Lab 2: Simulating an X86 Architecture in gem5

EEL 5734 – Fall 2022

Introduction:

In order to use gem5 to simulate a system, we have to:

- Define an architecture of that system using a configuration script.
- Then run (i.e., simulate) a program (binary executable) on that system. In general, this process can be used to evaluate (e.g., performance) of different configurations of a system for a given set of applications (i.e., benchmark suite).

In this lab, you will:

- Follow a gem5.com tutorial to create a gem5 configuration script to define a basic X86 architecture (see Figure 1) and run a given X86 binary code ("hello world") on it. (Part 1)
- Modify the source c code of the "hello world" program; and learn how to compile source code into binary code and use it in a gem5 simulation. (Part2)
- Learn about gem5 statistics and output (Part 3)

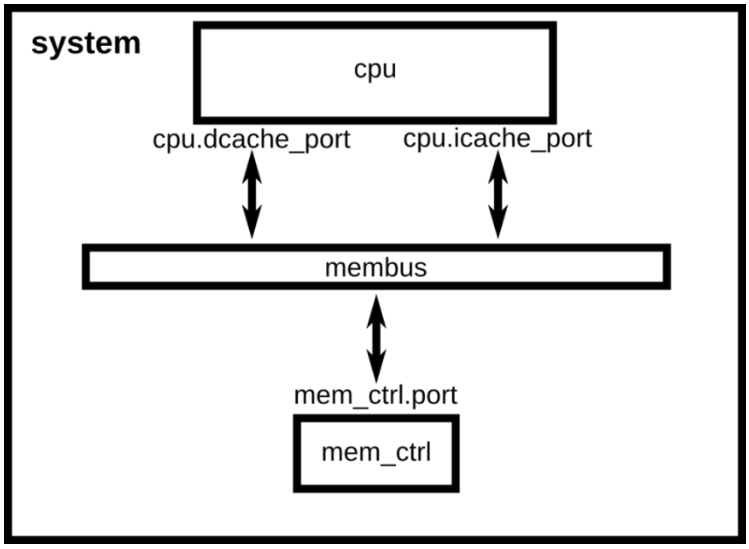

Figure 1. A basic X86 architecture

Part 1 – Creating a basic gem5 configuration script for a "hello world" simulation

In Part 1, you will follow the tutorial ("Creating a simple configuration script") from the gem5.com website to define a basic X86 architecture (Figure 1) and run a given X86 binary code on it.

https://www.gem5.org/documentation/learning_gem5/part1/simple_config/

Helpful notes:

- The bulk of the tutorial is to give you the necessary information to create the simple.py (Python) script on a Ubuntu terminal. On the Ubuntu terminal, you will have to create a configuration file named **simple.py** and enter all the command from the tutorial into it.
- I suggest that you copy/paste the commands (from the tutorial) into a document first (e.g., Word, Wordpad). **Carefully verify** the finished file against the tutorial commands (this will

save you a lot of debugging headache later).

- After verification, copy/paste into the **simple.py** file on the Ubuntu terminal. Alternatively, you can enter in in the gem5 "folder explorer".
- Note that you are directed to use the "touch" command to create **simple.py**. You should use whatever editor that you feel comfortable (e.g., vi).
- In the tutorial, there is an "**An aside on gem5 ports**" section, explaining their use of "port abstraction. It gives examples of:

```
system.cpu.icache_port = system.l1_cache.cpu_side
system.cpu.icache_port = system.membus.cpu_side_ports
```

Note that these two statements are NOT a part of the simple.py script that we will use in Lab 2. So don't include the above statements in your script.

Part 1(a) – Complete the above tutorial and capture a screenshot of the last screen of outputs, from your Ubuntu terminal screen.

Part 1(b) – Change the CPU clock frequency from 1 GHz to **2 GHz**; and then to **4 GHz** (capture two more screenshots).

Part 1(c): Keep the clock at **1 GHz;** change the memory configuration from DDR3_1600_8X8 to:

- **DDR4_2400_8X8,** should be faster the above DDR3 memory
- **DDR3_2133_8x8**, which models DDR3 with a faster clock.
- **HBM_1000_4H_1x64**, which models High Bandwidth Memory, used in GPUs and network devices.
- (3 more screenshots)

Part 1(d): Keep the clock at **1 GHz** and the memory configuration at DDR3_1600_8X8, change the CPU from TimingSimpleCPU() to **O3CPU().**

- (1 more screenshot)

Note for Part 1(d): It was stated at the end of the tutorial, *"Additionally, you can change the CPU model to `MinorCPU` to model an in-order CPU, or `DerivO3CPU` to model an out-of-order CPU. However, note that `DerivO3CPU` currently does not work with simple.py, because `DerivO3CPU` requires a system with separate instruction and data caches (`DerivO3CPU` does work with the configuration in the next section). Also note that `MinorCPU` does not work with X86 as of gem5 version 22.0.0."*

However, the O3CPU model does work with simply.py. That's why you are doing it for Part 1(d).

Part 2 – Modify hello.c source code, compile it into a binary that can be run in gem5

The source code hello.c can be found at: ~/gem5/tests/test-progs/hello/src. You are to do the following:

- Copy the hello.c file into one of "your" directories (e.g., ~/gem5/configs/tutorial/part1/my_hello).
- Modify your hello.c to print "Your name says hello!" (e.g., Herman Lam says hello!).

- Follow the instructions in the Appendix at the end of this lab to compile your hello.c file.
- Put the executable (e.g., hello) in "your" directory.
- Modify simple.py to execute the binary (with all the original clock frequency and models)
- (1 more screenshot)

Part 3 – Understanding gem5 statistics and output

Go to this page (https://www.gem5.org/documentation/learning_gem5/part1/gem5_stats/) and to gain an understanding about gem5 statistics and output.

After you have completed Part 2, run simple.py again, after which these files can be found in **m5out directory:** config.ini, config.json, and stats.txt. Answer the following questions:

1. This statement, `system.clk_domain.clock = '1GHz'`, is specified in simple.py. Which statement(s) **in the config.ini file** contain the corresponding information?
   Answer:

2. This statement, `system.cpu = TimingSimpleCPU()`, is specified in simple.py. Which statement(s) **in the config.ini file** contain the corresponding information?
   Answer:

3. The following statements are founded in the **config.ini file**, defining the "children" SimObjects for the system SimObject (i.e., defining the component of this system).
   ```
   [system]
   type=System
   children=clk_domain cpu dvfs_handler mem_ctrl membus workload
   ```
   Which of these SimObject components are defined in the simple.py configuration script? (Note: the rest of the "children" SimObjects are defaults).
   Answer:

4. After you executed the simply.py script, some of the summary statistics are shown in the terminal:
   (a) `"Global frequency set at 1000000000000 ticks per second"`
   In the stats.txt file, what line number is this statistics displayed?
   Answer:
   (b) `"Exiting @ tick 454646000 because exiting with last active thread context"`
   In the stats.txt file, list all the line numbers in which this tick number is displayed, not just the summary lines (4 and 5).
   Answer:

SUBMISSION INSTRUCTIONS

Turn in two files: one pdf and one zip file.

The pdf file should have the following naming convention:
LastNameFirstNameLab2.pdf ex: LamHermanLab2.pdf

The **pdf file** contains the following:

- **The first page** should contain:
  - A summary table of all your runs (fill in the following table)

Part#: Name	CPU model	CPU clock freq	Memory model	Exit tick #
Part1(a): 1GHz	TimingSimpleCPU	1 GHz	DDR3_1600_8X8	454541000
Part1(b): 2GHz	TimingSimpleCPU			
Part1(b): 4GHz	TimingSimpleCPU			
Part1(c): DDR4	TimingSimpleCPU			
Part1(c): DDR3	TimingSimpleCPU			
Part1(c): HBM	TimingSimpleCPU			
Part1(d): O3CPU	O3CPU			
Part2: ”MyHello”	TimingSimpleCPU			

  - After the summary table, give one paragraph summarizing your (fairly obvious) observations about the data.
  - After the paragraph, give your answers for the 4 questions for Part 3.

- **Next 8 pages:** each contains a screenshot of the last screen of outputs for the 8 cases of the above table from Parts 1 and 2.

The zip file should have the following naming convention:
LastNameFirstNameLab2.zip ex: LamHermanLab2.zip
There should be 3 files in the zip file:

1. Original simple.py file
2. simpleO3CPU.py (configuration file from Part 1(d))
3. Binary code of your updated hello.c (so I can verify that it indeed works if necessary)

Appendix: How to compile your c code

1. **sudo apt-get install gcc** (to install gcc if not installed already)
2. **gcc -O0 -ggdb3 -std=c99 -static -o binary_name source.c**
   This is to compile your c code (you can add or remove flags as per your requirement, but always include -static)**.**

Example: **gcc -O0 -ggdb3 -std=c99 -static -o helloLam helloLam.c**