Gem5-X + TiC-SAT
Full System Manual

[*]EMBEDDED SYSTEMS LABORATORY,
SWISS FEDERAL INSTITUTE OF TECHNOLOGY, LAUSANNE (EPFL)

[‡]SCHOOL OF ENGINEERING AND MANAGEMENT VAUD (HEIG-VD),
UNIVERSITY OF APPLIED SCIENCES WESTERN SWITZERLAND (HES-SO)

[**]DEPARTMENT OF COMPUTER ARCHITECTURE,
COMPLUTENSE UNIVERSITY OF MADRID

V2.2-TIC-SAT BY ALIREZA AMIRSHAHI[*], JOSHUA KLEIN[*], AND GIOVANNI ANSALONI[*]

BASED ON V2.0 OF GEM5-X TECHNICAL MANUAL BY YASIR QURESHI[*], WILLIAM SIMON[*], MARINA ZAPATER[*‡], KATZALIN OLCOZ[**], AND DAVID ATIENZA[*]

December 2022

Contents

1 Executive Summary

1.1 Abstract

The gem5 architectural simulator is well established and widely used in both the industry and academia. Based on gem5, we present gem5-X (*"a gem5-based full-system simulator with architectural eXtensions"*), a simulation framework that enables fast profiling and architectural exploration and optimization for system-level architectural innovations. Gem5-X provides out-of-the-box simulation of ARM-based systems with a full Linux stack, along with several architectural extensions like ISA extensions, clustering, heterogeneous many-core simulation, and the HBM2 memory model. Several enhanced features have also been added, like advanced check-pointing, workload automation (WA), and gperf profiler support.

This version of the gem5-X repository, named gem5-X-TiC-SAT, further provides support for a tightly-coupled systolic array accelerator developed in the context of transformers used in the FvllMonti project (https://fvllmonti.eu/).

This technical manual first provides guidelines on how to use various architectural features and support enhancements of gem5-X. More information on downloading and source code for gem5-X can be found at https://esl.epfl.ch/gem5-x. Then, In Section 8, we describe the features specific to the gem5-X-TiC-SAT extension.

1.2 Release Information

Version	Authors	Date	Changes
v2.2-TiC-SAT	Alireza Amirshahi, Joshua Klein, and Giovanni Ansaloni	Dec. 2022	Forked gem5-X manual for the TiC-SAT extension.
v2.2	Joshua Klein, Rafael Medina, Alireza Amirshahi, Marina Zapater, and Giovanni Ansaloni	Oct. 2022	Reformatting and setting up for new extensions.
v2.1	Joshua Klein and Darong Huang	Feb. 2022	Updated version information for gem5-X dependencies on Ubuntu 20.04 LTS, expanded contact info for current maintainers.
v2.0	Yasir Qureshi, William Simon, Marina Zapater, Katzalin Olcoz, and David Atienza	Aug. 2021	Core clustering, heterogeneous cores and SPM support added in Gem5-X.

1.3 Collaboration and Contact Information

The maintainers of this project can be contacted via email at {alireza.amirshahi, joshua.klein, giovanni.ansaloni, david.atienza}@epfl.ch

Because this project's scope is very large, we are always interested in potential collaboration efforts to develop new features and keep gem5-X updated to gem5 master. Please contact one of the aforementioned emails for inquiries, source code, and additional information.

2 Running gem5-X Full System (FS) Mode with ARMv8 and Linux

This chapter describes how to configure and run our ARMv8 64-bit FS simulation in gem5-X.

2.1 Necessary Files

Because our model is run in FS mode with a full Linux environment, we need several major system components. This includes,

- A bootloader
- A kernel binary, e.g., vmlinux
- A disk image
- A device tree binary

All of the aforementioned components must be compatible with the ARMv8.

2.1.1 Full System Files

Once you register for gem5-X at https://esl.epfl.ch/gem5-x, you will receive an email with a link to all the system files, except for the device tree. The file downloaded is named ***full_system_images.tar.gz***. This contains the disk image, bootloader, and kernel binary. Follow the instructions below to set it up.

```
1 tar -zxvf full_system_images.tar.gz
```

The files are as follows:

- Bootloader is under **[path_to_full_system_images]/binaries/**
- Kernels (vmlimux and vmlinux_wa) are at **[path_to_full_system_images]/binaries/**
- Disk image (gem5_ubuntu16.img) can be found at **[path_to_full_system_images]/disks/**

We now need to set up the path to *full_system_images* so that the files under it can be used and recognized by gem5-X during FS simulation.

```
1 cd <path_to_gem5-X>
2 ./apply-patch.sh <PATH_TO_FULL_SYSTEM_IMAGES>
```

Alternatively, you can also do

```
1 export M5_PATH=<PATH_TO_FULL_SYSTEM_IMAGES>
```

The full system files are now set up and ready to be used in FS mode.

2.1.2 Device Tree

The device tree files are under

```
<path_to_gem5-X>/system/arm/dt
```

If running on an Ubuntu-based host system, the following prerequisites must 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 generate the device tree binary files,

```
cd <path_to_gem5-X>
make -C system/arm/dt
```

2.2 Quick-Start Guide

This brief start-up guide will guide you through the basic steps to running your first full system (FS) simulation with gem5-X. This guide assumes you already have the bootloader, device tree, kernel file, and disk image setup as described in the previous sections.

2.2.1 Prerequisites

You will need to set up the gem5-X environment in order to compile and run the gem5-X binary using the SCons (SConstruct) builder. If running on an Ubuntu-based host system, you can use the following command to get all the required libraries. However, there are some known dependency problems on the latest Ubuntu image, i.e., 20.04. If you are running these host systems, we recommend you follow **Section 2.3** to build a docker image to run gem5-X inside.

```
    sudo apt install build-essential git m4 scons zlib1g \
    zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev \
    libgoogle-perftools-dev python-dev python-six python \
    libboost-all-dev swig
```

2.2.2 Building the gem5 Binary

Once the above is done, you will need to build an ARM gem5 binary. You can create multiple builds, including .fast, .opt, and .debug. If you are only concerned about running experiments, it is recommended to only create gem5.fast. However, if you need to debug anything or want to generate traces, you will need to build gem5.opt or gem5.debug. Do this with the following:

```
    cd <path_to_gem5-X>/
    scons build/ARM/gem5.{fast, opt, debug}
```

Additionally, if you would like to speed up the compilation process, you can use the option ”-jN” on the scons build line where N is the number of threads you want to assign for compilation.

2.2.3 Running Your FS Simulation

Once the build process is complete, you can launch your simulation in the following way.

```
cd <path_to_gem5-X>/

./build/ARM/gem5.{fast, opt, debug} \
--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 \
```

At this point, you should be able to connect to your running gem5 instance in another terminal with,

```
telnet localhost 3456
```

Alternatively, you can also build the terminal program provided with gem5-X and use it

```
cd <path_to_gem5-X>/util/term/
make
m5term 127.0.0.1 3456
```

Upon connecting to your gem5 instance, you should be able to access the kernel dmesg, followed finally by a login and a terminal in the gem5-X FS mode.

2.3 Hot-fixes for running gem5-X on Ubuntu 20.04 using Docker

Because of updates to gem5-X dependencies in Ubuntu 20.04, specific dependency versions must be installed. In particular, Python 2.7.5 and SCons 3.0.0 (build version py27h8a56064_0) must be used. These packages can be easily configured via a virtual environment. Here we provided a Dockerfile to create the docker image, containing all of the necessary dependencies to run gem5-X.

```
FROM ubuntu:20.04
SHELL ["/bin/bash", "-c"]
ENV DEBIAN_FRONTEND=noninteractive
```

```
4  RUN echo "deb http://dk.archive.ubuntu.com/ubuntu/ xenial main" \
5    >> /etc/apt/sources.list \
6    && echo \
7    "deb http://dk.archive.ubuntu.com/ubuntu/ xenial universe" \
8    >> /etc/apt/sources.list \
9    && apt -y update \
10   && apt install -y wget \
11   && wget \
12   https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh\
13   && bash Miniconda3-latest-Linux-x86_64.sh -b -p /opt/miniconda3
14 RUN source /opt/miniconda3/bin/activate \
15   && conda init \
16   && conda create --name py275 -c free python=2.7.5 -y \
17   && conda activate py275 \
18   && conda install scons=3.0.0=py27h8a56064_0 -y
19
20 RUN apt-get -y install build-essential gcc-arm-linux-gnueabihf \
21 gcc-aarch64-linux-gnu \
22   device-tree-compiler make git m4 zlib1g \
23   zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev \
24   libgoogle-perftools-dev python-dev \
25   libboost-all-dev swig=3.0.8-0ubuntu3 \
26   && apt-get -y install diod \
27   && apt-get -y install qemu qemu-user qemu-system \
28   qemu-user-static
```

The above docker file should be put inside a newly created folder, e.g., gem5x-docker/Dockerfile. Then you can build the docker image in the terminal:

```
1    cd <path to gem5x-docker/Dockerfile>
2    docker build -t gem5x .
```

Wait until the image is successfully created, then you can run the image by using the following command:

```
1    (sudo) docker run -it gem5x
```

Before you go to Section 2.2.2 to build the gem5, you need to first enable the conda environment:

```
1    source /opt/miniconda3/bin/activate
2    conda activate py275
```

Besides, two additional hotfixes are required in gem5-X's SConstruct file (**gem5-X/SConstruct**): First, due to deprecated features in gcc 9.3.0+, lines 365 - 370 should be commented out. Second, the GNU assembler version in the Sconstruct file needs to be updated, so change [-1] to [3] in line 435.

Now you can go back to **Section 2.2.2** to build the gem5 binary. Please contact the maintainers if issues continue to arise.

3 Support Enhancements of Gem5-X

In this chapter we will look into the following support enhancements we have added in gem5-X:

- Enhanced checkpointing
- gperf profiler
- File sharing between gem5-X and host system using 9P over Virtio
- Modifying disk image using QEMU

3.1 Enhanced Checkpointing

The boot process during the FS simulation in gem5-X automatically takes a checkpoint when the boot and login is complete and we get the terminal. Since the boot is in SimpleAtomic CPU model, the timing information is not there in the simulation. We can now switch to an accurate in-order or our-of-order (OoO) CPU model with all the timing information.

If your simulation is still running after the boot, you can exit it using the following command in the connected terminal,

```
m5 exit
```

Now we can use the checkpoint, that was automatically taken after boot and login, and switch to an accurate CPU model.

```
./build/ARM/gem5.{fast, opt, debug} \
--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 \
-r 1 \
--cpu-type={MinorCPU, DerivO3CPU}
```

The number after *–r* is the checkpoint number. In this case we are resuming from the first checkpoint. The CPU type can be *MinorCPU* for in-order core or *DerivO3CPU* for OoO cores.

Sometimes it is feasible to take a checkpoint using SimpleAtomic CPU model just before your region-of-interest (ROI) and then switch to an accurate in-order or OoO CPU. You can do this in either the command prompt or a script using the following command:

```
m5 checkpoint
```

If you are in a C/C++ program, you can use the following call within the program,

```
system("m5 checkpoint");
```

3.2 Gperf Profiler

Profiling capabilities within FS, by installing the gperf profiler on the disk image. The gperf statistical profiler developed by Google provides profiling capabilities on gem5-X itself with minimal overhead, enabling the identification of application bottlenecks and exploration of the effectiveness of architectural modifications and extensions.

To enable profiling using gperf when running a program, follow the instructions below;

```
LD_PRELOAD=/usr/lib/libprofiler.so.0 CPUPROFILE=<FILE_TO_SAVE_PROFILING>
CPUFREQUENCY=1000 <program>
```

This will launch the program to be profiled with profiling data being saved to file mentioned in *CPUPROFILE* parameter.

To view the data, we first convert it to .pdf file and then write it to the host machine as follows;

```
1 google-pprof --pdf <FILE_WITH_PROFILING_DATA> > <FILENAME>.pdf
2 m5 writefile <FILENAME>.pdf
```

The file *FILENAME.pdf* will now be available to be viewed in the host system under path passed to –d parameter when launching gem5-X simulation

```
-d /path/to/your/output/directory
```

3.3 9P over Virtio

We utilize the 9P protocol developed by Bell Labsover a virtio device driver to allow fast modification of files without modifying the root file system in gem5-X. While this feature is available in vanilla gem5, it is not enabled by default and has no kernel support. Both of these features are provided in gem5-X. Once Linux is booted, a folder on the host machine can be mounted within gem5 to access files on the host system. Without 9P mounting, every time a program is modified, we need to reload the disk image required for FS simulation and reboot Linux. In gem5, this process can take up to 20-30 minutes, a bottleneck that gem5-X eliminates. To use 9p over Virtio, follow the instructions below:

- First we need to install DIOD

  ```
  sudo apt-get install diod
  ```

- After installation, check where DIOD is installed by typing "which diod". This path should be updated in the file src/dev/virtio/VirtIO9P.py at line 62. Then re-compile gem5-X using scons command as usual.

```
cd <path_to_gem5-X>/
scons build/ARM/gem5.{fast, opt, debug}
```

- Use kernel "vmlinux_wa", during the gem5 simulation. This file is provided with gem5-X under full_system_images/binaries
- Use the following additional parameter when launching the simulation

  ```
  workload-automation-vio=<FULL_PATH_TO_SHARED_FOLDER_ON_HOST_SYSTEM>
  ```

- Once the system is booted, run the following in gem5 terminal

  ```
  mount.sh <FULL_PATH_TO_SHARED_FOLDER_ON_HOST_SYSTEM>
  ```

- Now any file under the "SHARED_FOLDER_ON_HOST_SYSTEM" appears in the /mnt directory in gem5 simulation.

3.4 Modifying disk image using QEMU

To run experiments an application and benchmarks in gem5-X, they need to be on the disk image. To to this we need to update and modify the disk image with the applications.

QEMU is used to modify the disk image. If running on an Ubuntu-based host system , the following prerequisites need to be installed.

```
sudo apt-get install qemu qemu-user qemu-system qemu-user-static
```

To mount the image

```
cd <PATH_TO_FULL_SYSTEM_IMAGES>/disks/
mkdir local_mnt
sudo mount -o loop,offset=$((2048*512)) gem5_ubuntu16.img local_mnt
sudo mount -o bind /proc local_mnt/proc
sudo mount -o bind /dev local_mnt/dev
sudo mount -o bind /dev/pts local_mnt/dev/pts
sudo mount -o bind /sys local_mnt/sys
```

Now we *chroot* into the image emulating using QEMU

```
cd local_mnt/
sudo chroot ./
```

At this point we are in the ARMv8 disk image and can now compile or download applications within the image. Since it is a Ubuntu 16.04 image, you can run the following first, before installing any new packages on it,

```
apt-get update
```

When the disk image has been update with the applications or benchmarks, we can exit it and unmount the image.

```
exit
cd ..
sudo umount local_mnt/proc
sudo umount local_mnt/dev/pts
```

```
5 sudo umount local_mnt/dev
6 sudo umount local_mnt/sys
7 sudo umount local_mnt
```

The modified image is now ready to be used for gem5-X simulation with new applications or benchmarks

4 High Bandwidth Memory v2 (HBM2)

High Bandwidth Memeory (HBM) is based on 3D stacked DRAM banks made possible due to Through Silicon Vias (TSVs) achieving a high bandwidth of up to 307.2 GB/s. To implement the functional behavior of the HBM2 memory model in gem5-X, we extend the DRAM controller model of gem5 according to the architectural details of HBM2. To have 8-channels with memory interleaving, we initialized 8 DRAM controllers, each 128 bits wide. We connect all 8 DRAM controllers to a 1024-bit wide system bus, that connects to the cache hierarchy.

To use 8-channel HBM2 in gem5-X full system simulation, with appropriate bus widths throughout the system all the way to the caches, use the following command:

```
cd <path_to_gem5-X>/

./build/ARM/gem5.{fast, opt, debug} \
--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 \
--l2bus-width=128 \
--membus-width=128 \
--mem-type=HBM2_2000_4H_1x128 \
--mem-ranks=1 \
--mem-channels=8 \
--mem-size=4GB \
--sys-clock=1600MHz \
```

No separate software support is required to use HBM2 in FS mode, and hence we are able to boot the Ubuntu Linux distribution using HBM2.

The HBM2 memory model can be found in the following file

```
<full_path_to_gem5-X>/src/mem/DRAMCtrl.py
```

5 Core Clustering

Core clustering enables group of compute cores to have their own shared cache, which can be last level cache (LLC), separate from other cores in the system. This reduces the shared resources between different compute clusters in the system to just cross bar interconnect and memory. In addition, clustering is also used when different type of cores are used in system. Same core types are clustered together with their own LLCs. This enables to have a heterogeneous system.

Cluster in now supported in gem5-X. To have different core clusters in gem5-X, use the following command:

```
./build/ARM/gem5.{fast, opt, debug} \
--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=<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  \
--l2_cluster_size=<NUM_OF_CORE_PER_CLUSTER> \
--mem-type=DDR4_2400_4x16 \
--mem-channels=4 \
--mem-ranks=4 \
--mem-size=4GB \
--sys-clock=1600MHz
```

This command will simulate a system with core clusters. Each cluster will have number of cores defined in *–l2_cluster_size* parameter. The number of cores defined by *-n* parameter should be divisible by the *–l2_cluster_size*. Dividing *n* by *l2_cluster_size*, gives the number of clusters in the system. Each cluster will have its own L2 (LLC) cache.

6 Heterogeneous Cores

Heterogeneity enables different workloads with vartying performance and energy constraints to be allocated to different core types in the system. Gem5-X supports both in-order and OoO cores in the same system. Different core types are distributed into different clusters.

To use heterogeneity in gem5-X, first the system is launched to boot up the linux to reach the region-of-interest (ROI), with the following command:

```
./build/ARM/gem5.{fast, opt, debug} \
--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=<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 \
--l2_cluster_size=<NUM_OF_CORE_PER_CLUSTER> \
--cluster_size_1=4 \
--mem-type=DDR4_2400_4x16 \
--mem-channels=4 \
--mem-ranks=4 \
--mem-size=4GB \
--sys-clock=1600MHz
```

This command will simulate a system with core clusters. The parameter *–cluster_size_1* defines the size of the 1st cluster of type 1. This should be the same as *–l2_cluster_size*. All the cores in the remaining clusters will be of type 2. For instance, if number of cores is defined to be 16, and both *–l2_cluster_size* and *–cluster_size_1* are set to 4, this implies to have 4 clusters in the system, each with 4 cores. The first cluster will have cores of type 1 and the remaining three clusters will have cores of types 2.

Once the ROI is reached, take a checkpoint using ”m5 checkpoint” command. Then one can resume from the checkpoint with the desired core types for each cluster. For the above code type 1 cores are set to be in-order and type 2 to be OoO, as in the following command:

```
./build/ARM/gem5.{fast, opt, debug} \
--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=<path_to_gem5-X>/system/arm/dt/armv8_gem5_v1_<NUM_CORES>cpu.dtb \
```

```
 9 -n <NUM_OF_CORES> \
10 --disk-image=gem5_ubuntu16.img  \
11 --caches  \
12 --l2cache  \
13 --l1i_size=32kB \
14 --l1d_size=32kB  \
15 --l2_size=1MB  \
16 --l2_assoc=2  \
17 --l2_cluster_size=<NUM_OF_CORE_PER_CLUSTER> \
18 --cluster_size_1=4 \
19 --mem-type=DDR4_2400_4x16 \
20 --mem-channels=4 \
21 --mem-ranks=4 \
22 --mem-size=4GB \
23 --sys-clock=1600MHz \
24 -r 1 \
25 --cpu-type=MinorCPU \
26 --cpu-type_2=DerivO3CPU \
```

7 Scratchpad Memory (SPM)

Scratchpad Memories (SPMs) are software programmable memories at the same level as L1 cache, but controlled by the user. Gem5-X supports SPMs, which are both local and shared between two consecutive cores.

To use SPMs gem5-X, the following command can be used:

```
./build/ARM/gem5.{fast, opt, debug} \
--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=<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 \
--spm \
--d_spm_size=128kB
```

The *–spm* commands enabes SPM in gem5-Xand *–d_spm_size* defines the SPM size, which is set to 128KB in the above example. The SPMs can be accessed by two consecutive cores. For instance, SPM0 is accessible by core0 and core1, SPM1 by core1 and core2, SPM2 by core2 and core3 and so on.

Since this is a FS mode of gem5-X, to use SPM, they need to be mapped using mmapp, as in the following code:

```
void * spm_mem_alloc (uint64_t mem_size, uint64_t mem_address)
{

        uint64_t alloc_mem_size, page_mask, page_size;
        void *mem_pointer;
        void *virt_addr;

        page_size = sysconf(_SC_PAGESIZE);
        alloc_mem_size = (((mem_size / page_size) + 1) * page_size);
        page_mask = (page_size - 1);

        int mem_dev = open("/dev/mem", O_RDWR | O_SYNC);
        if(mem_dev == -1)
```

```
14          {
15                  perror("Cannot open /dev/mem \n");
16                  //return -1;
17
18          }
19
20          mem_pointer = mmap(NULL,
21                          alloc_mem_size,
22                          PROT_READ | PROT_WRITE,
23                          MAP_SHARED,
24                          mem_dev,
25                          (mem_address & ~page_mask)
26          );
27
28          if(mem_pointer == MAP_FAILED)
29          {
30              perror("Cannot MAP \n");
31              //return -1;
32          }
33
34          printf("Memory Mapped \n");
35          virt_addr = (mem_pointer + (mem_address & page_mask));
36
37
38      return virt_addr;
39  }
```

The above core snippet returns a virtual pointer in SPM in FS mode. The parameter *uint64_t mem_size* is used to define the size of memory allocated within SPM. The parameter *uint64_t mem_address* defines the memory address of the SPM in physical memory space. So for SPM0 this should be at an offset after the main memory and I/O devices in gem5-X. So for instance of the main memory size is 4GB, the offset for SPM0 should be 4GB+2GB(I/O devices memory space), i.e. 6GB=6442450944. SPM1 should be at an offset defined by main-memory size + I/O devices + SPM0_size.

8 TiC-SAT

The tightly-coupled accelerator is enabled by interfacing a systolic array to the system components. gem5-X-TiC-SAT supports the modeling of this accelerator. The behavior of a systolic array with different latencies, kernel sizes, and cache sizes can be explored. This extension was the basis for the conference paper " TiC-SAT: Tightly-coupled Systolic Accelerator for Transformers" in ASP-DAC 2023[1].

8.1 Running gem5-X Full System Mode with ARMv8, Linux and systolic array accelerator

Table 1 reports the compatibility of gem5-X-TiC-SAT with respect to other gem5 extensions in gem5-X. No guarantees of compatibility with any present or future gem5-X version should be assumed beyond the ones provided in this table.

Table 1: gem5-X-TiC-SAT Compatibility Chart

Extension	Section	Compatible with On-chip wireless?	Notes
Support Enhancements	3	**Yes**	
HBM2	4	**Yes**	
Core Clustering	5	**Untested**	
Heterogeneous Cores	6	**Untested**	
SPM	7	**Untested**	

gem5-X-TiC-SAT can be cloned from the associated repository via the following command:

```
1 git clone https://github.com/gem5-X/TiC-SAT.git
```

After the environment is set up, TiC-SAT-capable systems can be launched from a terminal command line.

8.2 Configuration Parameters

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. They are described in the rest of this section.

8.2.1 Operation Latency of Custom Instructions

In gem5, the time an instruction takes to execute is determined by its operation latency within the CPU model's functional unit multiplied by the CPU core frequency (when not accounting for blocking operations). The operation latency for the custom instructions used by TiC-SAT is defined as **opLat** in src/cpu/minor/MinorCPU.py, lines 134, 141, 147:

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

[1] https://infoscience.epfl.ch/record/298067

```
5       opLat = 1
6
7   class MinorDefaultCusQueueFU(MinorFU):
8       opClasses = minorMakeOpClassSet(['CusAluQueue'])
9       timings = [MinorFUTiming(description="CusQueue",
10          srcRegsRelativeLats=[2])]
11      opLat = 1
12
13  class MinorDefaultCusParamWriteFU(MinorFU):
14      opClasses = minorMakeOpClassSet(['CusAluParamWrite'])
15      timings = [MinorFUTiming(description="CusParamWrite",
16          srcRegsRelativeLats=[2])]
17      opLat = 1
```

8.2.2 Configuring systolic array size

In the default TiC-SAT configuration, the systolic array size is defined in src/dev/arm/systolic_m2m.hh (line 43):

```
1 #define KERNEL_DIM 8
2 #define W_DATA 4
3 #define MAX_COL 2
```

In this example, the systolic array size (kernel size) is assigned to 8. This value creates an SA accelerator of size 8*8. Depending on the application and the area/energy limitation, one can assign other values for the kernel size. Note that the parameter of MAX_COL should have the value of $\frac{\text{KERNEL\_DIM}}{\text{W\_DATA}}$.

8.2.3 Configuring systolic array size

In the default TiC-SAT configuration, it is assumed that all the data are represented in 8 bits. To change this default, you can modify src/dev/arm/systolic_m2m.hh and src/dev/arm/systolic_m2m.cc to assign another bit-width. The first parameter to be changed is W_DATA. This parameter shows how many values we can fit in a 32-bit bus width. Note that the parameter of MAX_COL should have the value of $\frac{\text{KERNEL\_DIM}}{\text{W\_DATA}}$. Furthermore, all the parameters and arrays with type **int8_t** and **uint8_t** should be changed to **int(x)_t** and **uint(x)_t**, respectively, where (x) shows the new bit-width. Finally, the shifting index in src/dev/arm/systolic_m2m.cc (lines 54, 62, 71, 84, 115, 122) should be adjusted by the new values. For instance, to migrate from 8-bit values to 16-bits, line 54 will be changed as follows:

```
1 auto currVal = (int16_t)((val >> (16 * (W_DATA -i -1))) & 0xffff);
```