Horizon 2020 Program (2014-2020)

FET-Open Novel ideas for radically new technologies

FETOPEN-01-2018-2019-2020

Architecting More than Moore – Wireless Plasticity for Massive Heterogeneous Computer Architectures[1]

D5.4: Release of the open-source simulator with benchmark architectures

WP5 - Multi-scale Simulation

Contractual Date of Delivery	31/12/2022
Actual Date of Delivery	21/12/2022
Deliverable Security Class	Public
Editor	Giovanni Ansaloni (EPFL)
Contributors	EPFL (Leader), IBM
Quality Assurance	Sergi Abadal (UPC) Irem Boybat (IBM)

[1] This project is supported by the European Commission under the Horizon 2020 Program with Grant agreement no: 863337

Document Revisions & Quality Assurance

Deliverable Number	D5.4
Deliverable Responsible	EPFL
Work Package	WP5
Main Editor	Giovanni Ansaloni

Internal Reviewers

1. Irem Boybat (IBM)
2. Sergi Abadal (UPC)

Revisions

Version	Date	By	Overview
0.1	21/11/2022	Giovanni Ansaloni	Document created
0.2	02/12/2022	Giovanni Ansaloni, Joshua Klein, Rafael Medina	Content review
0.3	20/12/2022	Irem Boybat, Giovanni Ansaloni	Internal review by Irem Boybat
1.0	21/12/2022	Sergi Abadal, Giovanni Ansaloni	Internal review by Sergi Abadal

Executive Summary

This deliverable details the open sourcing of the code base developed in WP5. The resulting infrastructure was employed for performing system-level simulations validating of the technologies at the focus of the WiPLASH project, namely Analog In-Memory Computing (AIMC) cores and in-package wireless transmission. It is provided as self-contained forks of the gem5-X simulator, embedding custom extensions emulating AIMCs and wireless links.

The deliverable provides a report detailing the repositories structure (including a discussion on the employed license). Moreover, it describes how repositories can be used to perform system-level explorations of the capabilities of WiPLASH technologies, and how these can be extended and/or further refined, both in future releases by the consortium partners beyond the timeframe of WiPLASH, and by external users.

Repositories and technical documentation, including README files and example applications, are available at github.com/gem5-X/ALPINE and github.com/gem5-X/On-Chip-Wireless. As stated in the Description of the Action (DoA) , we provide them as open source and free-of-charge according to the BSD 3-clause license. We ask users to reference our relevant works[2,3] when using them in future endeavors.

[2] Klein, Joshua Alexander Harrison, et al. “ALPINE: Analog In-Memory Acceleration with Tight Processor Integration for Deep Learning.” IEEE Transactions on Computers, 2023.

[3] Medina Morillas, Rafael, et al. "System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms." Asia and South Pacific Design Automation Conference (ASPDAC), 2023.

Abbreviations and Acronyms

AIMC	Analog In-Memory Computing
CPU	Central Processing Unit
ISA	Instruction Set Architecture
MAC	Medium Access Protocol
MLP	Multi-Layer Perceptron
MVM	Matrix-Vector Multiplication
SoC	System on Chip
WP	Work Package

The *WiPLASH* consortium is composed by:

UPC	Coordinator	Spain
IBM	Beneficiary	Switzerland
UNIBO	Beneficiary	Italy
EPFL	Beneficiary	Switzerland
AMO	Beneficiary	Germany
UoS	Beneficiary	Germany
RWTH	Beneficiary	Germany

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Deliverable content and rationale

The focus of Work Package 5 (WP5) is the development of a novel system simulation framework, instrumental for assessing the impact of the innovations pursued by the WiPLASH project from a system- and application-wide perspective. In particular, the framework allows to quantify the benefits deriving from the use of nanoantennae for on-chip and in-package communication, and that of Analog In-Memory Computing (AIMC) tiles for speeding up computation.

As such, WP5 is characterized by two main end-points. The first end-point is the insights gained by the system explorations, detailed in D5.1/D5.2/D5.3 and in the scientific publications mentioned in Section 1.3. The second is the system simulation environments built to perform such explorations, described in this document.

Our efforts have resulted, for the duration of the project, in several design iterations, which allowed to refine the simulation infrastructure and to deepen explorations. Intermediate steps in this process have been reported in previous deliverables of WP5:

- D5.1 first introduced our approach and the envisioned strategy to embody it.
- D5.2 presented preliminary implementation of AIMC and wireless system simulations, and presented initial explorations.
- D5.3 introduced more general and flexible implementations in our platforms, including the possibility to instantiate wireless communication links between any architectural component in a simulated system. It also provides extensive experimental evidence on the benefit of the WiPLASH innovations, targeting a wide set of applications.

In this light, D5.4 reports the open-sourcing of the developed framework, and describe the related documentation which will allow third parties to assess the benefit of AIMC acceleration and in-package wireless communication, replicating and building on our findings.

Full system simulations in WiPLASH's WP5 are enabled by extending the gem5-X framework[4] (itself based on the widely adopted gem5 simulator[5,6]) with dedicated custom extensions. In this way, consortium partners are able to leverage the capabilities of gem5-X, such as support for the Linux operating systems, support for multiple cores and complex memory hierarchies, hardware validation of computing and storage components. At the same, the modular and expandable structure of gem5-X allowed us to develop dedicated extensions for the emulation of AIMC tiles and in-package wireless links, incorporating the hardware characterization undergone in WPs 1-4.

[4] Qureshi, Yasir Mahmood, et al. "Gem5-x: A many-core heterogeneous simulation platform for architectural exploration and optimization." ACM Transactions on Architecture and Code Optimization (TACO) 18.4 (2021): 1-27.
[5] Binkert, Nathan, et al. "The gem5 simulator." ACM SIGARCH computer architecture news 39.2 (2011): 1-7.
[6] Lowe-Power, Jason, et al. "The gem5 simulator: Version 20.0+." arXiv preprint arXiv:2007.03152 (2020).

1.2 Organization of open-source repositories

The resulting frameworks have been open-sourced in the form of github repositories, released as part of the gem5-X organization (github.com/gem5-X), as shown in Figure 1.

- The "On-chip-Wireless" repository, available at github.com/gem5-X/On-Chip-Wireless refers to the extension implementing nanoantennae, and allows to specify systems having in-package wireless links.
- Similarly, the "ALPINE" repository, available at github.com/gem5-X/ALPINE contains all material related to the emulation of systems embedding AIMC as tightly coupled accelerators.

Repositories are self-contained. In particular, each of them embeds a full installation of gem5-X version 2.1. This approach future-proofs the released code base, as the proper functionality of the developed extensions in not dependent on future changes in gem5 and/or gem5-X. Given the rapid release cycle of gem5/gem5-X, such aspect is particularly important for the long-term stability of the code release.

The repositories released in the context of the WiPLASH project co-exist with other efforts in the gem5-X organization. We see this as an added value, as gem5-X is an evolving ecosystem of system simulation solutions for assessing novel software and hardware technologies. In this context, we are confident this ecosystem will be very relevant for industry and academia alike, in no small part thanks to the solutions developed in WiPLASH's WP5.

Repositories present the following content:

- Installation files for the gem5-X system simulator.
- Dedicated extensions (implemented as architectural modules) pertaining to in-package wireless communication or analog in-memory computing, respectively.
- Benchmark applications, showcasing examples of the use of the above-mentioned extensions.
- Software support libraries to encapsulate low-level routines and in-line assembly code, easing the development of new applications by future users.
- Technical manuals illustrating how to set up and execute full system simulation. The manual borrows from the gem5-X documentation, but provides additional information regarding each extension.

1.3 Licensing

All material is provided under the BSD-3-Clause license[7]. This very permissive license allows redistribution and use of the content of the repositories in source and binary forms, with or without modification, as long as the following conditions are met:

- Redistributions of source code or binaries must retain the copyright notice

[7] The 3-Clause BSD license: opensource.org/licenses/BSD-3-Clause.

- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Such license is also used for the gem5 and gem5-X simulation environments, minimizing the adoption effort related to licensing for potential users.

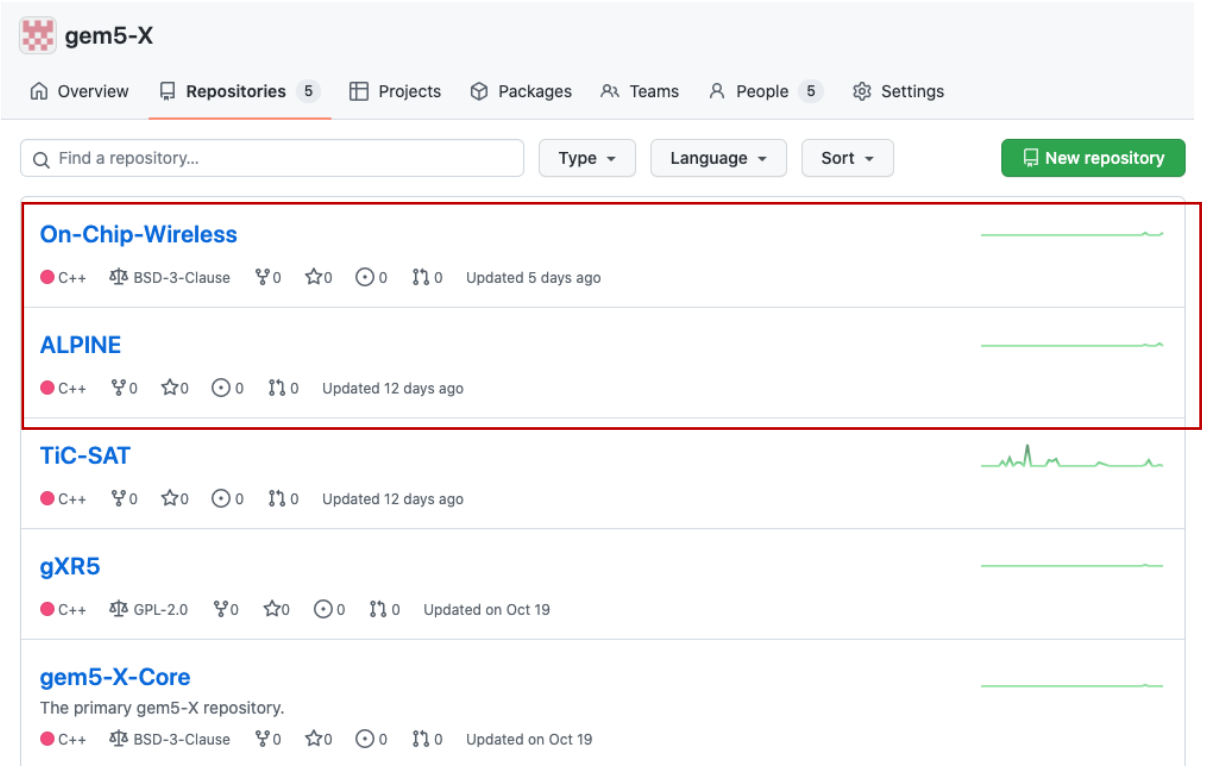

Figure 1: Gem5-X github organization, containing the "On-Chip-Wireless" and "ALPINE" repositories. Available at: github.com/gem5-X.

In addition, we ask users to reference our relevant publications when extending our works:

- For the "**ALPINE**" repository:
  Klein, Joshua Alexander Harrison, et al. "ALPINE: Analog In-Memory Acceleration with Tight Processor Integration for Deep Learning." IEEE Transactions on Computers, 2023.
- For the "**On-Chip-Wireless**" repository:
  Medina Morillas, Rafael, et al. "System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms." Asia and South Pacific Design Automation Conference (ASPDAC), 2023.

2 Setting up a gem5-X environment

The following steps are common for setting up either repository (or when performing a set-up of the main gem5-X environment). They are reported here, only once, for brevity.

2.1 Framework set-up

First, the simulator source files have to be cloned, using

```
git clone https://github.com/gem5-X/ALPINE.git
```

or

```
git clone https://github.com/gem5-X/On-Chip-Wireless.git
```

for AIMC or wireless extensions, respectively.

Then, components of the simulated software stack required for full system simulation (bootloader, kernel binary, disk image containing Ubuntu Linux) are obtained by registering for gem5-X at esl.epfl.ch/gem5-x to obtain a download link. The corresponding tar file should be decompressed as follows:

```
tar -zxvf fullsystemimages.tar.gz
```

Furthermore, the software components (*full system images*) should be linked to the simulator by setting the appropriate environment variables:

```
cd <path_to_gem5-X>
```

```
./apply-patch.sh <PATH_TO_FULL_SYSTEM_IMAGES>
```

If running on an Ubuntu-based host system, the following prerequisites need to be installed before generating the device tree binaries.

```
sudo apt-get install gcc-arm-linux-gnueabihf gcc-aarch64-linux-gnu
```

```
sudo apt-get install device-tree-compiler
```

To finish the set-up, the device tree must be compiled as follows:

```
cd <path_to_gem5-X>
```

```
make -C system/arm/dt
```

In the case of the AIMC extension (ALPINE repository), some configuration parameters pertaining to the accelerator must be defined before compiling the framework. They are detailed in Section 3.1.

2.2 Performing a full system simulation

Here we report the basic steps to define a gem5-X system and initiate a simulation.

An ARM gem5 binary is built as follows:

```
cd <path_to_gem5-X>/
```

```
scons build/ARM/gem5.fast
```

The build command, above, relies on python 2.7.5 and SCons 3.0.0. We provide a docker file to install these dependencies. Details on how to use it retrieve and use it are provided in the frameworks technical manuals, available in each repository.

Once the process above finishes, a simulation can be launched by defining a system and the disk images. An example of last step is provided below.

```
cd <path_to_gem5-X>/

./build/ARM/gem5.fast\
    --remote-gdb-port=0 \
    -d /path/to/your/output/directory \
    configs/example/fs.py \
    --cpu-clock=1GHz \
    --kernel=vmlinux \
    --machine-type=VExpress_GEM5_V1 \
    --dtb-file=<full_path_to_gem5-X>/system/arm/dt/
                armv8_gem5_v1_1cpu.dtb \
    -n 1 \
    --disk-image=gem5_ubuntu16.img \
    --caches \
    --l2cache \
    --l1i_size=32kB \
    --l1d_size=32kB \
    --l2_size=1MB \
    --l2_assoc=2 \
    --mem-type=DDR4_2400_4x16 \
    --mem-ranks=4 \
    --mem-size=4GB \
    --sys-clock=1600MHz
```

The command above starts the simulation of a system with one core running at 1GHz, and the specified size/characteristics of L1, L2 and main memory, mounting the provided Ubuntu disk image.

Users can interact with the simulation by connecting via telnet:

```
telnet localhost 3456
```

In this way, users can launch applications and profile their execution. Applications should be compatible with the ARM ISA. Usually this is achieved by cross-compiling them on the host machine.

The technical manuals provide further details on these set-up steps. These includes a description on how to use checkpointing to decrease simulation time, and how to employ profiling to gather run-time information of the simulated system and application.

3 ALPINE: Analog In-Memory Computing Tile Model and Interfaces

In this chapter, we describe how to retrieve, utilize, and program applications for the ALPINE Analog In-Memory Computing extension of gem5-X. This extension was the employed for the exploration at the core of our authored paper titled “ALPINE: Analog In-Memory Acceleration with Tight Processor Integration for Deep Learning”, by Klein et al., published in IEEE Transactions on Computers, 2023. An example of such exploration, related to a two-layers multi-layer perceptron, is presented in Figure 2.

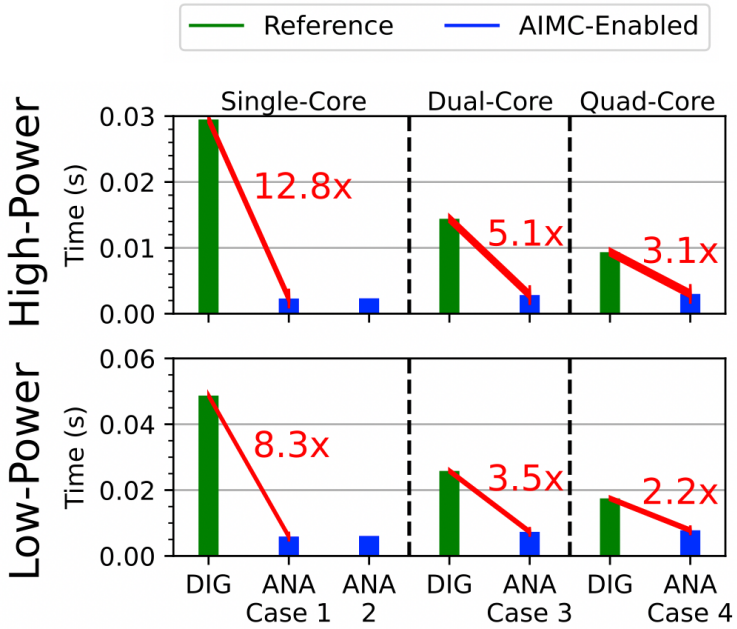

Figure 2: Aggregate run-time for running both digital reference MLP and its AIMC-enabled counterpart, for two systems with different capabilities. “DIG” refers to the digital reference application while “ANA” refers to the analogue AIMC-tile enabled application. Case numbers corresponds to different architectural arrangements, as specified in Klein et al., 2023.

3.1 Configuration Parameters

The guide for setting up, running, and utilizing the ALPINE extension in gem5-X-ALPINE largely follows that of gem5-X itself, as described in Section 2. Nonetheless, some parameters/configuration options lack scripting support, and should therefore be done before the gem5 binary is compiled with the scons script, as reported in Section 2.2. They are described in the rest of this section.

3.1.1. Operation Latency of Custom Instructions

The latency of processing on an AIMC tile is described in the file github.com/gem5-X/ALPINE/blob/master/gem5-X-ALPINE/src/cpu/minor/MinorCPU.py, line 142, in terms of CPU clock cycles. In the example below, assuming a CPU clock

frequency of 2GHz[8], processing on the AIMC would therefore take 100ns, since opLat equals 200 and 200 / 2GHz = 100ns.

```
class MinorDefaultCusProcessFU(MinorFU):
    opClasses = minorMakeOpClassSet(['CusAluProcess'])
    timings = [MinorFUTiming(description="CusProcess",
        srcRegsRelativeLats=[2])]
    opLat = 200
```

3.1.2. *Configuring Tile Generation*

In the default ALPINE configuration, AIMC tiles are generated using the following code in the file github.com/gem5-X/ALPINE/blob/master/gem5-X-ALPINE/src/dev/arm/aimc_cluster.cc (line 35):

```
AIMCCluster::AIMCCluster(
     const AIMCClusterParams * p) :
     BasicPioDevice(p,  p->pio_size),
     system(dynamic_cast<ArmSystem *>(p->system))
{
     warn("AIMC tile instantiated.");

    this->pioAddr = p->pio_addr;
    this->pioSize = p->pio_size;

     for (auto cpu : p->cpus) {
        cpus.push_back(cpu);
        tiles.push_back(new AIMCTile());
    }
```

The *for* loop in the code above creates one new AIMC tile object for every CPU on the system. These are then placed in a vector and accessed by custom instructions, as defined below in this section. Indeed, in our implementation, custom instructions access the vector of AIMCs using the CPU number as an index, as specified in the github.com/gem5-X/ALPINE/blob/master/gem5-X-ALPINE/src/arch/arm/isa/insts/data64.isa file.

The AIMC instantiation (src/dev/arm/aimc_cluster.cc) and the instruction set extensions definition (src/arch/arm/isa/insts/data64.isa) should therefore be jointly changed to modify the integration strategy of AIMC tiles, for example by only providing few cores with an AIMC accelerator, or instantiating multiple tiles for each core.

[8] The CPU clock frequency is itself a user-specified parameter, as illustrated in the gem5-X technical manual.

3.2 ALPINE AIMC Tile Model Implementation

The AIMC tile model in gem5-X and ALPINE is comprised of four main components: custom instruction definitions, the custom instruction to PIO device interface, the AIMC tile wrapper, and finally, the AIMC tile model itself.

3.2.1. ALPINE ISA Extension

Five custom instructions are defined for the tightly-coupled interface. Their format is shown in Table 1. Note that "X" in a table field refers to a "don't care" value. As mentioned above, the functionality of instructions is implemented in src/arch/arm/isa/insts/data64.isa.

Table 1: ALPINE custom instructions

gem5 Mnemonic	**OpCode**	**Rm**	**R/W**	**Ra**	**Rn**	**Rd**
cmprocess	0x00C	X	0x0	X	X	X
cmqueue	0x10C	packed input	0x1	X	X	X
cmdequeue	0x10C	X	0x0	X	X	Packed Output
cmparamread	0x20C	Tile Column	0x1	X	Tile Row	Parameter
cmparamwrite	0x20C	Tile Column	0x0	Parameter	Tile Row	Success

All instructions are 32-bit three-register R-type instructions with 11-bit op-codes, 5-bit indexing for registers, and 1-bit R/W. A description of each of the instructions' behavior is described below:

- **cmparamwrite**: Write the 8-bit parameter held in register Ra in the AIMC tile weight at index Rn, Rm.
- **cmparamread**: Read and return the 8-bit parameter into register Rd from the tile weight at index Rn, Rm.
- **cmqueue**: Write 32 bits (corresponding to four 8-bits values) to the input register of the AIMC. No explicit index in the input register is required, as this is automatically auto-incremented when queuing values.
- **cmdequeue**: Read in register Rd 32-bits from the AIMC output register, corresponding to four 8-bits values. Again, the index in the output register is auto-incremented upon a dequeue command.
- **cmprocess**: Perform the Matrix-Vector Multiplication operation by multiplying-and-accumulating each crossbar matrix column with the contents of the input memory and storing the truncated 8-bit output in the AIMC tile output memory. This operation also refreshes (sets to 0) the AIMC tile's input memory contents.

All instructions are implemented as wrappers to methods implemented in the AIMC Tile Wrapper Object. Therefore, the general implementation of each instruction simply gets a pointer to the AIMC Tile Wrapper, formats the instruction arguments, and then performs the associated AIMC Tile Wrapper method. We use the thread ID to access an AIMC tile, as we assume one private to each CPU core. As mentioned above, other integration schemes can be realized by modifying the tile generation mechanism (Section 3.1.2).

3.2.2. ISA-to-Device Interface Connection

In order to provide the tightly-coupled interface to the AIMC Tile Wrapper Object, the gem5 system object (defined in src/arch/arm/system.hh) is connected to both the wrapper object and ISA templates. The system object is included in the ISA templates by including the aforementioned *system.hh* file as well as /src/dev/arm/aimc_cluster.hh into /src/arch/arm/isa/includes.isa. The system object then includes a pointer to the AIMC wrapper object that allows the custom instructions to query the wrapper object.

3.2.3. AIMC Tile Wrapper Object and PIO Device

To configure, generate, and access the individual AIMC tile models, the AIMC Tile Wrapper object, (src/dev/arm/aimc_cluster.hh), is responsible for the placement of and delegation of tasks to the AIMC tiles. It is configured as a gem5 peripheral input/output (PIO) device and sits atop the ARM Realview Platform (src/dev/arm/Realview.py):

```
class AIMCCluster(BasicPioDevice):
    type = 'AIMCCluster'
    cxx_header = "dev/arm/aimc_cluster.hh"
    pio_addr = Param.Addr(0x10020000, "Address for AIMC core
                                        access.")
    pio_size = Param.Int32(0x1000, "Size of AIMC memory-
                                     mapped address range.")
    cpus = VectorParam.BaseCPU("CPUs/hardware threads
                                attached to this device.")
```

As required by PIO devices, the AIMC cluster is placed in the address range of [0x10020000 : 0x10021000].

The "cpus" parameter is used to generate AIMC tiles, because the current implementation of ALPINE generates one AIMC Tile per CPU on the simulated SoC. As detailed above, other strategies can be implemented by modifying the source code in src/dev/arm/aimc_cluster.cc.

3.2.4. AIMC Tile Model

The individual AIMC tiles modules are generated as structs, as shown below, in the file src/dev/arm/aimc_cluster.hh:

```
struct AIMCTile {
    // Constructor.
    AIMCTile() :
    crossbarHeight(2000),
    crossbarWidth(2000),
    crossbar(new int8_t[crossbarWidth*crossbarHeight]),
    inputMemory(new int8_t[crossbarHeight]),
    outputMemory(new int8_t[crossbarWidth]),
    inputMemoryCounter(0),
    outputMemoryCounter(0),
    vectorization(4)
    {
        for (int i = 0; i < crossbarHeight; i++) {
            inputMemory[i] = 0;
            for (int j = 0; j < crossbarWidth; j++) {
                crossbar[(i * crossbarWidth) + j] = 0;
            }
        }

        for (int i = 0; i < crossbarWidth; i++) {
            outputMemory[i] = 0;
        }
    }

    const int crossbarHeight;   // Height of input memory.
    const int crossbarWidth;    // Width of output memory.
    int8_t * crossbar;          // Parameters crossbar.
    int8_t * inputMemory;       // Input memory (pre-DAC).
    int8_t * outputMemory;      // Output memory (post-ADC).
    int inputMemoryCounter;     // Index into input memory.
    int outputMemoryCounter;    // Index into output memory.
    const int vectorization;    // How many values do we
                                //     queue/dequeue?
};
```

The struct holds the parameter corresponding to the emulated tile physical dimensions (height and width i.e., number of rows and columns). Furthermore, it stores the state of the AIMC accelerator: the values of its stored weights and that of the input and output registers. It is also in charge of auto-incrementing counters in the input and output registers upon a queue or dequeue operation.

3.3 AIMClib

AIMClib is a header-only C/C++ software library which facilitates the interface to the AIMC cluster from an application perspective. It is held in a folder separate from ALPINE in the master/aimclib repository branch. AIMClib encapsulates the intrinsics necessary to execute the custom instructions, as well as basic methods for queuing and dequeuing larger data structures such as arrays and vectors. Furthermore, it includes a C++ "checker" module that is used to debug AIMClib implementations by emulating gem5/ALPINE behavior in software (i.e., outside of an ALPINE system).

To use AIMClib in a C program, simply include aimc.hh in the source code of the desired program to make the function prototypes provided by AIMClib available. To use the checker module specifically, the option "-DUSE_CHECKER" should be included in the gcc/g++ compilation. The most relevant function prototypes are included below:

```
// Write a matrix/vector to the AIMC tile crossbar.
inline void mapMatrix(int aimc_x, int aimc_y,
                      int height, int width,
                      int8_t ** m);
inline void mapMatrix(int aimc_x, int aimc_y,
                      int height, int width,
                      int8_t * m);

// Queue/dequeue to/from AIMC tile input/output memories.
inline void queueVector(int size, int8_t * v);
inline void dequeueVector(int size, int8_t * v);

// Perform Matrix Vector Multiplication in the AIMC tile.
inline void aimcProcess();
```

In addition to the base prototypes listed above, these methods are also templated, in case casting from a higher-precision data type to *int8_t* is required.

3.4 ALPINE Sample Application

We provide the application code to perform inference on a 1024x1024 Perceptron. It is written in C++ code using AIMCLib calls to interface the accelerator module. This application can be compiled and run on a gem5-X/Alpine instance as specified in Section 2.2. The application code is available in the gem5-X-ALPINE repository as aimclib/example.cc, and reported below.

```
int main(int argc, char * argv[])
{
    // Test bench parameters.
    int n_x         = 1024; // MLP input/output dimensions.
    int T_x         = 10; // Number of inferences.

    // Set up and initialize vectors/matrices.
    int8_t ** input     = new int8_t*[T_x];
    int8_t * W1         = new int8_t[n_x*n_x];
    int8_t ** output    = new int8_t*[T_x];
    for (int i = 0; i < T_x; i++) {
        input[i] = new int8_t[n_x];
        output[i] = new int8_t[n_x];
        for (int j = 0; j < n_x; j++) {
            input[i][j] = (int8_t)rand();
            output[i][j] = (int8_t)rand();
        }
    }
    for (int i = 0; i < n_x*n_x; i++)
        W1[i][j] = (int8_t)rand();

    // Map weights to AIMC tile.
    mapMatrix(0, 0, n_x, n_x, W1);

    // Do inference.
    for (int i = 1; i < T_x; i++)
    {
        // Queue input for next inference in first layer.
        queueVector(n_x, input[i]);

        // Do MVM.
        aimcProcess();

        // Dequeue output from AIMC tile MVM.
        dequeueVector(n_x, output[i]);
    }

    // Cleanup and return.
    for (int i = 0; i < T_x; i++) {
        delete[] input[i];
        delete[] output[i];
    }
    delete[] input;
    delete[] output;
    delete[] W1;

    return 0;
}
```

A dedicated compilation script (aimclib/build_example.sh) is also provided. The application can be compiled with or without the -DUSE_CHECKER option. Using the option, compiled binaries will use stand-in behavioral models for performing AIMClib calls. Otherwise, binaries will target the gem5-X + ALPINE system simulation.

The gem5-X + ALPINE simulation output reports, among other statistics, the number of executions on AIMC tiles (calls to the **cmprocess** instruction), as well as the amount of queueing and dequeuing operations. This data, in conjunction with the hardware characterization performed in WP4 by the project partner IBM, can be exploited to perform an energy/performance analysis of the application execution on AIMC-accelerated systems. The WiPLASH partners IBM and EPFL indeed followed this approach, considering both a low-power and a high-performance architecture, in the *Klein et al.* IEEE Transactions on Computers article mentioned in Section 1.3.

4 On-chip Wireless Module and System Integration

Gem5-X-On-Chip-Wireless is a dedicated fork of Gem5-X including custom modules for emulating systems with in-package wireless links. This extension was the basis for the conference paper “System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms”, presented at the ASP-DAC 2023 conference.

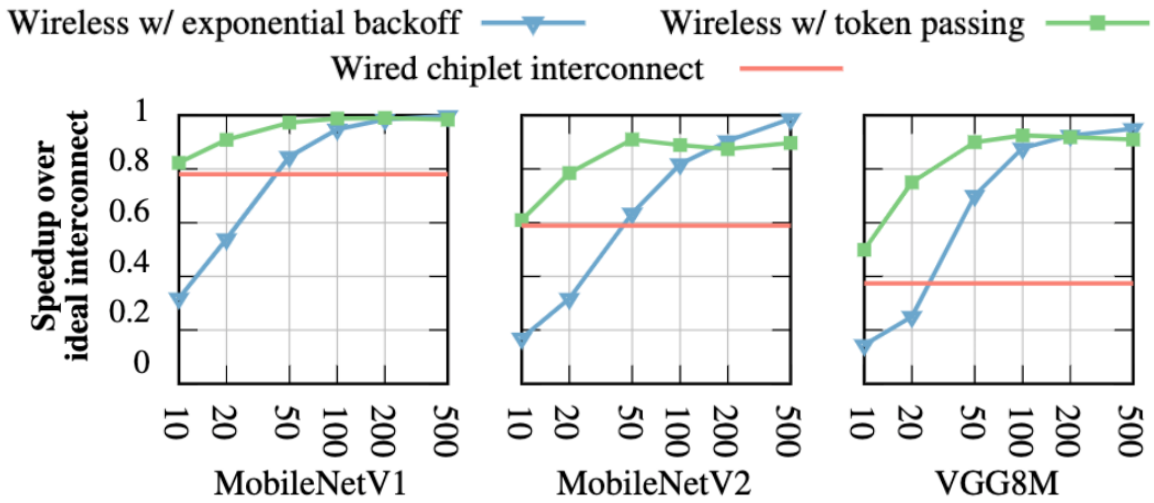

Figure 3: Speed-up over ideal interconnect of token passing and exponential backoff MAC protocols, for different link bandwidths and three representative applications. Performance over UCIe inter-chiplet wired link is also presented, for reference.

The extension allows the emulation of systems comprising wireless links among different architectural components (processors, private or shared caches, main memory, etc.). Links can be parametrized in terms of latency, bandwidth and employed Medium Access Control (MAC) in order to gather performance results such as the ones shown in Figure 3.

Two MAC protocols are implemented at present:

- **Token passing (**Figure 4**)**. When using this protocol, a virtual token is exchanged among components connected to a wireless link. Only the component (e.g. processor core) which holds the token can initiate a data transaction, such as a read or write operation. The holder releases the token immediately if it has no outstanding transaction, otherwise the release is performed upon the completion of the first transaction in the queue.
- **Exponential backoff (**Figure 5**).** In this protocol, all components interfacing the wireless link can initiate a transaction at any time. If a collision is detected, all involved transactions are aborted. Each of them is further tried at a random time inside a time interval. The size of the interval is exponentially increased when collisions are detected, and decreased upon a successful transaction. The initial retry window (slot) size, as well as the maximum value of the exponent employed to increase it, can be parametrically determined in our implementation.

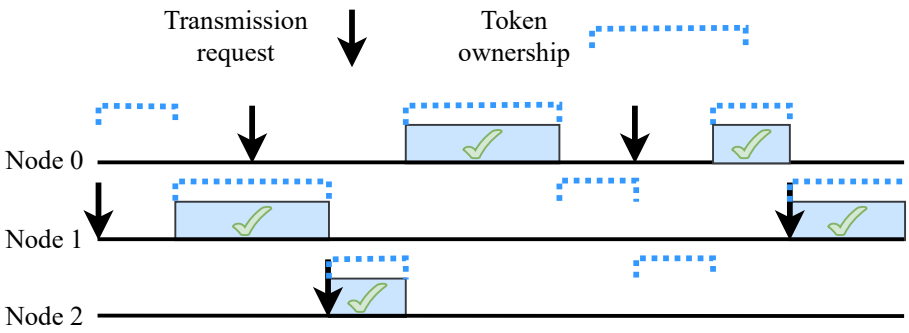

Figure 4: Timing diagram of the token passing protocol.

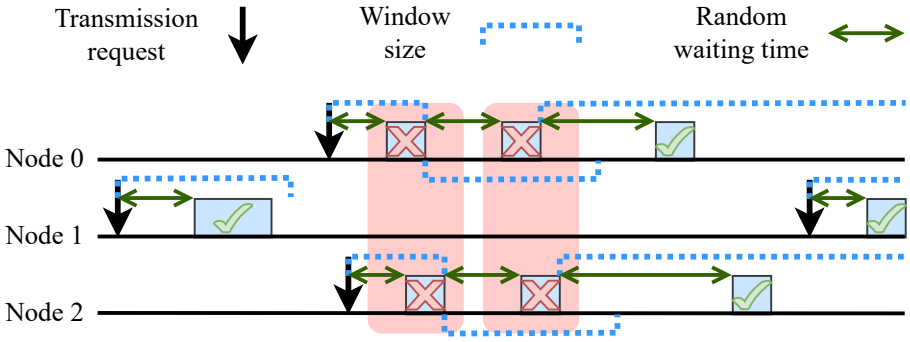

Figure 5: Timing diagram showing the behavior of the exponential backoff protocol under successful transmissions (green ticks) and collisions (red crosses).

The wireless transceiver module is built upon the standard component interfacing mechanism ("crossbar"[9]) of gem5, augmented with additional features for collision detection and retransmission of packets according to a MAC protocol. This approach allows to freely instantiate wireless links across the system hierarchy of an emulated system, without requiring any modification to other architectural components or in software applications.

4.1 Running gem5-X Full System Mode with wireless extensions

Gem5-X-On-Chip-Wireless can be cloned from the repository and built in the following the usual procedure for gem5-X, as mentioned in Section 2. After the environment is set up, wireless-capable systems can be specified either from a terminal command line or from a configuration file.

4.1.1. Defining wireless-capable systems from the command line

An example of using the former approach is provided below.

```
./build/ARM/gem5.fast \
--remote-gdb-port=0 \
-d /path/to/your/output/directory \
```

[9] The term "crossbar" refers in this Section to the standard component used in gem5 and gem5-X to implement interfaces among storage and computing elements in systems. Its meaning is therefore distinct with respect to the one used in Section 3, where it instead referred to an AIMC structure.

```
configs/example/fs.py \
--cpu-clock=1GHz \
--kernel=vmlinux \
--machine-type=VExpress_GEM5_V1 \
--dtb-file=<path_to_gem5-
X>/system/arm/dt/armv8_gem5_v1_<NUM_CORES>cpu.dtb \
-n <NUM_OF_CORES> \
--disk-image=gem5_ubuntu16.img \
--caches \
--l2cache \
--l1i_size=32kB \
--l1d_size=32kB \
--l2_size=1MB \
--l2_assoc=2 \
--mem-type=DDR4_2400_4x16 \
--mem-ranks=4 \
--mem-size=4GB \
--sys-clock=1600MHz \
--membus-wireless \
--wireless-bandwidth=12.5GB/s \
--mac-protocol=exp_backoff
```

The command above generates a system with <NUM_CORES> number of cores, L1 and L2 caches of the defined sizes and a CPU clock of 1GHz. The system will mount a disk containing Ubuntu Linux and boot from it.

Options specifically related to Gem5-X-On-Chip-Wireless are in the last three lines of the command. Using them, a wireless memory bus is instantiated, connecting main memory with the L2 cache. The wireless link has a bandwidth of 12.5GB per second, and employs an exponential backoff protocol to arbitrate bus collisions.

Command line options related to in-package wireless links are

- --**l2bus-wireless**: instantiates a wireless link connecting L1 and L2 caches.
- **--membus-wireless**: instantiates a wireless link connecting L2 caches and main memories.
- --**wireless-bandwidth=<BANDWIDTH>**: set the bandwidth of the wireless link.
- **--mac-protocol=<exp backoff / token pass>**: selects the MAC protocol, either as exponential backoff or as token passing, as described in D5.3.
- **--retry-slot-size=<SIZE>**: sets the size of the retry slot when using the exponential backoff protocol, specified as a multiple of the time required to transmit a byte according to the available bandwidth.
- **--backoff-ceil=<MAX EXPONENT>**: sets the upper limit to of the size of the retransmission window in the exponential backoff protocol.

Defining systems via command line offers a fast avenue towards exploring the performance of in-package wireless. Nonetheless, it also limits flexibility in the system generation. Indeed, only systems with a two-level cache hierarchy are supported, and only either L1/L2 and L2/main-memory wireless links can be instantiated.

4.1.2. Defining wireless-capable systems with configuration files

In a system configuration file, a wireless link can be instantiated similarly to a standard gem5 crossbar, adding the parameters specific to wireless transmission (bandwidth, employed MAC protocol etc.). An example related to link using an exponential backoff protocol is reported below.

```
system.wireless_link = WirelessXBar(
                    clk_domain = system.clk_domain,
                    bandwidth = options.wireless_bandwidth,
                    mac_protocol = options.mac_protocol,
                    retry_slot_size =
options.retry_slot_size,
                    backoff_ceil = options.backoff_ceil)
```

Architectural components can then be connected to the ports of the wireless link as with usual gem5-X crossbars, freely composing a system as desired. In the example below, wireless ports are connected to L1 and L2 caches:

```
system.l1cache.master = system.wireless_link.slave
system.l2cache.slave = system.wireless_link.master
```

An example system configuration file is provided in the gem5-X-On-Chip-Wireless repository, here: github.com/gem5-X/On-Chip-Wireless/blob/master/gem5-X-wireless/configs/common/CacheConfig_wirelessExample.py.

4.2 On-Chip-Wireless module implementation files

The main files containing the wireless module descriptions are in the directory src/mem/. A brief description is provided in the following:

- `WirelessXBar.py` contains a description of the crossbar and of its parameters.
- `wireless\xbar.hh` is a header file defining variables and functions prototypes of the wireless module.
- `wireless\xbar.cc` describes the functionality of the gem5-X wireless module. It is built upon the standard crossbar implementation in gem-5. In addition, it supports collision detection and retransmission based on token passing and exponential backoff protocols, as well as a parametric link bandwidth.

4.3 Example application

Included in the gem5-X-On-Chip-Wireless repository is the STREAM benchmark suite[10]: benchmarks/Stream/Stream.c .The suite is provided with no adaptation in the C code, as the on-chip wireless components are transparent to software. OpenMP pragmas are instead updated in order to distribute the execution on different cores, as exemplified in the code snipped below for the Copy() benchmark:

[10] STREAM benchmark suite: www.cs.virginia.edu/stream/.

```
void tuned_STREAM_Copy()
{
        ssize_t j;
#pragma omp parallel for proc_bind(close)
        for (j=0; j<STREAM_ARRAY_SIZE; j++)
            c[j] = a[j];
}
```

5 Conclusions and Perspectives

The deliverable describes the structure, content and license of the repository the consortium employed to release the system level simulators developed during the course of the project.

Two repositories have been published on the github platform. The full system simulators therein are forks of the gem5-X environment. As such, they are part of the same organization of github, constituting important elements in the gem5-X ecosystem. The repositories are able to emulate the full hardware/software stack of systems embedding the core innovations of WiPLASH, namely in-package wireless communication and accelerated computation based on Analog In-memory Computing (AIMC).

These tools were the basis for the research contributions achieved in WiPLASH - WP5, which demonstrated the system-level benefit of nanoantennae and AIMC tiles for next-generation computing systems. We now release them under the very permissive BSD-3-Clause license, which will allow academic and industrial institutions to build upon our development efforts and our findings.

The present document is the last deliverable of WP5. Work package activities for the remainder of the project will proceed along two directions. First, we will deepen our explorations, further refining our frameworks to gain novel insights. Second, we will react feedbacks from the system simulation communities after the code release.