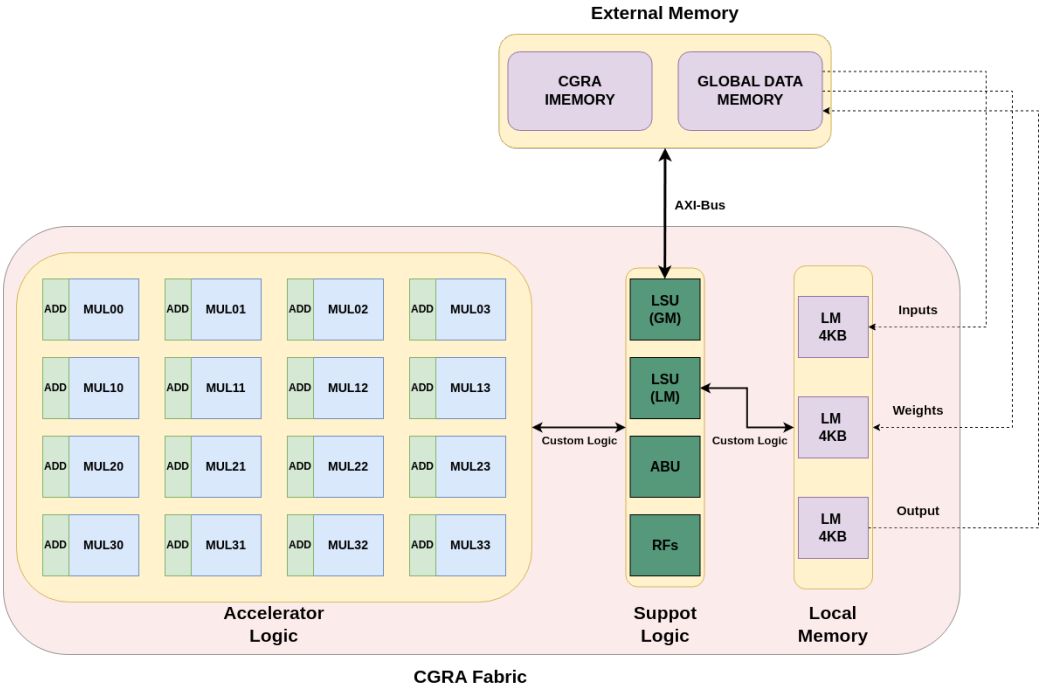

Figure 8: Proposed CGRA Architecture

For clarity purposes, the interconnection network is not drawn. Also, the exact number of the support logic will be better determined after further examination of the test case implementation. As shown in Figure 8, the CGRA will communicate only with the external memory of the system, providing the instruction (+ configuration) words to the ID units of the CGRA and a Data communication which will fetch the appropriate layer specific data words (in our build weights and inputs) to the LM (Local Memory) tiles. After the completion of the partial output (aligned with the given memories), the contents of the output Local Memory will be finally transferred to the External Memory. Other support units (Register Files and/or extra Load Store Units) must be explored in the low level for the exact number to be provided.

In this section, we presented a modeling of our selected approximate components [31][32], as they will be integrated into the R-Blocks environment, along with an estimation of the power and area needs of each tile. This exploration was followed by a MAC and Memory needs with exploration in ZigZag using the intermediate layer of YOLOv6 ERBlock 4. Those evaluations gave a first picture of how the approximate tiles will function in the R-Blocks CGRA environment as well as how many MAC (MUL + ALU) tiles and memory is needed for accelerating use case related NNs.

2.3. 2D-array GeMM accelerator

The GeMM accelerator is built to accelerate the general matrix multiplication process. It is highly parameterizable. The design-time configurable parameters are listed below.

TABLE 5: DESIGN-TIME CONFIGURABLE PARAMETERS OF GEMM ACCELERATOR.

Parameter	Meaning
dataWidthA	Input matrix A data width (integer)
dataWidthB	Input matrix B data width (integer)
dataWidthC	Output matrix data width (integer)
dataWidthAccum	Accumulator data width (integer)
meshRow	The row number of the GEMM array
meshCol	The column number of the GEMM array
tileSize	The tile size of each tile

GeMM accelerator's microarchitecture is shown in the Figure 9 below. It has an 2D-array of dot product unit, namely DotProd. Each DotProd block conducts the dot product of one row of matrix A and column of matrix B. It is important to notice that each element of A is reused across each column, while each element of B is reused across each row, and each multiplication result is accumulated to a partial result of C, thereby maximizing data reuse.

At run-time, the actual workload dimensions (M, K, N) can be programmed into the accelerator through RISC-V Control and status register (CSR) instructions. The CSR manager oversees the CSR commands, which allows the CPU to program the GeMM accelerator. It also has a double buffered CSR for pre-loading CSR configurations while an ongoing task is running.

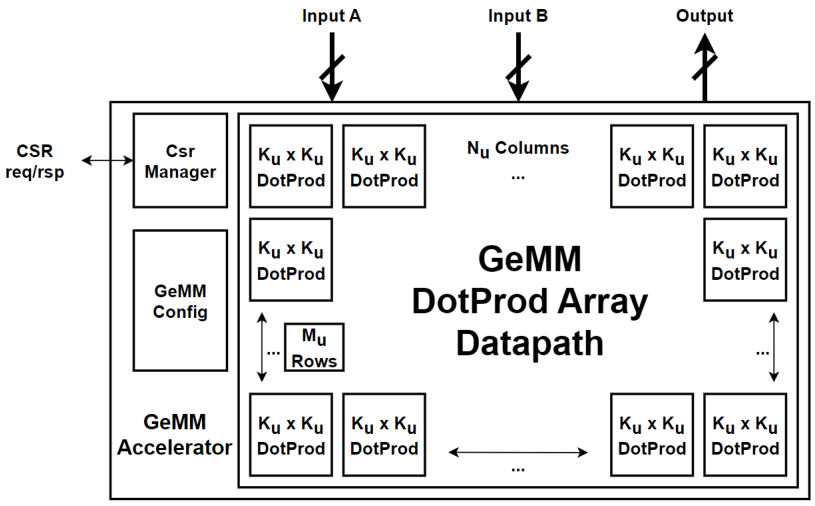

Figure 9. Digital 2-D Array GeMM accelerator.

GeMM ZigZag Modeling:

Figure 10 shows an 8x8x8 GeMM accelerator modelled in ZigZag as a three-dimensional array of multipliers. Each multiplier has a fixed input precision, energy cost and area.

```
def multiplier_array_dut():
    """Multiplier array variables"""
    multiplier_input_precision = [8, 8]
    multiplier_energy = 0.04
    multiplier_area = 1
    dimensions = {
        "D1": 8,  # unrolls M
        "D2": 8,  # unrolls N
        "D3": 8,  # unrolls K
    }

    multiplier = Multiplier(
        multiplier_input_precision, multiplier_energy, multiplier_area
    )
    multiplier_array = MultiplierArray(multiplier, dimensions)

    return multiplier_array
```

Figure 10. ZigZag Model of the GeMM array.

This array is extended with a hierarchy of memories, where it is unrolled along the specified dimensions of the multiplier array. Each memory has a limited capacity and a limited number of read, write, read-write ports with fixed bandwidth. Figure 11 shows the hierarchy of modelled memories visually. The interconnection of the different memories is handled through the port allocation. For more details, the model can be found here in GitHub: https://github.com/KULeuven-MICAS/zigzag/tree/gemm-example/zigzag/inputs/hardware

Figure 11. Memory Hierarchy Graph of the GeMM Array.

2.4. Near memory ANN accelerator

The objective is to provide a flexible means for the construction of structures to accomplish a diverse range of ANN functions, as exemplified by Convolve partners' problems. The implementations should provide significant energy savings over existing approaches whilst

still providing adequate performance and fidelity. To address the range of requirements, a 'kit of parts' approach has been adopted so that systems may be constructed (and adapted) according to need. To minimize energy, the components are customizable in two independent ways:

- When parts are instantiated the *maximum* size of operands can be set. This can reduce the size of datapaths, multipliers etc. from the 'default' sizes provided in software implementations (e.g. 32 bits).
- The operation sizes can be specified on an *individual* (operation-by-operation) basis at run time, so only the necessary precision is used, reducing memory accesses.

The instantiation of parts is also flexible, thus a simple system can be very small whilst parallel evaluation can be introduced according to need.

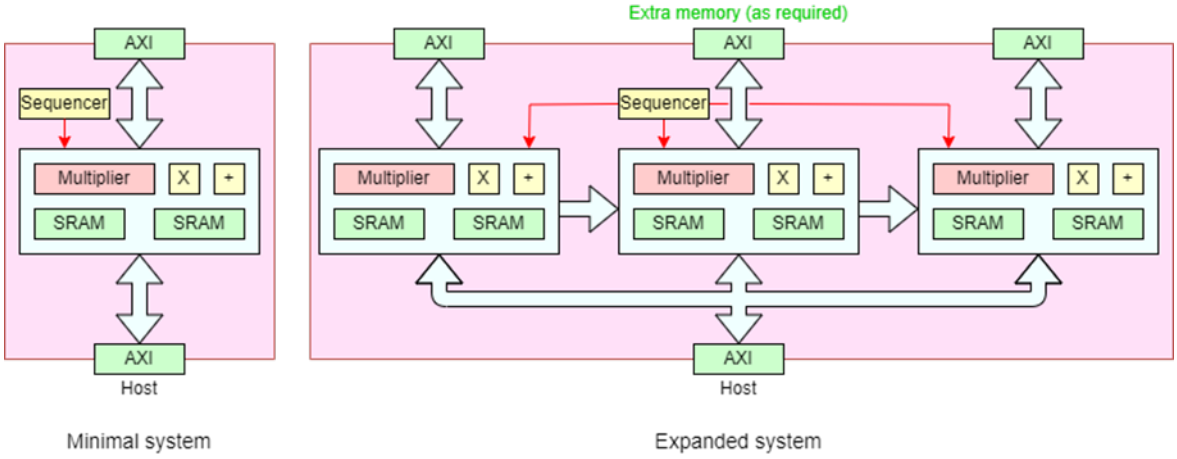

Figure 11. Near memory accelerator architecture.

ANN inference demands significant operand bandwidth. Particular focus thus concentrates on memory accesses since operand loads are a major energy sink. In particular the ANN weight matrices – by far the largest data structures – are compressible to the fewest bits/element the user is content with; this facility is controllable at the bit level so that (for example) if 7-bit operands are appropriate there would be a 72% saving in bits transferred compared with using 32-bit operands. Although the operand compression will vary according to the application (and often the *layer* within the application) and the end users' needs examination of existing data suggests a 3-4× saving seems entirely achievable. Operand representation is fixed-point for computational efficiency but the scaling (point position) is user-selectable in each operation step.

There are some 'standard' components needed for ANN applications; chief amongst these is the matrix multiplier, supplemented by a programmable, non-linear post-processor. Together, these can support a perceptron layer [2]: by using either several such units or applying such a processor repeatedly (or some appropriate combination) multi-layer networks can be constructed. These basic blocks can be supplemented with other processors if more sophisticated functions are wanted: for example the first of these addressed has been an accelerator combining fractions of two data streams (called the Hadamard Product Unit) used in a structure called a Gated Recurrent Unit (GRU) [3], currently part of Jabra's denoising algorithm. It is not difficult to add other supplementary parts as required.

Operation focusses around local memory (whenever possible) to reduce transfer distance and, hence, energy cost. A simple implementation may have a minimal set of processing blocks and a single local memory; larger, multiprocessor systems can have multiple processing blocks, each with its own local memory (to provide the necessary bandwidth) with some limited non-local access for operand transfer. Many applications have data 'footprints' which will fit within a *fairly* small SRAM (on the order of a few million operands). Applications with large data sets can stream inputs via a system bus and there is inbuilt latency tolerance for this.
Processing any particular input typically requires a set of operational steps. These are listed as a programme for a sequencer unit which schedules each operation (exploiting any provided parallelism) and resolves possible data hazards.

Although reducing the *size* of operands represents a useful saving in memory transfers (and, hence, energy) it is apparent that further savings can be made by *omitting* operands whose effect on the result is 'insignificant'. This may be highly application-dependent but experiments suggest that the 'degradation' in output performance may not be significant with (say) ~80% of the operands omitted in any given calculation. Assessment of 'degradation' is difficult since the original standard is, itself an approximate value; furthermore if the network is subject to losing operands whilst being trained it may well adapt to this better. It is clear, however, that there is significant promise in reducing the number of calculations by such a factor.

The effect on memory transfers is partly mitigated by the need to specify *which* operands are omitted in any given weight matrix since their position is 'random'. Exact details depend on the operand size, the matrix dimensions and the 'sparseness': a rough guide might be that a 'sparse' matrix requires about twice the representational size, multiplied by its sparseness so if 20% of values are retained the cost is reduced to around 40%. This is largely independent of the operand size so overall savings in bits transferred may be reducible by 90% or more.

Above this, it should be noted that the system is scheduled by hardware so (after set up) there is little need for software intervention, saving instruction fetch and decode costs. The hardware implementation is still programmable to accommodate different weight matrix representations, permitting experimentation with this mechanism.

A further form of ANN operation supported is the Convolutional Neural Network (CNN) [3], of particular interest in signals' analysis. Effectively these have a sparse weight matrix ('kernel') which is applied repeatedly, 'scrolled' in each iteration. Here only the non-zero area of the matrix is stored (in local SRAM) and the matrix multiplier includes options to manipulate weight fetches accordingly.

There is an interest in Dynamic Neural Networks (DyNNs) [4] within this work-package. Provided that a system is constructed so that the operands are available in an appropriate memory partition, the ability to adjust the depth of a network 'on the fly' is implicit in the system construction: it becomes a matter of scheduling operations appropriately. However during the design phase a new mechanism for adjusting (reducing) the number of operations may

have emerged. In some circumstances it may be possible to produce adequate results with even sparser weight arrays than those mentioned above. These could be applied and augmented by further weights only when the output from that layer appears inadequate for purpose; the result being a degradation in a particular layer (or layers) rather than the omission of processing stages. In a deep network, where the information representation is evolving layer-by-layer, this may be more fidelitous than the layer-dropping approach.

An observation about any form of DyNN is that the 'quality' of an intermediate result must be adequately (and cheaply) assessed. It is not clear if this is possible at reasonable cost although examination of the problem from the hardware implementation perspective has elicited some ideas using statistics which can be gathered quite cheaply during the operation which will be investigated further. If this can be realised it can contribute further, complementary energy savings.

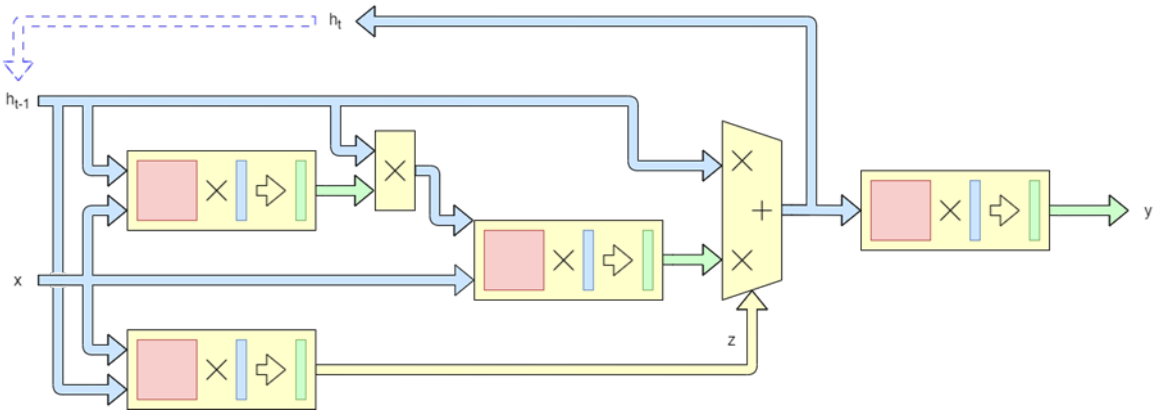

Figure 12. GRU structure – each yellow block represents a matrix multiply-accumulate and threshold unit, potentially the same physical hardware time-division multiplexed. The yellow mux represents the Hadamard product unit, combining two streams by a fraction specified element by element using a third stream.

Parameters for hardware configuration

For the matrix multiplication block, including the programmable threshold function block, the following parameters can be supplied:

TABLE 6: DIFFERENT ARCHITECTURAL PARAMETERS AND THEIR VALUE RANGES.

Name	Value range	Notes
ADDR_SZ	0→32	Max address range for activations, potentials and weights
MAX_ACT_SZ	0→32	Max # bits for activation value
MAX_WEIGHT_SZ	0→32	Max # bits for weight value
MAX_POT_SZ	0→32	Max # bits for potential values
GUARDBITS_SZ	0→16	# upper bits of accumulator discarded (with saturation) when writing back to memory

ACC_SHIFT_SZ	0→16	Relative shift of product when being added to accumulator
Threshold unit		
ADDR_SZ	0→32	Max address range for potentials
INPUT_ARG_SZ	0→32	# bits of potential to keep when performing look-up
OUTPUT_SZ	0→32	# bits in output activation

For the Hadamard Unit, the following parameters can be supplied:

Name	Value range	Notes
ADDR_SZ	0→32	Max address range for activations
MAX_ACT_SZ	0→32	Max # bits for activation value
FRAC_SZ	0→32	Max # bits for fractional mixer value
OP_ACT_SZ	0→32	Max # bits for output activation value

The GRU consists of three fully connected feed-forward layers and two Gated Recurrent Units (GRU). A single GRU consists of five fully connected feedforward layers and one Hadamard product unit. There are no additional parameters for these compound structures, only the parameters for each individual component.

Parameters for software (dynamic) configuration

During normal operation, each physical hardware accelerator may be scheduled to process input, which may be its own output or from another accelerator. The type of operation (fully connected, sparsely connected or convolutional) can be specified, along with the data widths and the physical addresses of the source and destination data streams.

Note that for sparsely connected layers, each weight is a two-value pair: the weight value along with the index of the source neuron (its ID). This index is used to access the activations memory instead of assuming a strictly linear series of indices. For convolutional mode, the kernel dimensions are sufficient to generate the kernel memory address stream.

TABLE 7: PARAMETERS FOR DYNAMIC CONFIGURATION.

Name	Value range	Notes
OP_TYPE	0→2	0 = fully connected, 1 = sparsely-connected, 2 = convolutional
WEIGHT_SZ	0→ MAX_WEIGHT_SZ	# bits in weights for current task
WEIGHT_IDX_SZ	0→4	# bits in the source neuron ID field inside each weight data structure (used in sparse connectivity mode only)
WEIGHT_VAL_SZ	0→MAX_WEIGHT_SZ	# bits in the weight value inside each weight data structure (used in sparse connectivity mode only)

ACT_SZ	0→MAX_ACT_SZ	# bits in activations for current task
KERNEL_X	0→15	Width of convolution kernel (only used in convolutional processing)
KERNEL_Y	0→15	Height of convolutional kernel (only used in convolutional processing)
INPUT_STREAM_X	0→2048	Width of the input image
INPUT_STREAM_Y	0→2048	Height of the input image

2.5. Dynamic SNN accelerator

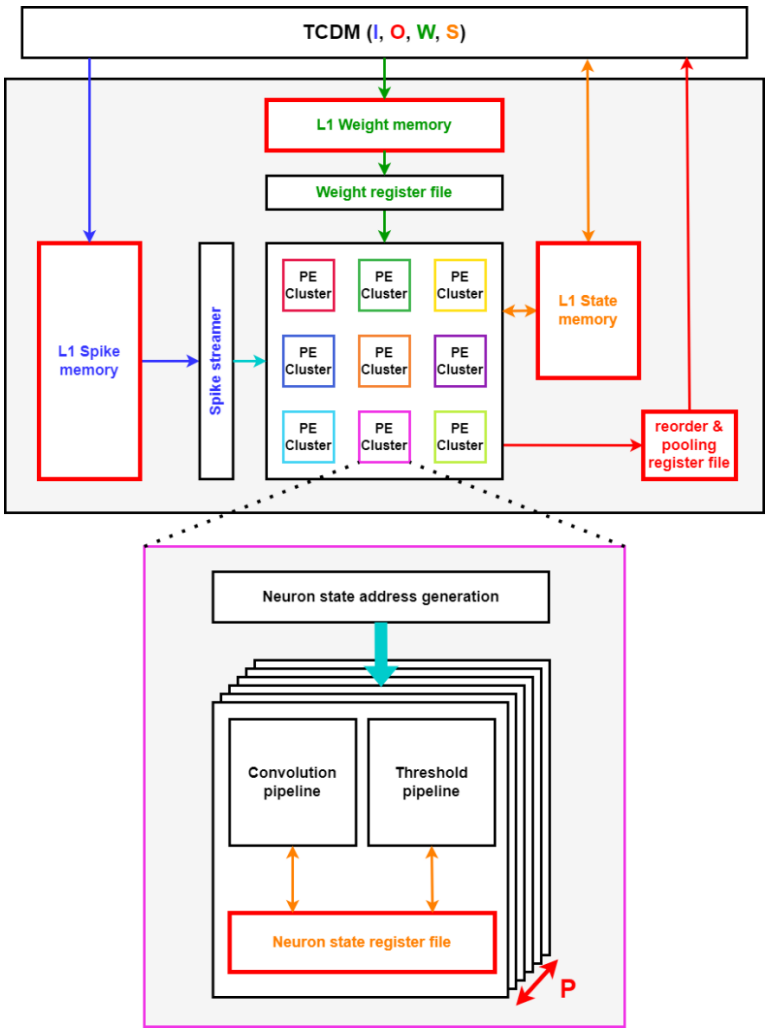

Figure 13 The design time parameters of the SNN accelerator architecture. The parameters include memory size, memory widths and P parallel PEs.

The SNN accelerator has many parameters that need to be decided during the design time, as shown in Figure 13. These design time parameters have a significant impact on the latency, accuracy, area and energy of the SNN accelerator. By changing the parameters, we can have

different tradeoff points between the metrics. So, to find the best tradeoff for the use case, there is a need for a simulation model to determine the metrics at the different tradeoff points. Next to the flexibility of design time parameters, the schedule on the SNN accelerator can be changed. Different schedules can affect the latency and energy of the SNN accelerator. Therefore, we need a simulation model to explore the possible schedules.

The Dynamic SNN accelerator is modelled at two levels of abstractions, the RTL model and the ZigZag model. The two models are described below.

RTL model

The RTL model can provide an estimate of the latency, accuracy, area and energy of the SNN accelerator. To get an estimate of the latency and accuracy, the RTL model can be simulated using a cycle-accurate RTL simulator, like Cadence Xcelium or Verilator. For the simulation, the user must provide the parameters and the input spikes of an SNN model. For an estimate of the area of the accelerator, we can synthesize the RTL model, for example with Cadence Genus. Cadence Genus provides an area estimate after synthesis, based on information in the LEF libraries. The area estimate consists of the cell area after synthesis and an estimated post-route net area. To get an estimate of the energy, the simulation and synthesis need to be combined. After synthesis, we have a netlist of the accelerator. We can run the same SNN simulation used for the latency and accuracy, however, the RTL model is replaced with the synthesized netlist. In this simulation, we can extract the switching activity of all nets in the accelerator. The switching activity can then be used in Cadence Genus to get an accurate energy estimate of the accelerator.

ZigZag model

While we can get accurate estimates with the RTL model, the runtime of both synthesis and simulation is in the order of multiple hours. For DSE, we need a more abstract model, which can be evaluated faster. The abstract model will be made using the ZigZag framework. However, ZigZag cannot be used for SNN models and accelerators. To support SNN models and accelerators, three extensions were made to ZigZag:

1. *Support for neuron states:* Neuron states are internal memory values that evolve over time. A LIF neuron has one internal state: the membrane potential. Neuron states are implemented as extra operands. Each timestep, the neuron states have to be loaded into the accelerator, updated, and stored back somewhere in the memory hierarchy. By modelling the neuron states as operands, the effects of their mapping on memory access, data re-use, data transfers and allocated memory are reflected in the cost model.
2. *Support for the time dimension:* Due to the internal state of the LIF model, the scheduler needs to also map execution in the time dimension. This increases the mapping space by an order of magnitude. However, the mapping has constraints that ensures chronological evaluation and update of neuron states. This reduces the mapping space significantly. Also, similar to depth-first schedules, we can explore time-first schedules. In time-first schedules, (part of) a layer is executed for multiple timesteps, before moving to the next layer. This increases the reuse of neuron states significantly, at the cost of larger feature maps between layers.

3. *Support for sparsity:* Input sparsity is an important factor for the energy and latency of an SNN accelerator. Therefore, input sparsity is taken into account for the ZigZag model. The extension accepts spike traces from the execution of an SNN to compute the input sparsity of each layer. For each layer, the compute energy and latency is scaled down by the input sparsity. Moreover, the register accesses of an operand can be scaled down by the input sparsity.

With these extensions, ZigZag can be used to model the SNN accelerator. Moreover, we can use the DSE capabilities of ZigZag to explore different schedules.

3. CIM accelerators and their performance models

3.1. Adder-tree Free SRAM-based CIM (AFSRAM-CIM) architecture

Adder-tree Free SRAM-based CIM (AFSRAM-CIM) architecture (TUD) – The performance models of SRAM-based digital CIM can be generalized by energy modeling as energy is an integral of power consumption over time (delay). Therefore, accurate energy consumption model plays a crucial role to evaluate the efficiency of a digital CIM (DCIM) and perform design space exploration. Thus, to evaluate the performance of adder-tree free SRAM-based CIM (AFSRAM-CIM) and conduct design space exploration, we adopt the analytical model for SRAM-based DCIM presented in [1]. As shown in Figure 14, AFSRAM-CIM has two main components, namely a multi-bank SRAM array for weight storage and multiplication operations and digital periphery for accumulation operation.

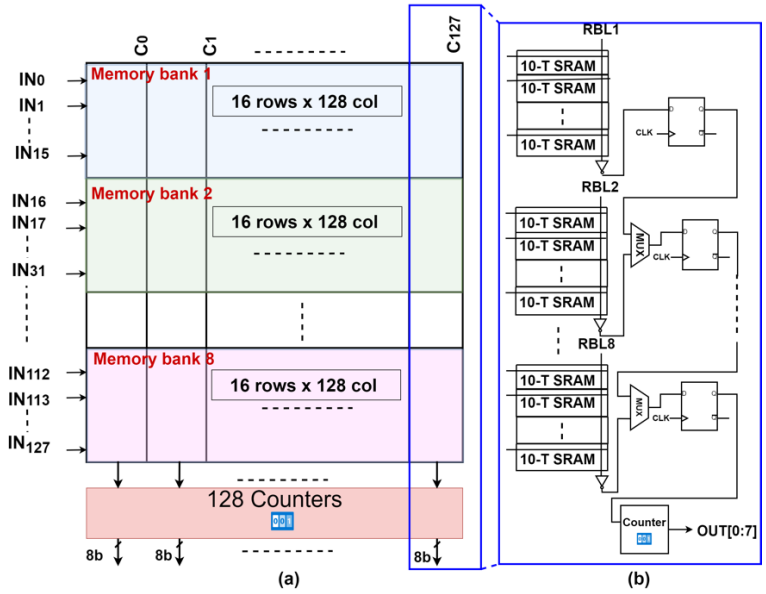

Figure 14: AFSRAM-CIM architecture (a) SRAM array for storage and embedded multiplication (b) counter based accumulation

The Multiply Accumulate (MAC) operation in AFSRAM-CIM is realized in two stages; the first stage is in-cell multiplication of the input with the stored weight while performing a read

operations and the second stage is accumulation of the product at the periphery. Therefore, the total energy of AFSRAM-CIM can be computed as shown in Equation (1):

$$E_{total} = E_{mul} + E_{acc} \quad (1)$$

Where the total energy of the AFSRAM-CIM (E_{total}) is computed as the sum of the energy consumed by the multiplication and storage stage (E_{mul}) and the energy consumption of the accumulation stage at the periphery (E_{acc}).

Energy cost model for in-cell multiplication

In AFSRAM-CIM the SRAM cells are modified (from 6 transistor to 8 transistor cell) to allow "free" bitwise multiplication by leveraging the memory read operation. Since the bitwise multiplication is embedded within the bit-cell, it has two advantages over the conventional DCIM architectures. On the one hand it avoids the need of extra logic to perform the multiplication operation and on the other hand it leads to energy saving as the multiplication operation consumes only the bit-cell read energy. Thus, to perform bitwise multiplication operation the input vector values are propagated along the wordlines, and the read operation, embedded multiplication is performed by discharging the bitlines depending on the input value and the weight value stored in the memory cells as shown in Figure 2. The energy required to execute this operation E_{mul} can be expressed as the energy related to the charge/discharge of internal capacitances related to the memory cell read operations (E_{read}). Therefore, E_{mul} is equivalent to E_{read} as shown in Equation (2).

$$E_{mul} = E_{read} \quad (2)$$

In an SRAM-based DCIM E_{read} is the energy consumed during read operation mainly due to charging and discharging of the capacitances across the SRAM array wordlines (WL) and bitlines (BL). Thus, E_{read} can be expressed as:

$$E_{read} = (N.E_{wl} + M.E_{bl}).P_{cycle} \quad (3)$$

Where N is the total number of rows (wordlines in the SRAM array), M is the total number of columns (bitlines in the SRAM array). Similarly, E_{wl} and E_{bl} are the energy consumed by charging and discharging the wordline and bitline capacitances, while P_{cycle} is the precharging cycle of the read bitlines.

The wordline energy (E_{wl}) is modelled as:

$$E_{wl} = C_{wl}.{V_{dd}}^2.B_w \quad (4)$$

Where C_{wl} is wordline capacitance, V_{dd} is the supply voltage and B_w is the weight value stored in the cell. Similarly, the bitline energy (E_{bl}) can be modelled as:

$$E_{bl} = C_{bl}.{V_{dd}}^2.B_w \quad (5)$$

Where C_{bl} is bitline capacitance, V_{dd} is the supply voltage and B_w is the weight value stored in the cell.

Energy cost model for accumulation

In AFSRAM-CIM accumulation is performed using counter and register based approach. The accumulation is achieved by latching the in-cell multiplications and feeding them to counters and register for addition and accumulation. Therefore, the accumulation energy (E_{acc}) component of Equation (1) is modelled as:

$$E_{acc} = G_{cap}.G_{cnt}.{V_{dd}}^2.Acc_{cc}.M.Op_{cnt} \quad (6)$$

Where G_{cap} is the gate capacitance of a standard logic gate, G_{cnt} is the gate count (total number of gates) in an accumulator unit and V_{dd} is the supply voltage. Acc_{cc} is the cycle time of a single accumulation operation, M is the total number of columns (accumulator units) in the architecture while Op_{cnt} is the total number of accumulation operations.

These analytical models are scalable to provide energy-estimation of an arbitrary configuration. Moreover, they can be easily extended to model different DCIM configurations. For instance, to model an adder tree based DCIM with in-cell multiplication, only the E_{acc} model needs to be modified to include the adder tree components.

3.2. Fibbinary multi-bit multiplier Approximate DCIM architecture

DSE for DCIM is essential, particularly in evaluating parameters such as the number of SRAM cells within a column, the count of partitions, and the supported bit width. These parameters hold significant sway over the efficiency and effectiveness of DCIM architectures. One way to facilitate this DSE is through the utilization of analytical models. These models serve as tools for simulating and analyzing various architectural configurations, allowing researchers to assess the impact of different parameters on system performance and energy efficiency.

An existing analytical model for Compute in Memory [1] has provided valuable insights into its operation. However, the integration of innovative strategies, such as multi-bit multipliers and deviation from the standard adder tree, necessitates changes to the existing analytical models. The analytical model discussed below is based on the one described in [1]. For a more in-depth analysis of the original model it is recommended to read the original paper.

This section presents an analytical model, building on prior works, to keep it functional for the proposed architecture. First the parameters, both user defined and derived, will be discussed followed with the modified formulas. References to parameters outlined in tables below can be traced back in Figure 15, offering a visual aid in contextualizing the analytical model's scope and implications. This model is limited to energy needed for a certain operation; area is excluded. Lastly, the weights are assumed to be stationary for the model.

TABLE 8: USER DEFINED HARDWARE MODEL PARAMETERS

Parameter	Description
H	Number of SRAM cells inside a column
W	Numbers of SRAM cells inside a row
B_w	Bitwidth of Activation/Weight
P	Number of CIM macros

TABLE 9: DERIVED PARAMETERS

Parameter	Description
C_{RL}	Readline capacitance per cell

C_{Mult}	Multiplier capacitance per cell
C_{FA}	1-bit Full adder capacitance per cell
V	Supply voltage
D_1	Activation propagation axis size of the CIM array
H_w	Weights bits stored in parallel in the CIM
F	Total number of 1-b full-adders in the adder tree

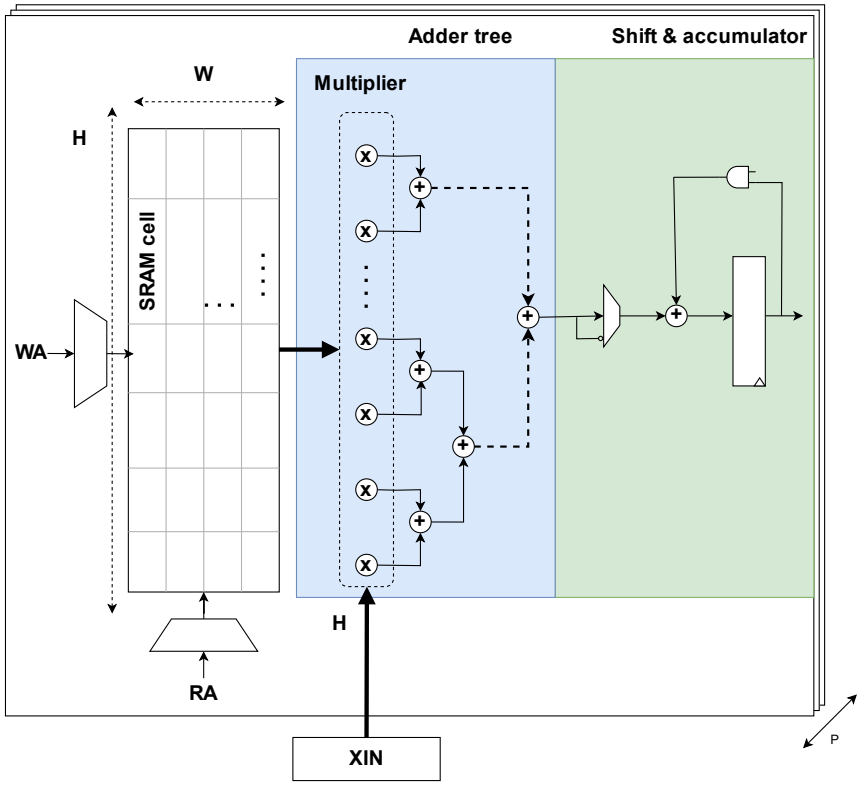

Figure 15, Architecture overview of Fibbinary multi-bit multiplier DCIM

As deduced from Figure 15, the primary energy contributors include SRAM, multipliers, adder trees, and bitshift and accumulate operations. Additionally, some peripheral logic is necessary to manage SRAM decoders, control the number of bitshifts, and store input bits. The primary formula for the analytical model of the DCIM architecture is expressed as:

$$Etotal = Emul + Eacc + Eperipherals$$

The energy consumption for multiplication comprises the energy required for reading SRAM cells (E_{cell}) and logical operations (E_{logic}):

$$E_{mul} = E_{cell} + E_{\text{logic}}$$

Where E_{cell} is determined by factors such as readline capacitance per cell (C_{RL}), the number of weight bits per operand (H_w), and D_1, representing the number of operands per memory row:

$$E_{cell} = C_{RL} \cdot V^2 \cdot H_w \cdot D_1$$

To adjust for the number of MACs, the number of SRAM cells in a column needs halving since we perform 2xN multiplications, leading to the following formula:

$$E_{\text{log ic}} = V^2 \cdot C_{\text{mult}} \cdot \left(\frac{H}{2}\right)$$

Here, C_{gate} denotes the capacitance for the custom multiplier, and (H/2) equals the number of multipliers per macro.

Post-multiplication, accumulation is performed in the adder tree, followed by a bitshift and accumulation. The energy formula is as follows:

$$E_{\text{acc}} = E_{\text{addertree}} + E_{shift}$$

For the adder tree:

$$E_{addertree} = C_{FA} \cdot V^2 \cdot F$$

Where C_{FA} represents the gate capacitance of a single full adder, and F denotes the number of one-bit full adders, calculated as:

$$F = \Sigma_{i=0}^{\log_2(H)}(5+i) \cdot \left(\frac{H}{2^i}\right)$$

The bitshift and accumulate operations scale with both B_w and H, necessitating the development of an analytical model.

Peripheral energy ($E_{peripherals}$) includes energy consumed by the input buffer and decoders. The input buffer's energy consumption scales linearly with H and slightly with B_w. Assuming partitions are sufficiently large, the input buffer's area consumption can be disregarded. Although decoder complexity scales with both H and W, the increase in energy consumption is outweighed by the adder tree and SRAM, thus justifying its exclusion for complexity considerations. Any additional logic energy consumption is negligible.

This analytical model will be used to derive the architecture with the best energy per operation for the user defined architecture.

3.3. An RRAM-based analog CIM accelerator with self-healing

Figure 16 shows the high-level architecture of RRAM-based analog CIM accelerator using serial current driver approach that utilizes a very low constant current source to drive the bit-cells, connected in series, and perform MAC operation in the form of a voltage output.

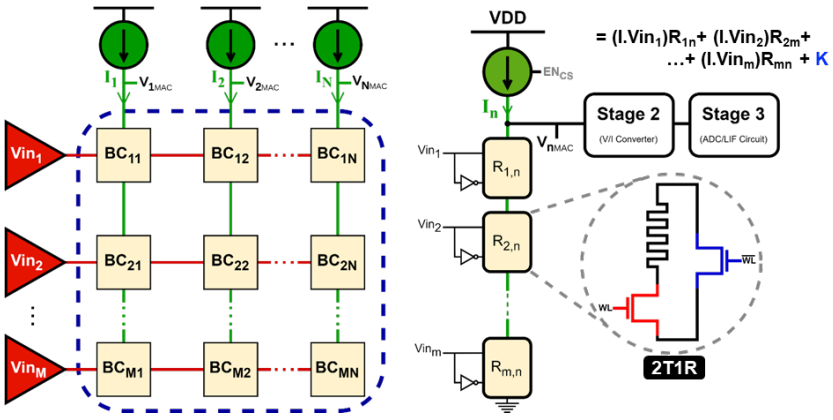

Figure 16: RRAM-based analog CIM accelerator.

In analog domain, current and latency are the main contributing factors in determining power as well as energy efficiency of the system. Thus, it is essential to optimise these parameters based on the requirement of the application. The power consumed by a crossbar is given as:

$$P_{xbar} = V_{DD}.I_{xbar}$$

Since each column in our crossbar design is driven using a constant current (I_C), the power is constant for a given crossbar size and can be given as:

$$P_{xbar} = V_{DD} \times I_C \times \left\lceil \frac{M}{m} \right\rceil \times N$$

where N is the no. of columns in the given crossbar, M is the required no. of rows and m is actual no. of devices placed in series within a column. For a particular column, the power (P_C) can be given as:

$$P_C = V_{DD} \times I_C \times \left\lceil \frac{M}{m} \right\rceil$$

Thus, the value of I_C directly affects the overall power consumption of the crossbar and by decreasing this current, the power can be reduced further. This approach might be good in limiting the power, especially for an edge device powered by a battery, where the current must be within the limits of the capability of the battery. However, this does not guarantee equal reduction in energy consumption, which for a given column, can be expressed as:

$$E_C = V_{DD}.I_C.T_{MAC}\ (Static\ energy) + \frac{1}{2}.C_L.(V_{MAC})^2\ (Dynamic\ energy)$$

where T_{MAC} is the latency of the crossbar and can be further express as:

$$T_{MAC} \approx 5 \times C_L \times \frac{V_{MAC}}{I_C}$$

where V_{MAC} is the output voltage of a column analogous to MAC operation and C_L is the equivalent output load capacitance. Additionally, C_L in itself is a function of no. of devices (m) in a column and can be given as:

$$C_L = f(m)$$

where the function is a complex relation between the no. of devices in a column (m), input states of these devices as well as the data stored in each devices. However, we can fairly assume that the C_L will increase as more and more devices are added in series within a column, such that $\frac{\partial f}{\partial m} > 0$.

Thus, the overall energy efficiency of a column can be express as:

$$E_C = V_{MAC} \times C_L \times (5.V_{DD} + \frac{V_{MAC}}{2})$$

$$\Rightarrow E_C \approx V_{MAC} \times f(m) \times (5.V_{DD})$$

It can be seen that the energy consumed by a crossbar column is independent of the column current and is primarily dependent on its final output value (V_{MAC}) as well as the load capacitance, which is indirectly dependent on the no. of devices (m) placed in series within a column. In contrast, the power of the column is inversely proportional to 'm' as given by the equation of P_C . Thus there exist an optimum no. of devices (m) per column at which the

crossbar achieves highest power as well as energy efficiency and this becomes one of the critical parameter while performing Design Space Exploration.
Furthermore, supply voltage is another important parameter which affects both power and energy efficiency and can be further improved with voltage scaling.

Second, peripheries are another major contributing factor towards the overall efficiency of the system and must be designed carefully, which mainly depends upon the intended application. However, a key aspect to consider is array-periphery co-design i.e. not only focussing on improving the functionality of the crossbar but also ensuring that the periphery design complements the crossbar's functionality. This synergy is important to prevent any individual components within the system flow, typically peripheries, from becoming bottlenecks and make the entire system operate efficiently. Moreover, research efforts are required to enable crossbar columns, mimicking a neuron, to intercommunicate among each other in analog domain while the periphery is required only at the end to convert the final prediction into digital values. This can greatly enhance the overall system efficiency by reducing the reliance on power-intensive peripheries for every column, particularly in dense neural networks where such dependence can severely compromise efficiency.

Third, reliability of the memories is another challenge which must be addressed to maintain accurate functionality of the CIM micro-architecture based on the proposed crossbar. Several techniques have been proposed to address this challenge, categorized into hardware-side solutions, such as self-healing architectures [2], and software-side approaches [3]. In a self-healing architecture, the approach is straight-forward where a monitoring circuit constantly checks for the deviation in memory resistance states (Read-disturb) via a dummy column and triggers the rewriting of these memories, like DRAM, once the deviation exceeds a particular threshold. Read-disturb is a non-deterministic process and depends upon various factors including the input states and the stored data. Subsequently, most of the techniques rely on a simple approach by periodically refreshing the weights. However, frequent rewriting can detrimentally impact system efficiency, not only because of significant energy consumption, which can be many times greater than that of the read operation but also due to the resulting downtime during rewriting process.

Thus, better monitoring schemes are required that triggers rewriting only when there is actual necessity. Moreover, the best possible solution would be to mitigate the problem of read disturb by employing very low read currents, as in the case of this crossbar or having bi-directionality. Bi-directionality allows the periodic reversal of read current direction through memories while retaining functionality, thus compensating for resistance deviations without necessitating rewriting. This aspect is a primary focus of the proposed crossbar design, leveraging constant current to perform MAC operations. By incorporating another constant current at the column's other end, the current direction can be reversed without altering the underlying MAC operation. From the software aspect, the robustness of a neural network can be leveraged by exploring various training techniques whereby the neural network can be trained while suppressing a particular problematic state, depending upon the memory technology. This approach ensures that the trouble-some weights are minimized in occurrence, reducing errors resulting from deviations in these states to negligible levels.

Therefore, the aforementioned strategies summarises various avenues of improvement for the proposed crossbar based CIM architecture. These improvements can be achieved not only through hardware optimisation but also via software mitigation, provided the availability of adequate experimental data for the memory technology being used.

4. Summary

This deliverable presented the progress on hardware accelerators and the performance models of the different hardware accelerators developed in Convolve project. First we discussed the performance models of digital accelerators and then the CIM based accelerators. As a next task, we will perform DSE of the aforementioned accelerators using the ZigZag or GVSoC framework.

Bibliography

[1] P. H. V. J. S. G. a. M. V. L. Mei, "ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators," in *IEEE Transactions on Computers*, 2021.

[2] G. H. G. T. F. C. L. B. a. D. R. N. Bruschi, "GVSoC: A Highly Configurable, Fast and Accurate Full-Platform Simulator for RISC-V based IoT Processors," in *IEEE 39th International Conference on Computer Design (ICCD)*, 2021.

[3] A. B. P. K. B. V. H. V. h. a. M. M. B. Van Den Broeck, "Time-domain generalized cross correlation phase transform sound source localization for small microphone arrays," in *5th European DSP Education and Research Conference (EDERC)*, 2012.

[4] E. V. K. W. M. J. P. &. C. H. de Bruin, "R-Blocks: an Energy-Efficient, Flexible, and Programmable CGRA," in *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[5] D. E. J. L. M. a. P. R. G. Rumelhart, "Parallel distributed processing, volume 1: Explorations in the microstructure of cognition: Foundations.," in *The MIT press*, 1986.

[6] K. B. V. M. C. G. D. B. F. B. H. S. a. Y. B. Cho, ""Learning phrase representations using RNN encoder-decoder for statistical machine translation."," in *arXiv preprint arXiv:1406.1078*.

[7] A. I. S. a. G. E. H. Krizhevsky, ""ImageNet classification with deep convolutional neural networks."," in *Communications of the ACM 60, no. 6 (2017): 84-90*.

[8] Y. G. H. S. S. L. Y. H. W. a. Y. W. Han, ""Dynamic neural networks: A survey."," in *IEEE Transactions on Pattern Analysis and Machine Intelligence 44, no. 11 (2021): 7436-7456*, 2021.

[9] P. S. J. &. V. M. Houshmand, "Benchmarking and modeling of analog and digital SRAM in-memory computing architectures," *arXiv preprint arXiv*, 2023.

[10] M. A. Yaldagard, S. Diware, R. V. Joshi, S. Hamdioui and R. Bishnoi, "Read-disturb detection methodology for rram-based computation-in-memory architecture," in *IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2023.

[11] S. Diware, A. Singh, A. Gebregiorgis, R. V. Joshi, S. Hamdioui and R. Bishnoi, "Accurate and energy-efficient bit-slicing for RRAM-based neural networks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, pp. 164-177, 2022.

[12] 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)*.

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

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

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

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

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

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

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

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

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

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

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

[38] V. Leon, K. Asimakopoulos, S. Xydis, D. Soudris and K. Pekmestzi, "Cooperative Arithmetic-Aware Approximation Techniques for Energy-Efficient Multipliers," in *56th ACM/IEEE Design Automation Conference (DAC),* Las Vegas, NV, USA, 2019.

[39] S. Hashemi, R. I. Bahar and S. Reda, "DRUM: A Dynamic Range Unbiased Multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD),* Austin, TX, USA, 2015.

[40] V. Mrazek, R. Hrbacek, Z. Vasicek and L. Sekanina, "EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE),* Lausanne, Switzerland, 2017.

[41] V. Leon, G. Zervakis, D. Soudris and K. Pekmestzi, "Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 26, pp. 421-430, 2018.

[42] K.-J. Cho, K.-C. Lee, J.-G. Chung and K. K. Parhi, "Design of low-error fixed-width modified booth multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 12, pp. 522-531, 2004.

[43] T. Homma, L. Atlas and R. M. II, "An Artificial Neural Network for Spatio-Temporal Bipolar Patters: Application to Phoneme Classification," *Advances in Neural Information Processing Systems,* vol. 1, p. 31–40, 1988.

[44] G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger, *Densely Connected Convolutional Networks.,* 2018.

[45] M. Adriaansen, M. Wijtvliet, R. Jordans, L. Waeijen and H. Corporaal, "Code Generation for Reconfigurable Explicit Datapath Architectures with LLVM," in *2016 Euromicro Conference on Digital System Design (DSD),* Limassol, Cyprus, 2016.

[46] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris and K. Pekmestzi, "Design-Efficient Approximate Multiplication Circuits Through Partial Product Perforation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 24, pp. 3105-3117, 2016.