A bit of everything else

Some other things about gem5
Cache models
gem5’s cache models

“Classic” caches

One model: Cache
Can be configured as a nested hierarchy
Replacement policies, prefetchers, etc.
Can be “mostly exclusive” or “mostly inclusive”
Fake, magic, coherence protocol

Good for functional simulation and low-fidelity memory systems simulation

Simple port interface, easy to configure in python
gem5’s cache models

Ruby & SLICC
High-fidelity, detailed coherence models
SLICC: A language to define state machines
Ruby: A set of models used in SLICC protocols

Lots of available protocols
MESI_Treee_Level, MESI_Two_Level, MOESI_hammer, and more
See src/learning_gem5/part3 for a “simple” protocol

On-chip network and topology are parameters
SLICC specifies just the protocol
Makes configuring quite complex
“Classic” ports

CPU
CPU
CPU
DMA
Other

RUBY

DRAM Ctrl
DRAM Ctrl
Ruby

On-chip interconnect

L1 Cache controller

L2 Cache controller

Directory controller

DMA Controller

Directory controller

YYY Controller
gem5’s cache models

**CHI** (Based on AMBA 5 CHI specification)

**Ruby/SLICC-based model**
- Machines can be configured as different cache levels
- Configurable like classic, but high-fidelity from Ruby

Expect to start seeing more CacheHierarchies implemented with CHI

Much more info on Ruby and SLICC:
Memory models

Memory controller:

- Controller logic (MemCtrl) separate from timing (Interface)
- Event-driven, high-fidelity, but not “cycle accurate”

Interfaces:

- DDR(3/4), LPDDR(2,3,5), HBM (with pseudo channels), HMC
- NVM (PCM-like)

Can also connect to DRAMSim3 and DRAMSys
CPU models
ISA independence

CPU model is independent of ISA

Any CPU model and work with any ISA (mostly)

Carefully designed API between ISA and CPU model
gem5 ISAs

src/arch/

- amdgpu
- arm
- mips
- power
- riscv
- sparc
- x86

Not all equally well supported. ARM, RISC-V X86 most used/tested.

Each directory contains devices, ISA-specific objects, system interface, ISA definition
ISA definition

`src/arch/<isa>/isa`

A domain-specific language for ISAs

- Written in python (`src/arch/isa_parser.py`)
- Honestly, very confusing, not much documentation
- Output in build/.../generated

Decodes instructions (decoder/*.*isa)

Implements instructions (insts/*.*isa)

This is what is called when an instruction “executes” (we’ll see)

Creates “StaticInst” classes
StaticInst/ExecContext

**StaticInst**: Describes the kind of instruction (isNop(), isInteger(), etc.)

**ExecContext**: Interface for *ISA definition* to interact with *CPU model*

**ThreadContext**: Interface for devices/etc. to interact with architectural state

Provides implementation for execution (parameter: ExecContext)

- **execute**: Modify ExecContext based on instruction
- **initiateAcc**: Send memory reference
- **completeAcc**: Like execute for mem insts
- **advancePC**: ISA-specific
CPU Models

gem5 exposes a flexible CPU interface

**AtomicSimpleCPU**: No timing. Fast-forwarding & cache warming.

**TimingSimpleCPU**: Single-cycle (IPC=1) except for memory ops.

**O3CPU**: Out-of-order model. Highly configurable.

**MinorCPU**: In-order model (not fully tested with x86)

**kvmCPU**: x86 and ARM support for native execution
Memory modes

**Timing**
- Used for simulation
- Calls sendTimingRequest, etc.
- All timing is modeled

**Atomic**
- No timing
- Used for fast-forwarding
- Some structures are warmed up

**Atomic_noncaching**
- Used for KVM CPU
- Directly manipulates the backing memory
GPU and device models

AMD GPU
  Supports GCN and VEGA
  ROCm 4.X
  Full system support

Many devices supported for FS simulation
  Ethernet (and multi-system simulation)
  VNC for graphics
  IDE controllers for disks
  No Mali GPU for ARM
  VirtIO

Most devices are functional-only
Contributing to gem5

Use gerrit for code review

See CONTRIBUTING.md

Everyone is welcome!

Some details from Andreas Sandberg
How to structure your change

• What characterizes a good change?

  Small: Smaller changes are easier to review and understand.
  Well-defined: One commit == logical change
  No unrelated changes: Don’t sneak bug fixes into feature commits
  Descriptive commit message
  Always use your real name and email in the commit meta data

• What characterizes a change that makes reviewers cringe?

  Multiple changes going into the same commit “various bug fixes in Foo”
  Large changes that could have been broken into incremental changes
  Poorly written commit messages
## The structure of a commit message

### Summary:
python: Move native wrappers to the _m5 namespace

### Body:
Swig wrappers for native objects currently share the _m5.internal namespae with Python code. This is undesirable if we ever want to switch from Swig to some other framework for native binding (e.g., PyBind11 or Boost::Python). This changeset moves all of such wrappers to the _m5 namespace, which is now reserved for native code.

### Meta data:
- Change-Id: I2d2bc12dcb05b57b7c5a75f072e08124413d77f3
- Signed-off-by: Andreas Sandberg <andreas.sandberg@arm.com>
- Reviewed-by: Curtis Dunham <curtis.dunham@arm.com>
- Reviewed-by: Jason Lowe-Power <jason@lowepower.com>
Commit message: Summary line

Summary: python: Move native wrappers to the _m5 namespace

- *Short* summary of your change (max 65 characters)
  *Think of it as a subject in an email*
- Should uniquely identify your change
- Typically the first thing a potential reviewer sees
- Sometimes the only information shown about a change

- Keywords used to identify affected components
  *See MAINTAINERS.yaml for details*
Commit message: Body

**Body:**
Swig wrappers for native objects currently share the \_m5\.internal name space with Python code. ...

- Should describe your change in detail – think of it as documentation
  *Reviewers will read this before they see any code*

- Describe *what* the change does and *why*
  *Not necessarily how, that should be clear from the code*

- Describe any implementation trade-offs
- Describe known limitations
- Describe testing done
Commit message: Metadata

Meta data:

- **Change-Id**: Unique ID used by Gerrit to identify the change (generated)
- **Signed-off-by**: It’s complicated...
- **Reviewed-by**: Use this to acknowledge reviewers (generated by Gerrit)
- **Reviewed-on**: Link to review request (generated by Gerrit)
- **Reported-by**: Use this to acknowledge users that report bugs
- **Issue-on**: Use this to reference a Jira issue (https://gem5.atlassian.net/jira/software/c/projects/GEM5/issues/)
- **Tested-by**: Can be used to acknowledge testers
Code submission flow

1. Post change for review
   - Apply stick to reviewer
   - Wait for reviews
   - Reviewers happy?
     - No → Update change
     - Yes → Commit change

2. Commit change → Done
The job of a reviewer

- Evaluate technical aspects
  * Is it doing what it says in the commit message?
  * Is a technically sound implementation?

- Evaluate implementation aspects
  * Is the commit message describing the change?
  * Is it following the style guidelines?

- Legal aspects
  * Patch author’s responsibility, but reviewers should look out for obvious issues.

You are the reviewers!
Caveats

gem5 is a tool, not a panacea

Most models are not validated against “real” hardware

See “Architectural Simulators Considered Harmful”
https://doi.org/10.1109/MM.2015.74

There are bugs!
Getting (more) help

Main gem5 website: [http://gem5.org/](http://gem5.org/)

More like this:

[https://www.gem5.org/documentation/learning_gem5/introduction/](https://www.gem5.org/documentation/learning_gem5/introduction/)

Mailing lists: [http://gem5.org/Mailing_Lists](http://gem5.org/Mailing_Lists)

- **gem5-users**: General user questions (you probably want this one)
- **gem5-dev**: Mostly code reviews and high-level dev talk

gem5 slack: [https://tinyurl.com/gem5slackinvite](https://tinyurl.com/gem5slackinvite)

Jira issue tracker: [https://gem5.atlassian.net/](https://gem5.atlassian.net/)