GemIMC: A Configurable HW Architecture for Technology Agnostic IMC based NN Inference

Emilien Taly, Roberto Guizzetti, Pascal Urard, Ioana Vatajelu

GemIMC: A Configurable HW Architecture for Technology Agnostic IMC based NN Inference

Emilien Taly*†, Roberto Guizzetti*, Pascal Urard* Elena-Ioana Vatajelu†
*STMicroelectronics, Crolles, France
†Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, F-38000 Grenoble, France

***Abstract*—This paper presents GemIMC, a High Level Synthesis (HLS) based configurable digital unit architecture for accelerating Neural Networks (NN) at the edge using In Memory Computing (IMC). The proposed architecture is capable of supporting any type of memory technology, be it CMOS or resitive, with digital or analog storage capabilities. GemIMC aims to facilitate design space exploration among different IMC-related parameters and provide a top architecture for prototyping. By taking as input the IMC-tile parameters such as storage type (analog/digital), size, latency, power and process variability, GemIMC provides a an estimation of the full system's latency, area, and power consumption, taking into account the full digital control. The Tensorflow-to-GemIMC environment flow allows for direct evaluation of the ineference accuracy for any type of NN, considering the variability associated with analog computation (when needed). The proposed work provides a flexible and efficient solution for IMC-based NN inference on the edge, along with a methodology for writing the IMC tile model to ensure compatibility with the top architecture.**

I. Introduction

In recent years, Deep-Edge computing has become essential in various systems by bringing computations close to the sensors. This approach saves a significant amount of energy by running only a wake-up system continuously in the background [1] and it typically includes a Neural Network (NN) accelerator to perform computations and make decisions. The Deep-Edge environment is one of frugal resources, therefore, the notions of energy and surface efficiency are critical. To comply with energy constraints more and more architectures based on In Memory Computing (IMC) for NN computations have emerged. The current research focuses on the development of the IMC-tile. An IMC-tile is a essentially a memory block that has computation capabilities; it contains the memory array and the periphery (address decoders and read/write drivers) specifically designed to enable Multiplications ACcumulations (MACs) directly in memory. Even though they are all designed to perform the same computation, their operation mode, performance and power efficiency depend strongly on the type of memory employed, such as Static Random Access Memory (SRAM) or Non Volatile Memory (NVM). Moreover, at the architecture level, the digital control which encompasses the tile, is also diverse and dependent on factors such as computing parallelism, NN model mapping, or weights- and data-encoding [2]–[10]. The state of the art presents numerous solutions of NN inference accelerated by IMC but they are either build from bottom-up, which means that the proposed architectures are custom designed for a specific memory technology and operation mode, or top-level only where the IMC-tile is seen as a black box which outputs the MAC result. Besides many other works are only focusing on the design and optimization of the IMC-tile without considering the architecture-level implications, despite the fact that the architecture view can play a crucial role in evaluating performance based on the intended application. In addition, several frameworks exist such as NeuroSim [11] [12], which provide performance estimations of specific IMC-tile accelerating NN inference without considering a top-down view, while the framework proposed in [13] addresses this limitation but it is limited to only two types of SRAM architecture. Therefore, further research is needed to explore the potential benefits of a comprehensive approach that considers both the memory device specificity and the architecture level design.

This paper introduces GemIMC, a technology agnostic and configurable hardware (HW) architecture to control an IMC-tile for NN inference. The architecture is based on Gemini, a Neural Processing Unit (NPU) for NN inference introduced in [14] and [15]. Specifically, we consider small NN models and map the entire model onto the tile without weights reloading. To evaluate a IMC-tile, we propose a flow that takes the behavioral model of the IMC-tile as input, including power consumption estimation and analog variability for Analog IMC (AIMC) and outputs the characteristics of the resulting NN implementation such as inference accuracy, and estimation of the area, latency and power consumption. The proposed flow also supports Digital tile (DIMC).

The objective of this study is to understand the impact of various IMC-tile parameters on digital HW cost and latency for a given NN model. Additionally, we investigate the different in term of parallelism and NN models mapping for our architecture. The proposed configurable HW architecture is designed in C++ using High-Level Synthesis (HLS) and can be directly integrated with one or many tiles at the Register Transfer Level (RTL). This paper is organized as follows. Section II presents an overview of the current state-of-the-art in IMC-tile design and existing frameworks for their evaluation. Section III describes the various parameters of the proposed architecture, their respective impacts on the design and the dataflow for only single tile. Section IV explains the evaluation metrics of the proposed architecture such as latency and digital HW cost function of architectural and NN parameters. Section V outlines the methodology for writing the IMC-tile model and presents the results for one NN model. Section VI concludes the paper.

II. Related work

The types of memories used for IMC range from SRAM [2]–[5] to NVM with PCM [6] [7] or RRAM [8]–[10] cells. This technology versatility leads to a large variety of IMC-tile designs, each taking advantage of specific technology characteristics - such as type of memory point (charge-based or resistance-based), type of data storage (analog or digital) and ability to access multiple memory cells simultaneously - and implementing different strategies for performing the MAC operations. The points of difference between existing IMC-tiles can be classified into seven categories: input encoding (binary [5] [7] [9] [10] or on multiple bits [2], [3] [4] [6] [8], physical computation (current-based or charge-based [5]), memory cell (SRAM or NVM with single [8] or multi level storage [6] [7] [9] [10]), implementation of analog-to-digital converters (ADC), parallelism (maximum number of WordLines (WL) and BitLines (BL) that can be activated at the same time), weight mapping scheme (the order in which the 3D filter weights will be unrolled on a column, and the way in which the weight bits will be mapped), and data/weights precision.

All these device-related differences between IMC-tiles and the large spectrum of operation modes lead to unique functionality of the digital control. This complexity also makes it difficult to compare the effects of using different IMC solutions at system level or even to comprehensively estimate the performance and accuracy of an NN implemented using IMC. To alleviate these issues, frameworks have been developed in support to IMC-tile evaluation in the NN inference context, such as Neurosim [11]. This C++ environment supports SRAM or NVM tile (digital or analog), and accepts different levels of abstraction, ranging from a cell-to array-level. DNN+Neurosim [12] adds a circuit-level view with consideration of the digital control architecture of the IMC-tile. It also includes an algorithm-level view with a wrapper to interface NeuroSim and Tensorflow (TF) for the software part of the NN.

The limitations of these frameworks lie in their configurability. For input encoding, only binary is supported. In terms of parallelism, the assumption is that all WLs can be and are activated (even though some architectures [2] [7] [8] [9] activate only a portion for calculations). This has an impact on the digital HW part. Additionally, the user must provide a tile size equal to the NN size, which can be limiting in cases where a fixed tile size needs to be tested on different NN sizes or if the technology does not allow for the fabrication of large tiles (due to variability or other constraints). Reference [13] focuses more specifically on the impact of the digital control on the NN inference. However, they only focus on SRAM-based IMC-tile with two types of architecture, supporting voltage or pulse encoding. Parallelism is not taken into account, and only one type of weight mapping is considered.

In this paper we propose GemIMC which goes beyond the state of the art solutions by being flexible enough to allow the use of any IMC-tile design and operation mode, the parallelism at different levels and it includes the digital HW for realistic cost and performance evaluations.

III. Architecture description

The GemIMC environment consists of a Python wrapper, the technology-agnostic GemIMC core in C++, and a metrics calculation block, as shown in Figure 1. The Python wrapper describes the NN model and performs the training taking into account TF quantization using Quantization Aware Training (QAT). The GemIMC core includes the HLS of the digital HW architecture which to controls the IMC-tile and performs the calculations that are not IMC compatible. The IMC tile model provides the MACs result at each cycle. Metrics such as inference accuracy, latency, power, and surface are then calculated.

Moreover, the IMC performance and usability is evaluated on a the single-tile architecture. The latter allows for higher parallelism and it could be the optimal choice depending on the latency requirements of the application.

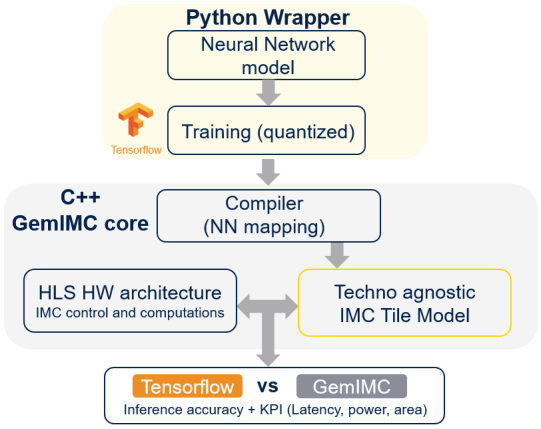

Fig. 1: General overview of GemIMC

A. Configurable parameters

The implemented architecture is user-configurable according to the parameters list shown in Table 1. Where the first column contains the parameter notation, the second column indicates to which type of IMC this parameter is applicable: AIMC (A), DIMC (D), or both (A/D), while the third column briefly describes the parameter. These parameters can be divided into 2 categories: (1) parameters that define the IMC-tile and (2) parameters related to the parallelism.

The parameters that define the IMC-tile are: the tile size (Tx, Ty), the number of states a memory cell can store: Level Cells (LC), the data precision (Dn), the weights precision (Wn), the number of data bits which can be processed simultaneously (DP), the number of Activated WordLines (AWL) and the number of Activated BitLines (ABL), the precision of ADC and of DAC (ADCn, DACn).

The LC parameter allows to support Single or Multi-Level Cell (SLC, MLC). It should be noted that the parameter is represented in terms of the bits number per cell. To enable encoding of a weight precision (Wn) on cells with a different number of levels, the value of LC can vary across the same tile (as described, for instance, in [7]). The DP parameter supports any type of input data encoding. Regarding the ADCs, their

precision may be lower than the maximum output range given by :

$$ADC_{n_{max}} = \lceil \log_2(2^{LC-1} \cdot 2^{DP-1} \cdot AWL + 1) \rceil$$

Their number may also be lower than ABL depending on the IMC-tile architecture. The maximum number of ADCs per tile is equal to ABL.

The parameters related to the parallelism are: MPAR, MACPAR, and BITPAR. MPAR parallelism refers to the calculation of pixels on multiple filters simultaneously. MACPAR parallelism refers to the calculation of multiple MACs operations of the same convolution simultaneously. BITPAR parallelism refers to the calculation of multiple data bits simultaneously. The maximum value of BITPAR is equal to the data precision (Dn).

The different parallelisms are illustrated in Fig. 2(a) on a convolution layer, while Fig. 2(b) shows the parameters for AIMC- and DIMC-tile as well as the differences between them. AIMC is based on analog computation through the activation of multiple WLs with a multiplication that can be on multiple bits. DIMC is based on deterministic computation through the activation of a single WL with binary multiplication and accumulation via an adder tree in the tile. Generally, for AIMC, the filter coefficients are placed on the different columns and the 3D filters are unrolled on the rows. Conversely, for DIMC, the filters are placed on the different rows and the 3D filters are unrolled on the columns. The parameters ADCn/DACn, LC, DP, and AWL are specific to AIMC. For a single-tile AIMC, the parameters AWL and DP are equivalent to MACPAR and BITPAR respectively. The filter parallelism, noted MPAR, is define by :

$$MPAR = \frac{ABL}{Cell_{per_{Wn}}} \quad with \quad Cell_{per_{Wn}} = \lceil \frac{Wn}{LC} \rceil$$

For DIMC, the parameters AWL and DP are equal to 1, while ABL defines the MACPAR parallelism on a single tile.

The rest of this chapter includes a detailed description of the single-tile architecture.

TABLE I: List and description of architectural parameters

Name	Memory	Description
Tx, Ty	A/D	Tile size (columns, row)
LC	A	Number of weights bits encoded on one cell
Dn, Wn	A/D	Data and Weights precision
DP	A	Number of data bits processed at same time
AWL	A	Activated Wordline per tile
ABL	A/D	Activated Bitline per tile
ADCn, DACn	A	ADC and DAC precision
MPAR	A/D	Filter parallelism
MACPAR	A/D	MAC parallelism
BITPAR	A/D	Data bits parallelism

B. Single-tile

Figure 3 shows a block diagram of the single-tile architecture. It consists of two SRAM blocks (Param SRAM and Data SRAM), the IMC-tile, the control and the digital post-tile computation. The Param SRAM stores the parameters of each layer of the NN as well as the biases of the filters. The

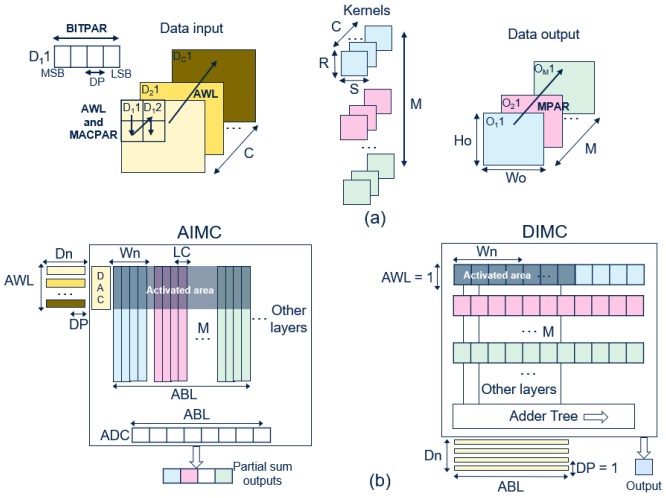

Fig. 2: Parameters for AIMC and DIMC : (a) a convolution layer with parallelism description. R, S, and C represent the 3D filter, and M the filters number. $D_1 1$ and $O_1 1$ respectively represent the first input data and output data pixel. The arrows indicate the directions of parallelism, (b) Difference between AIMC and DIMC with the activated area. depending on the parameters

Data SRAM stores the input image of the NN as well as the intermediate computation data.

The AWL parameter defines the sizing of a memory word and the number of data processed at same time. For convolution and maxpool layers, the AWL parallelism prioritize the channels. For depthwise layers, where there is only one input channel (C) the parallelism is done by rows and the filter is traversed column by column. The number of consecutive words in a channel block is equal to $\frac{MPAR}{AWL}$. This placement allows for easy management of the dataflow and ensures that the necessary data is available at each cycle.

The AWL data read is then placed in registers for processing. Depending on the DP parameter, the row controller, consisting of multiplexers, will place the current DP bits in the correct location in the Ty-sized bus. They will be placed according to the current sequencing of the NN, which is dependent on the NN mapping. The column controller will generate a mask (0/1) of Tx bits to activate the necessary columns according to ABL and the NN mapping. Figure 4 illustrates the row and column controller module. Four inputs data channels (AWL=4) are read from the DATA SRAM. Multiple stages of multiplexers are employed to select the appropriate DP bits from each input data based on the bits to be processed. Subsequently, additional multiplexer stages direct the bits into the row controller registers at the correct positions depending on the IMC tile memory area to be activated. This activation is determined by the inference sequencing. Any remaining registers are set to zero. At the output of the architecture, the number of wires matches the number of rows in the IMC tile (Ty), facilitating direct connection to the WLs or the DAC inputs. Decoding is performed externally to the IMC tile. This method is highly efficient for serial computation (DP=1) or temporal PWM modulation of input data due to the low DAC cost. Requiring one DAC per row, which would be inefficient

for voltage modulation. Similarly, columns are activated in the same manner using column controller registers. The number of registers equal the number of columns in the IMC tile. These registers are set to 1 or 0 based on the filters to be activated according to the NN sequencing. The NN mapping in the IMC-tile is performed by the compiler to optimize IMC-tile space. The layers can be placed on the same column or on different columns.

The digital post-tile component consists of several blocks, including constant shifters and an adder for summing the different bits of the weights placed on the Wn column. This block can be removed if the sum is performed in the IMC-tile, as in [5]. Additionally, there is a shifter block followed by an accumulator to manage the Dn bits, as well as a number of MACs greater than AWL. A quantization block is included to return to a real value between each layer, based on the TF quantization. Finally, there is the activation function component, where only ReLU is supported. The number of digital post-tile and quantization blocks depends on parallelism and will be discussed in the next subsection.

The storing stage block manages the writing of intermediate images to the data SRAM. The maxpool stage block manages the case of maxpool layers where the IMC-tile is bypassed. The registers of the post-tile blocks are reused to perform comparison operations. The architecture supports convolutional, depthwise, dense, and maxpool layers.

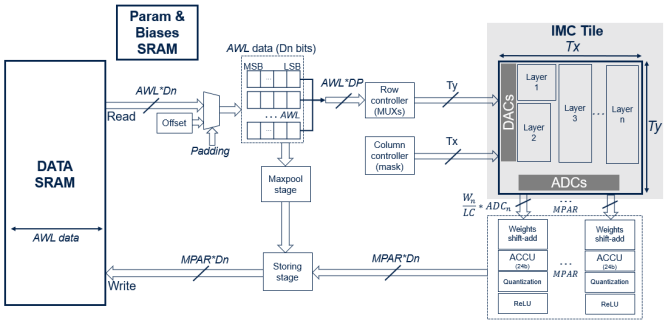

Fig. 3: Block diagram of single-tile architecture

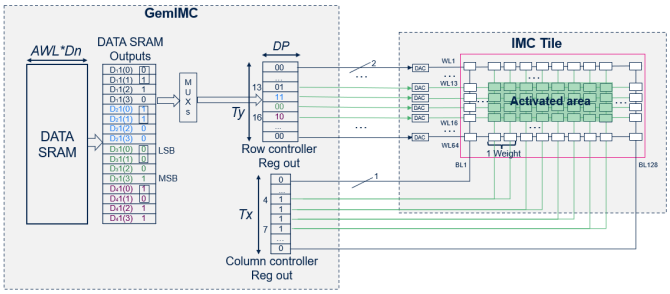

Fig. 4: Operating principle of the IMC Tile control with the row and column controllers registers and the wires directly connected to the WLs and BLs of the IMC Tile

Figure 5 illustrates the scheduling of the architecture, where different functions are pipelined to enable one processing of the IMC-tile per cycle. The beginning of a layer involves reading the parameters into param SRAM to initialize the control registers. The bias registers are then updated in several cycles, depending on the number of filters and param SRAM width. During the last cycle of the bias update, a read of the data SRAM is performed, and the IMC control ports (rows and columns) are updated. The same data is processed for $\frac{D_n}{DP}$ cycles before starting the second wave of data. At the start of a new convolution, the quantization of the outputs of the previous convolution's accumulator is done over several cycles before writing the outputs to memory.

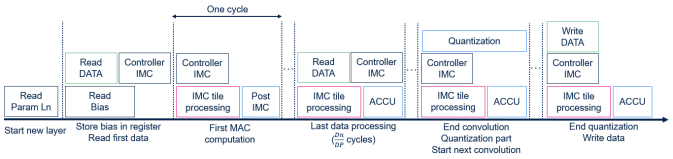

Fig. 5: Scheduling of the architecture for convolution layer starting from the beginning of the layer up to the writing of one wave of results

IV. Architecture Evaluation

The digital HW architecture is evaluated based on the configuration parameters described in Section III and is dependent on the NN. The latency is accurately determined, while the digital HW cost is determined by analyzing the number and precision of the main digital blocks. An extrapolation can be performed to obtain the power and area for a given technology.

A. Latency

The ideal latency for a convolution is defined by Equation 1. It is the ratio of the number of MACs required to perform a convolution and the product of all parallelisms. The BITPAR parallelism is considered to be well-chosen with 100% efficiency, which is lower than $BITPAR_{max}$ (see III.b). For a depthwise layer and maxpool, C is equal to 1 for the number of required MACs. For a dense layer, it is the product of the input and output neurons.

$$Lat_{conv_{ideal}} = \frac{R \cdot S \cdot C \cdot W_o \cdot H_o \cdot M}{MPAR \cdot AWL \cdot MACPAR \cdot} \cdot \lceil \frac{Dn}{DP \cdot BITPAR} \rceil \quad (1)$$

Equation 2 provides the real latency of the architecture based on different parallelisms. The latency is based on the number of cycles of the digital HW part. It is assumed that the next DP bits are provided to the IMC-tile at each cycle, and the actual number of cycles may vary depending on the encoding type, such as PWM or pulse count. The ceil of each parallelism indicates the loss of efficiency, i.e., the times when all parallelisms are not used at 100%. For instance, when C=9 and AWL=8, the equivalent latency is C=18.

$$Lat_{conv_{reel}} = R \cdot S \cdot W_o \cdot H_o \cdot \lceil \frac{C}{AWL} \rceil \cdot \lceil \frac{M}{MPAR} \rceil \cdot \lceil \frac{Dn}{BITPAR} \rceil \quad (2)$$

Equation 3 shows the loss of efficiency, with the ratio of the actual latency of the architecture to the total number of MACs in the layer. The objective is to maximize efficiency for each layer of a given NN, and to find the parallelisms that optimize efficiency with a fixed architecture. A reconfigurable architecture is the best-case scenario to have the adequate parallelisms for each layer, but at the cost of the maximum

digital HW of each parallelism. The reconfigurability aspect is not discussed in this paper.

$$Efficiency_{conv}(\%) = \frac{Lat_{conv_{ideal}}}{Lat_{conv_{reel}}} \cdot 100 \quad (3)$$

B. Digital HW cost

Figure 6 provides a detailed view of the HW implementation. For the post-tile computation phase, the weights shift-add component is responsible to aggregate the results into a single weight value derived from multiple columns. The shifter adjusts the result based on the specific bits being processed within the data input. Finally, the accumulation unit aggregates these results. In the quantization section, the final convolution result is multiplied by the scaling factor over 16 bits, which is executed in four cycles. The row controller section illustrates the hierarchical levels of multiplexers used to initially select the DP bits from among the Dn bits. Subsequently, it addresses the Ty registers appropriately, ensuring that each data point can be directed to any register as required.

Table II provides the operand precision for the input/output modules of post-tile, quantization, and IMC tile control for a single-tile. The number of post-tile and quantization modules is equal to MPAR. The adder weights sub-module's precision is determined by the number of adder inputs (i), each with a different precision. The constant shifter preceding the adder varies for each weight bit position, ranging from MSB to LSB. This sub-module is not present if the operation is performed directly in the IMC-tile (Cf. II).

The register ACCU size of 24 bits for a single-tile was chosen arbitrarily. It is dependent on Wn and Dn: $Reg_{ACCU_{1tile}} = D_n \cdot W_n + 8$.

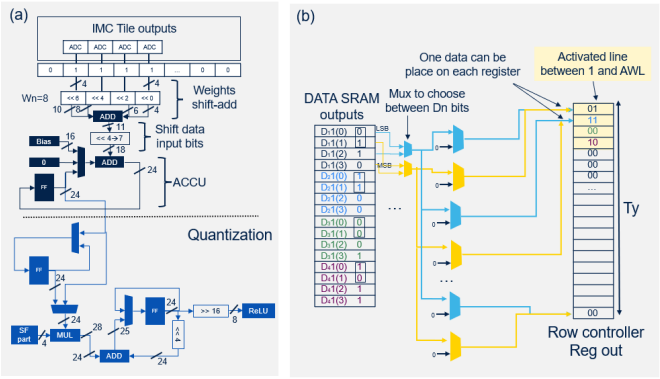

Fig. 6: HW details : (a) Post-tile and quantization part (b) Row controller

V. Methodology and Results

To evaluate the digital HW architecture with an IMC model, a methodology to write the IMC model is provided in Figure 7. The digital HW provides control for the rows and columns to be activated at each cycle. An architectural parameters file in the form of defines is used to configure the environment, and the compiler fills arrays with the mappings of weights on the IMC-tile. The IMC model must include a method for mapping numerical weight values onto the physical cell arrays, as well as parameters such as current, operating voltage, and equations related to uncertainties. The calculation core of the IMC-tile array provides the results of the IMC-tile to the digital HW at each cycle.

In this study, we will exclusively evaluate the digital architecture component to compare it with Gemini, an internal NPU at STMicroelectronics. This evaluation aims to determine the hardware cost of the digital part within an IMC-based architecture. For a fair comparison, the cost of the IMC Tile should also be considered and can be included during the evaluation process. The evaluation was performed on a VGG-like neural network model, detailed in Figure 8. The results presented in Table III are derived from a gate-level simulation based on an inference of the VGG-like model. The equivalent number of PEs corresponds to the maximum number of MAC operations that can be performed in one cycle with the associated hardware. For GemIMC, the architectural parameters were set as follows: AWL=32, MPAR=32, DP=2, LC=2, Tx=128, and Ty=256. The equivalent number of PEs is calculated as $\frac{MPAR \times AWL}{DP}$.

The number of cycles required for inference with a VGG-like model does not follow a straightforward 2:1 ratio between the two architectures. This discrepancy is due to GemIMC's specific efficiency of 75% with this neural network. Layers 1 through 4 lack a sufficient number of channels and filters relative to the parallelization. The area cost is mainly driven by the SRAM DATA cost, which varies based on the targeted application. The standard cell portion of GemIMC is relatively small, being 4.5 times smaller than that of the NPU. The dynamic power cost of the digital part in an IMC-based architecture is notable, being 2.5 times lower than that of the NPU, excluding the IMC tile power consumption.

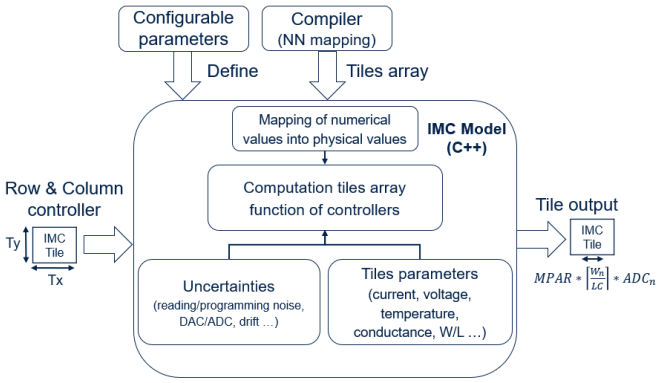

Fig. 7: Block scheme of IMC model

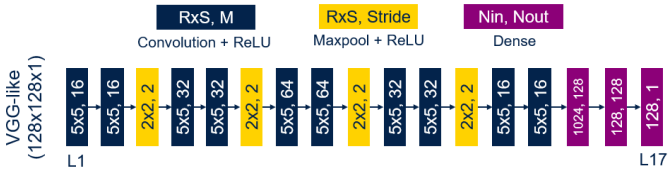

Fig. 8: Details of VGG-like NN model for model evaluation

TABLE II: Digital HW operand precision function of architectural parameters

Module	Sub-module	Operand Precision (bits)	Number
Post-tile	Adder weights	$Add_{Weights_{in(i)}} = ADC_n + W_n - (i \cdot \lceil \frac{W_n}{\lceil \frac{W_n}{LC} \rceil} \rceil)$ with $1 \leq i \leq \lceil \frac{W_n}{LC} \rceil$ $Add_{Weights_{out}} = ADC_n + W_n$	MPAR
	Register ACCU	24	MPAR
	Adder ACCU	$Add_{ACCU_{in1}} = Adder_{Weights_{out}} + D_n$ $Add_{ACCU_{in2}} = 24$ $Add_{ACCU_{out}} = 24$	MPAR
	Register bias	16	MPAR
Quantization	Multiplier SF	$In = 24 \times 4$ and $Out = 28$	MPAR
	Register SF	16	MPAR
	Register	24	2×MPAR
IMC tile controller	Mux row	2->1	AWL×Ty×DP
	Mux column	2->1	Tx
	Register row	DP	Ty
	Register column	1	Tx

TABLE III: Comparative results between Gemini NPU and GemIMC architecture

18nm 0.68V 25° @200MHz	PEs equivalent	SRAMs (KB)	Cycles (VGG-like)	Area (mm²)	Dynamic power (µW/MHz)
Gemini NPU	128	Weights: 128 Data: 128	4M	Logic : 0.1 SRAM : 0.55	Logic : 100 SRAM : 37
GemIMC (without IMC Tile)	256	Data: 128	3M	Logic : 0.023 SRAM : 0.265	Logic : 33 SRAM : 19.5

VI. Conclusions

In this paper, a configurable digital HW architecture for technology-agnostic IMC-based neural network inference at the edge has been proposed. This configurability takes into account the IMC-tile parameters and provides a HW architecture that can adapt to any IMC technology and the physics of the MAC operation, it can be adapted to any type of NN and can incorporate parallelism at different levels. In addition to providing an RTL that can be directly integrated, it enables an evaluation of the entire system in terms of latency, power, and area by taking as input a variabilty-aware IMC model (to guarantee a thorowgh evaluation of AIMC solutions). The paper also presents a methodology to evaluate the resulting architecture and highlights the impact of different parallelisms on latency and digital HW cost.

References

[1] Weiwei Shan et al. "a 510-nw wake-up keyword-spotting chip using serial-fft-based mfcc and binarized depthwise separable cnn in 28-nm cmos". *IEEE Journal of Solid-State Circuits*, 56(1):151–164, 2021.

[2] Qibang Zang et al. 4b/4b/8b precision charge-domain 8t-sram based cim for cnn processing. *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5, 2023.

[3] Kaili Zhang et al. A novel 9t1c-sram compute-in-memory macro with count-less pulse-width modulation input and adc-less charge-integration-count output. *IEEE Transactions on Circuits and Systems I*, 2023.

[4] Heng Zhang et al. Ssm-cim: An efficient cim macro featuring single-step multi-bit mac computation for cnn edge inference. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–12, 2023.

[5] Yiming Chen et al. Samba: Single-adc multi-bit accumulation compute-in-memory using nonlinearity- compensated fully parallel analog adder tree. *IEEE Transactions on Circuits and Systems I:*, 70(7), 2023.

[6] Riduan Khaddam-Aljameh et al. Hermes-core—a 1.59-tops/mm2 pcm on 14-nm cmos in-memory compute core using 300-ps/lsb linearized cco-based adcs. *IEEE Journal of Solid-State Circuits*, 57(4), 2022.

[7] Win-San Khwa et al. A 40-nm, 2m-cell, 8b-precision, hybrid slc-mlc pcm computing-in-memory macro with 20.5 - 65.0tops/w for tiny-al edge devices. *2022 IEEE International Solid- State Circuits Conference*.

[8] Hongwu Jiang, Shanshi Huang, Wantong Li, and Shimeng Yu. Enna: An efficient neural network accelerator design based on adc-free compute-in-memory subarrays. *IEEE Tran. on Circuits and Systems I*, 70(1).

[9] Jong-Hyeok Yoon et al. A 40-nm 118.44-tops/w voltage-sensing compute-in-memory rram macro with write verification and multi-bit encoding. *IEEE Journal of Solid-State Circuits*, 57(3):845–857, 2022.

[10] Dingbang Liu et al. An energy-efficient mixed-bit cnn accelerator with column parallel readout for reram-based in-memory computing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(4):821–834, 2022.

[11] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(12):3067–3080, 2018.

[12] Xiaochen Peng et al. Dnn+neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 32.5.1–32.5.4, 2019.

[13] Yimin Wang, Zhuo Zou, and Lirong Zheng. Design framework for sram-based computing-in-memory edge cnn accelerators. *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021.

[14] Ali Oudrhiri et al. Performance modeling and estimation of a configurable output stationary neural network accelerator. *hal-04168803*, 2023.

[15] Nathan Bain et al. Quantization modes for neural network inference: Asic implementation trade-offs. *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 01–08, 2023.