GEM5 Extensions: Broadening Support to Microcontrollers with GUI

Ashutosh Vishwakarma

IIT Jammu

April 19, 2025

Guide: Dr.Subhasis Bhattacharjee

Overview

What is GEM5?

- **GEM5** is a modular platform for computer-system architecture research.
- It merges features from **M5** (full system simulator) and the original **GEM5** (CPU modeling framework).
- Supports simulation of a wide range of ISAs: **x86, ARM, RISC-V, SPARC, MIPS**, and more.
- Enables both **system-level** and **cycle-accurate** CPU simulation.
- Widely used for:
  - Academic and industrial architecture research
  - Performance analysis and profiling
  - Design-space exploration
- **Open-source** and highly extensible — ideal for custom architecture extensions.

What is GEM5?

- **GEM5** is structured into multiple components that interact modularly:
    - **CPU Models**: TimingSimple, Minor, O3, and atomic models.
    - **Memory System**: Includes caches, memory controllers, and interconnects.
    - **Devices**: Peripheral models like UART, timers, and disk controllers.
    - **ISAs**: Support for multiple instruction sets like ARM, RISC-V, x86, etc.
    - **Full System vs. Syscall Emulation Modes**
- Designed to be **modular and extensible**, allowing researchers to plug in new components.

What is GEM5?

- **Hybrid C++ and Python Design**:
  - Core simulation engine written in C++ for performance.
  - Configuration and modeling interfaces written in Python for flexibility.
- **User Interaction**:
  - Users write Python scripts to define system architecture, CPUs, memory, peripherals.
  - These scripts internally call C++ classes and methods via Python bindings.
- **Simulation Flow**:
  - Python config $\rightarrow$ System Instantiation $\rightarrow$ Simulation via C++ backend.
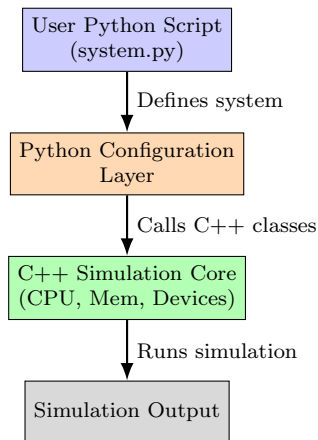
What is GEM5?

Figure 1: Simulation flow in GEM5

Goals

- Extend support for diverse microcontroller architectures.
- Target ISAs include ARM Cortex-M and AVR.
- Enable modeling of custom peripherals and I/O devices.
- Incorporate interrupt-driven architecture support.
- Facilitate real-time performance analysis and monitoring.
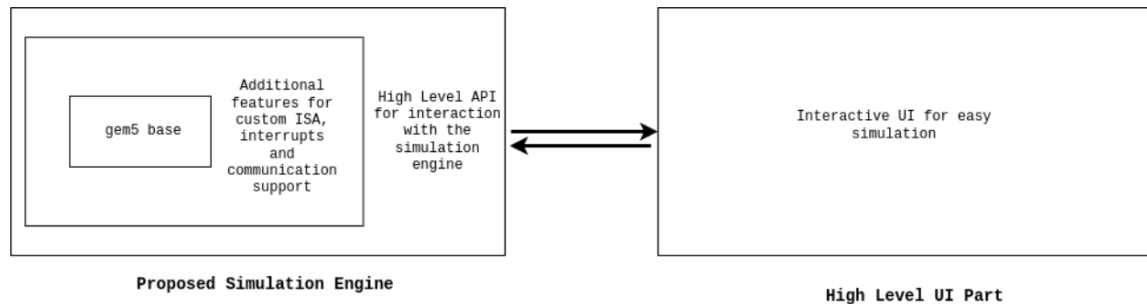- Provide tools for debugging and execution trace visualization.

System Overview

Proposed Simulation Engine

High Level UI Part

Figure 2: Proposed System Architecture

Why It Matters

- **Microcontrollers power the embedded world** — from smart homes to medical devices.
- Existing GEM5 support focuses mainly on general-purpose computing architectures.
- Extending GEM5 enables researchers to:
    - Model real-world embedded and IoT systems.
    - Explore architectural trade-offs in low-power environments.
    - Test RTOS behavior and peripheral interactions in a controlled simulation.
- Opens doors for hardware-software co-design in constrained systems.
- Facilitates early-stage development and validation of next-gen embedded solutions.

Why It Matters

- **Existing Simulators:**
  - *AVR Simulator, SimAVR, Proteus:* Limited scope, vendor-specific.
  - *QEMU:* General-purpose, lacks fine-grained modeling for peripherals and interrupts in microcontrollers.
  - *Renode:* Powerful, but more focused on specific embedded use cases.
- **Limitations:**
  - Minimal hardware-software co-design flexibility.
  - Limited extensibility for new ISAs and custom architectures.
  - Often lack integrated performance profiling and debugging tools.
- **GEM5-based Simulator:**
  - Fully extensible with support for custom ISAs like AVR and ARM Cortex-M.
  - Integration with real-time operating systems and custom peripherals.
  - High-level UI for configuring and analyzing simulations.
  - Enables in-depth architectural research and exploration.

Approach

- **Step 1: Research and Analysis** – Study GEM5 internals, identify extensible components.
- **Step 2: ISA Integration** – Add support for AVR architecture.
- **Step 3: Peripheral & Interrupt Modeling** – Develop modules for custom I/O and interrupt logic.
- **Step 4: UI Development** – Build an intuitive visual interface for simulation configuration.
- **Step 5: Debug Tools & Testing** – Add real-time performance monitoring, debugging, and trace tools.

Step 1: Research and Analysis

- **Exploring GEM5 Internals:**
  - Investigated the core components of GEM5 including CPU models, memory systems, and simulation flow.
  - Found limited developer documentation — referred to the official Doxygen documentation 🔗.
  - Studied through 3,358 source files comprising 965,657 lines of code.
- **Studying AVR Architecture:**
  - Reviewed instruction set, register architecture, and interrupt handling for AVR microcontrollers.
- **Feasibility Analysis:**
  - Evaluated how a lightweight 16-bit AVR ISA could be integrated into GEM5's CPU model framework.
  - Focused on maintaining modularity and clean abstraction layers.
- **Initial Outcomes:**
  - Conducted a simulation-based evaluation titled `Comparative Study on Execution Speed & CMR of Various ISAs` 🔗.
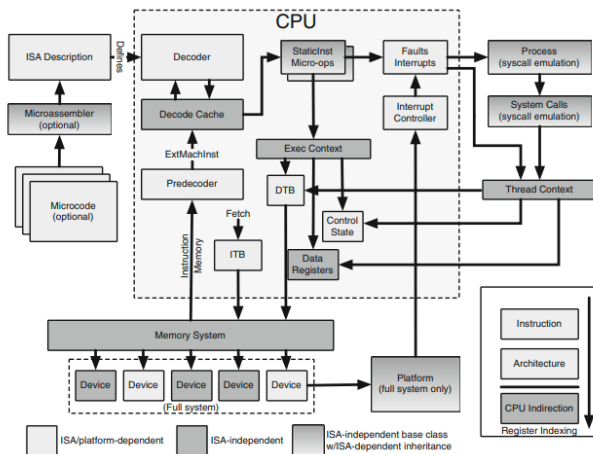
Step 2: ISA Integration

Figure 3: Dependence of gem5 components on ISA

Step 2: ISA Integration

- For the project we need to implement:
  - ISA Description
  - Decoder
  - Fault Interrupts
  - Predecoder
  - Python Bindings to CPU & I/O models.
  - GUI for the simulator
- For the begging a set of basic instructions are being implemented:
  - **Arithmetic Instructions:** ADD, SUB
- Current implementation includes:
  - ISA Description with two above-mentioned instructions
  - `AVRFault` for fault handling
  - **Registers:** 32 general-purpose registers (R0-R31), SREG (Status Register)
  - **Program Counter (PC):** 16-bit PC for instruction addressing
  - **Instruction Fetch and Decode:** Basic fetch-decode-execute cycle
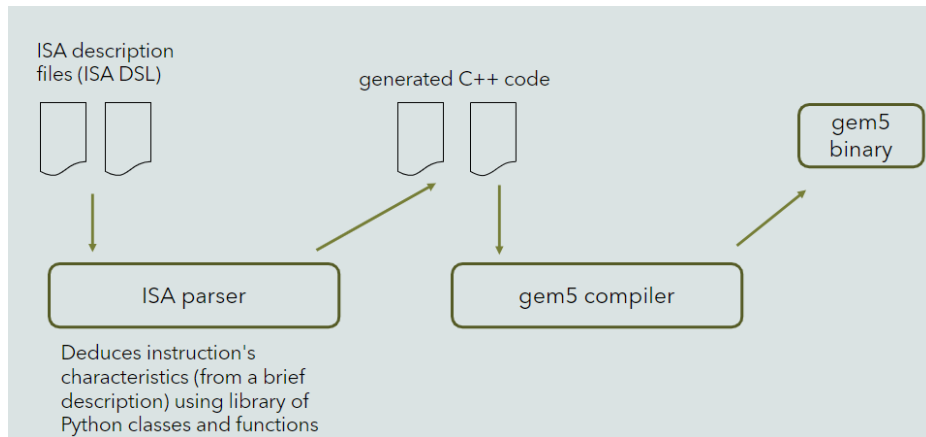
Step 2: ISA Integration

Figure 4: ISA description flow

Step 2: ISA Integration

Traversing the pipeline with 'add' instruction:

- Bitfield definition for the instructions to be later used in the decoder:

```
def bitfield OPCODE    <15:10>;
def bitfield REG_D     <8:4>;
def bitfield REG_R     <3:0>;
def bitfield IMM8      <7:0>;
```

Step 2: ISA Integration

- Format declaration for `add` instruction. The declaration defines the output blocks for the code to be generated:

```
    def format Add(code, *opt_args) {{
      iop = InstObjParams(name, Name, 'AddOp',
                          {'code': code,
                            'predicate_test': predicateTest,
                            'op_class': 'gem5::enums::IntAlu'
                          },
                          opt_args)
      header_output = AddDeclare.subst(iop)
      decoder_output = AddConstructor.subst(iop)
      decode_block = AddDecode.subst(iop)
      exec_output = AddExecute.subst(iop)
      disasm_output = AddDisassembly.subst(iop)
    }};
```

Step 2: ISA Integration

- Add class declaration including the constructor, execute method, disassembly method and PC advancement method:

```
def template AddDeclare {{
  class %(class_name)s : public gem5::AVRISAInst::AVRStaticInst
  {
    public:
      %(class_name)s(gem5::AVRISAInst::MachInst machInst);
      Fault execute(ExecContext *, trace::InstRecord *) const override;
      std::string generateDisassembly(Addr pc,
           const loader::SymbolTable *symtab) const override;
      void advancePC(PCStateBase &pc_state) const;
  };
}};
```

Step 2: ISA Integration

- Generated C++ code for the Add class from the DSL code above:

```cpp
#undef OPCODE
#define OPCODE     bits(machInst, 15, 10)
#undef REG_D
#define REG_D      bits(machInst,  8,  4)
#undef REG_R
#define REG_R      bits(machInst,  3,  0)
#undef IMM8
#define IMM8       bits(machInst,  7,  0)
class Add : public gem5::AVRISAInst::AVRStaticInst
{
  public:
    Add(gem5::AVRISAInst::MachInst machInst);
    Fault execute(ExecContext *, trace::InstRecord *) const override;
    std::string generateDisassembly(Addr pc,
        const loader::SymbolTable *symtab) const override;
    void advancePC(PCStateBase &pc_state) const;
};
```

Step 2: ISA Integration

- Getting integration success with these two instructions opened door for further extending the instruction set.
- The groundwork for extension is laid out with the fault handling, PC management and register modeling.

Future Work

- Develop `Python` binding for **AVR CPU** model to execute the AVR instructions.
- Implement the memory model the CPU (Harvard architecture although GEM5 currently only supports Von Neumann architecture).
- Implement the `AVRIO` model to handle the I/O operations.
- Benchmark testing.
- Extracting the runtime statistics from the simulator to a GUI.

Conclusion

- Extended **GEM5** to support **microcontroller architectures**, specifically **AVR**.
- Built a foundation for **custom ISA integration**:
  - Instruction decoding
  - Fault handling
  - Register modeling
- Implemented groundwork for **interrupt-driven simulation**.
- Proposed a plan for **GUI-based configuration and visualization**.
- Enables:
  - **In-depth architectural research**
  - **Hardware-software co-design**
  - **Early validation** of embedded systems

References I

[1] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, *"Accuracy evaluation of GEM5 simulator system,"* in 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2012.

[2] S. V. Bharadwaj and C. K. Vudadha, *"Evaluation of x86 and ARM architectures using compute-intensive workloads,"* in 2022 IEEE International Symposium on Smart Electronic Systems (iSES), 2022.

[3] R. Saha, Y. P. Pundir, S. Yadav, and P. K. Pal, *"Impact of size, latency of cache-L1 and workload over system performance,"* in 2020 International Conference on Advances in Computing, Communication Materials (ICACCM), 2020.

[4] A. D. George, *"An overview of RISC vs. CISC,"* in Proceedings of the Twenty-Second Southeastern Symposium on System Theory, 1990.

[5] M. Ling, X. Xu, Y. Gu, and Z. Pan, *"Does the ISA really matter? A simulation-based investigation,"* in 2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 2019.

References II

[6] T. Jamil, *"RISC versus CISC,"* IEEE Potentials, vol. 14, no. 3, pp. 13–16, 1995.

[7] A. A. Abudaqa, T. M. Al-Kharoubi, M. F. Mudawar, and A. Kobilica, *"Simulation of ARM and x86 microprocessors using in-order and out-of-order CPU models with gem5 simulator,"* in 2018 5th International Conference on Electrical and Electronic Engineering (ICEEE), pp. 317–322.

[8] B. Vikas and B. Talawar, *"On the cache behavior of Splash-2 benchmarks on ARM and Alpha processors in gem5 full system simulator,"* in 2014 3rd International Conference on Eco-friendly Computing and Communication Systems, pp. 5–8.

[9] S. Lee, Y. Kim, D. Nam, and J. Kim, *"Gem5-AVX: Extension of the Gem5 Simulator to Support AVX Instruction Sets,"* IEEE Access, 2024.

Supplementary Materials

[1] Ashutosh Vishwakarma, *"Comparative Study on Execution Speed and CMR of Various ISAs using gem5."* Available at: *https://drive.google.com/...*

[2] Ashutosh Vishwakarma, *"BTP Project Logs."* Weekly logs and development notes. Available at: *https://shorturl.at/0oIEB*

[3] Ashutosh Vishwakarma, *"gem5 Project GitHub Repository."* Source code and simulation scripts. Available at: https://github.com/ragnar-vallhala/gem5.git

[4] *"gem5 Developer Documentation."* Official documentation for developers. Available at: *https://www.gem5.org/documentation/*

[5] *"ISCA 2024: gem5 Workshop and Tutorial."* Gem5 workshop details. Available at: *https://www.gem5.org/events/isca-2024*

[6] *"Adding Instructions to gem5."* A bootcamp guide to custom instruction modeling. Available at: *https://shorturl.at/1WLoB*

[7] *AVR ISA Reference Manual.* AVR instruction set and architecture details. Available at: *http://www.mmajunke.de/...*

Thank You

Questions?