System Simulation with gem5, SystemC and other Tools

Christian Menard (TU Dresden, Germany)
Matthias Jung (Fraunhofer IESE, Germany)
Detailed Description of Contents for this Talk

System Simulation with gem5 and SystemC

The Keystones for Full Interoperability

Christian Menard, Jeremie Castillon
Technische Universität Dresden
Dresden, Germany
Email: christian.menard@tu-dresden.de
Jeremie.castillon@tu-dresden.de

Matthias Jung
Fraunhofer HHI
Kaiserslautern, Germany
jungm@hhi.fraunhofer.de

Norbert Weiss
University of Kaiserslautern
Kaiserslautern, Germany
weissn@cs.uni-klu.de

Abstract—SystemC TLM based system prototypes have become an essential tool in industry and research for concurrent design and software development, as well as early prototyping and space exploration. However, there exists a lack of seamlessly, continuously and automatically validated SystemC models of modern processors. Therefore, many research groups are the cycle accurate open source SystemC Simulation Environment (SCSim) that provides support for simulation studies of processors and other hardware systems. The SCSim is designed to be highly portable and can be used with other simulation tools to create complex system models.

I. INTRODUCTION

Today's companies have to deal with complex hardware architectures such as multithreaded, sophisticated interconnects and memory systems. Virtually everything (VPE) [1] are widely used to allow for early design space exploration and to discover the time-to-market (TTM) costs and efforts by developing software and hardware concurrently. They are high-speed, fully functional soft models of physical hardware systems that can simulate the exact behavior of real hardware. With their help, complex Processor System-on-Chips (PSoC) can be simulated with reasonable simulation speed and visibility and controllability over the entire system. This allows designers to explore designs with minimal effort. However, these models are limited to simulating small parts of the system, such as individual processors or memory subsystems. In contrast, our system models, Advanced SystemC, model the entire system. The model is based on the gem5 framework, which is a modular platform for computer-system architecture research [2]. It is not only very useful for education, but also the industry employs gem5 for research. For instance, ARM and AMD use gem5 extensively for design space exploration and actively contributes to the open source project. However, gem5 is not limited to academia, as its development started before the IEEE certified the official SystemC and TLM standard in 2010 [2]. Since then, both frameworks, gem5 and SystemC, have evolved significantly in parallel. Therefore, gem5 is compatible to TLM models that exist in industry and academia.

In this paper, we present for the first time a comprehensive coupling between SystemC and gem5 that provides full interoperability. Through this coupling, any SystemC models that implement the TLM interface can be connected to any gem5 models, as shown in Figure 1. To the best of our knowledge, there exists no reference which describes system and semantics of both frameworks and how both simulation kernels can be coupled in order to enable full interoperability.

Paper: [Link]

Sources: [gem5/utils/tlm/README]
Virtual Prototyping
Virtual Prototypes in Industry

Functional software models of physical hardware:
- Visibility and controllability over the entire system
- Powerful debugging and analysis tools
- Reuse of components for future projects
- Fast Design Space Exploration (for HW engineers)
- Easy to exchange, worldwide
- Concurrent HW and SW development:

![Diagram showing effort and time-to-market for hardware, software development, testing/integration, and product support/maintenance.]

Software development

Hardware development

Testing / Integration

Product support and maintenance
Virtual Prototypes in Industry

Functional software models of physical hardware:
- Visibility and controllability over the entire system
- Powerful debugging and analysis tools
- Reuse of components for future projects
- **Fast Design Space Exploration** (for HW engineers)
- Easy to exchange, worldwide
- Concurrent HW and SW development:

![Effort vs. Time-to-Market Diagram]

- Earlier TTM
- Higher Quality

...
Simulate of (widely) heterogeneous systems

- Many different models of cores, accelerators, and communication infrastructure required.

Simulation of the Memory Subsystem

- Focuses on the memory subsystem, but detailed simulation of realistic workloads is required.
SystemC IEEE 1666

- Modeling language for HW and SW components
- Extends C++ to an event-driven simulation kernel
- Various levels of accuracy
- IEEE Standard, Maintained by Accellera
- 10-100x Faster than CA VHDL/Verilog Simulation

→ However, normal CA SystemC is not fast enough to, e.g., boot an OS.
Transaction Level Modeling (TLM)

- CA SystemC
  - Pin Accurate
  - Simulate each pin separately
- TLM
  - Function Call
  - Simulate transactions up to 10,000x Faster

Source: Doulos Ldt. www.doulos.com
Transaction Level Modeling (TLM)

Simulate each pin separately

Simulate transactions up to 10,000x Faster
Generic Payload

- **Initiator (CPU)**
- **Interconnect (BUS)**
- **Target (MEM)**
- **Target (I/O)**
Generic Payload

Generic payload object:
- Command
- Address
- Data
- Byte Enables
- Response Status
- Extensions

Payload reference:
- Initiator (CPU)
- Interconnect (BUS)
- Target (MEM)
- Target (I/O)

Socket connections:
- Initiator Socket
- Target Socket
- Initiator Socket
- Target Socket
- Initiator Socket
- Target Socket
Generic Payload

Payload reference

Initiator (CPU)

Initiator Socket

Target Socket

Interconnect (BUS)

Command
Address
Data
Byte Enables
Response Status

Extensions

Target (MEM)

Target Socket

Target Socket

Target Socket

Target Socket

Target Socket

Target Socket

Generic payload object

Payload reference
**TLM Coding Styles and Mechanisms**

**TLM Use Cases**

- SW Application Development
- SW Performance Analysis
- Architecture Analysis
- Hardware Verification

**TLM 2.0 Coding Style** *(Just Guidelines)*

- **Loosely-timed**
  - Single-phase, blocking API
  - `debug_transport`, `b_transport`

- **Multi-phase, non-blocking API**
  - Approximately -timed

**TLM Mechanisms** *(Definitive API for enabling Interoperability)*

- Blocking transport
- DMI
- Quantum
- Sockets
- Generic payload
- Extensions
- Phases
- Non-blocking transport

Source: Doulos Ldt. www.doulos.com
Tool Vendors for TLM 2.0 VP

TLM is widely used in Industry:

- The market of virtual platform tools:
  - Synopsys - Platform Architect
  - Cadence - Virtual System Platform
  - Mentor Graphics - Vista Virtual prototyping
  - Imperas - OpenVP
  - ASTC - VLAB Works

- Virtual Platform Core Models:
  - ARM (Fastmodels):
    - only LT models based on JIT, non-free, library
  - ARM Carbon (Former Carbon Design Systems):
    - Cycle Accurate (CA) Models in TLM Wrapper, non-free, library
  - Imperas / OVP:
    - only LT, Free

→ An accurate, free available and changeable core model is needed
Coupling gem5 with SystemC
gem5 supports a SystemC coupling:
- Gem5 is built as a C++ library.
- It is linked into a SystemC simulation.
- A SystemC object implements the gem5 event queue.
- **How can we communicate with other SystemC modules?**
Transaction Models in gem5

**Timing**
- The most detailed access: queuing delay + resource contention
- Similar to the TLM `nb_transport` interface.

**Atomic**
- Accesses are a faster than detailed access
- Used for **fast forwarding** and **warming up caches**
- Similar to the TLM `b_transport` interface
- Not good for performance simulation

**Functional**
- Similar to `transport_dbg` e.g. loading binaries, avoiding deadlocks in multi-level cache coherent networks
Converting between TLM and gem5

**Master** - External Slave - **Slave Transactor** - **Target**

- `recvFunctional(...)` → `transport_dbg(...)`
- `recvAtomic(...)` → `b_transport(...)`
- `recvTimingReq(...)` → `nb_transport(...)`

**Initiator** - **Master Transactor** - **External Master** - **Slave**

- `transport_dbg(...)` → `recvFunctional(...)`
- `b_transport(...)` → `recvAtomic(...)`
- `nb_transport(...)` → `recvTimingReq(...)`
Transaction Explained

CPU  BUS  External Slave

Slave Transactor

Memory

gem5 World  SystemC World
Transaction Explained

gem5 Packet
- Cmd
- Data
- Addr
- Size
- Flags
  Sender State

CPU & BUS
  gem5 World

External Slave
Slave Transactor

Memory
  SystemC World
Transaction Explained

gem5 Packet
- Cmd
- Data
- Addr
- Size
- Flags

Sender State

CPU

BUS

External Slave

Slave Transactor

Memory

gem5 World

SystemC World
Transaction Explained

**gem5 Packet**
- Cmd
- Data
- Addr
- Size
- Flags

**Sender State**

**Generic payload object**
- command
- data_ptr
- address
- data_length
- byte_enable_ptr
- streaming_width

**Extensions**

**gem5 World**
- CPU
- BUS
- External Slave

**Slave Transactor**

**SystemC World**
- Memory
Transaction Explained

gem5 Packet
- Cmd
- Data
- Addr
- Size
- Flags

Sender State

Generic payload object
- command
- data_ptr
- address
- data_length
- byte_enable_ptr
- streaming_width

Extensions

CPU
BUS
External Slave
Slave Transactor
Memory

gem5 World
SystemC World
Transaction Explained

**gem5 Packet**
- Cmd
- Data
- Addr
- Size
- Flags

**Sender State**

**Generic payload object**
- command
- data_ptr
- address
- data_length
- byte_enable_ptr
- streaming_width

**Extensions**

**CPU** & **BUS**

**External Slave**

**Slave Transactor**

**Memory**

**gem5 World**

**SystemC World**
Transaction Explained

gem5 Packet
- Cmd
- Data
- Addr
- Size
- Flags

Sender State

Generic payload object
- command
- data_ptr
- address
- data_length
- byte_enable_ptr
- streaming_width

Extensions

gem5 World
- CPU
- BUS
- External Slave

Slave Transactor

SystemC World
- Memory
Transaction Explained

```
<table>
<thead>
<tr>
<th>gem5 Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Addr</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Flags</td>
</tr>
</tbody>
</table>

Sender State

```

```
<table>
<thead>
<tr>
<th>Generic payload object</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
</tr>
<tr>
<td>data_ptr</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>data_length</td>
</tr>
<tr>
<td>byte_enable_ptr</td>
</tr>
<tr>
<td>streaming_width</td>
</tr>
</tbody>
</table>

Extensions

```

```
<table>
<thead>
<tr>
<th>gem5 World</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>BUS</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>SystemC World</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Slave Transactor</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>External Slave</th>
</tr>
</thead>
</table>

```

UPDATE
Transaction Explained

gem5 Packet

- Cmd
- Data
- Addr
- Size
- Flags

Sender State

CPU & BUS

gem5 World

External Slave

Slave Transactor

Memory

SystemC World
How to get Started?
How to get Started?

- Study the Examples in `/gem5/utils/tlm/`
  - **Slave Example:**
    - gem5 TrafficGen
    - membus
    - External Slave
    - Slave Transactor
    - TLM Simple Memory
  
  - **Master Example:**
    - TLM Traffic Generator
    - Master Transactor
    - External Master
    - membus
    - gem5 Memory
  
- Elastic Trace Example [5] (see left)

- Full System Example:
  ```
  ../../../build/ARM/gem5.opt ../../../configs/example/fs.py
  --tlm-memory=transactor --cpu-type=TimingSimpleCPU --num-cpu=1
  --mem-type=SimpleMemory --mem-size=512MB --mem-channels=1 --caches
  --l2cache --machine-type=VExpress_EMM
  --dtb-filename=vexpress.aarch32.ll_20131205.0-gem5.1cpu.dtb
  --kernel=vmlinux.aarch32.ll_20131205.0-gem5
  --disk-image=linux-aarch32-ael.img
  ```
Practical Usage: General Flow

1. Compile gem5 normally: 
   \texttt{scons build/ARM/gem5.opt}

2. Compile gem5 as a library:
   \texttt{scons --with-cxx-config --without-python --without-tcmalloc \ build/ARM/libgem5_opt.so}

3. Include the gem5 modules \texttt{Gem5SimControl} and \texttt{Gem5SlaveTransactor}
   and/or \texttt{Gem5MasterTransactor} in your SystemC project and connect them to
   your SystemC models. Be sure to pass an individual port name to the constructor of each transactor.

4. Compile your project and link against the gem5 library.

5. Run normal gem5 with a custom python script or \texttt{fs.py} with
   \texttt{--tlm-memory=<port-name>} to generate \texttt{m5out/config.ini}. Be sure to set
   the \texttt{tlm_data} attribute of the External Masters/Slaves to the port name of the corresponding SystemC transactor.

6. Run your SystemC project and pass the \texttt{m5out/config.ini} file to your
   \texttt{Gem5SimControl} object.
struct Target: public sc_module {

    // TLM interface socket:
    tlm_utils::simple_target_socket<Target> socket;

    // Storage
    unsigned char *mem;

    // Constructor
    Target(sc_core::sc_module_name name, /* ... */);
    SC_HAS_PROCESS(Target);

    // TLM interface functions
    virtual void b_transport(tlm::tlm_generic_payload& trans,
                              sc_time& delay);
    virtual unsigned int transport_dbg(tlm::tlm_generic_payload& trans);
    virtual tlm::tlm_sync_enum nb_transport_fw(
        tlm::tlm_generic_payload& trans,
        tlm::tlm_phase& phase,
        sc_time& delay);

    // ...
};

util/tlm/examples/slave_port/sc_target.hh
Hands On: Connect the Memory to gem5

```c
int sc_main(int argc, char **argv)
{
    // Instantiate all modules
    Gem5SystemC::Gem5SimControl sim_control("gem5", /* config ... */);
    Gem5SystemC::Gem5SlaveTransactor transactor("transactor", "transactor");
    Target memory("memory", /* config ... */);

    // Bind modules
    memory.socket.bind(transactor.socket);
    transactor.sim_control.bind(sim_control);

    // Start simulation
    sc_core::sc_start();

    return EXIT_SUCCESS;
}
```

→ util/tlm/examples/slave_port/main.cc
Hands On: Configure gem5

```python
# Create a system with a Crossbar and a TrafficGenerator
system = System()
system.membus = IOXBar(width = 16)
# This must be instanciated, even if not needed
system.physmem = SimpleMemory()
system.cpu = TrafficGen(config_file = "tgen.cfg")
system.clk_domain = SrcClockDomain(clock = '1.5GHz',
                       voltage_domain = VoltageDomain(voltage = '1V'))

# Create an external TLM port:
system.tlm = ExternalSlave()
system.tlm.addr_ranges = [AddrRange('512MB')]
system.tlm.port_type = "tlm_slave"
system.tlm.port_data = "transactor"

# Route the connections:
system.cpu.port = system.membus.slave
system.system_port = system.membus.slave
system.membus.master = system.tlm.port

# Start the simulation:
root = Root(full_system = False, system = system)
root.system.mem_mode = 'timing'
m5.instantiate()
m5.simulate()
```

```python
util/tlm/conf/tlm_slave.py
```
Hands On: Run the Simulation

1. **Build the example:**
   
   ```
   $ cd util/tlm && scons
   ```

2. **Create a gem5 config.ini file:**
   
   ```
   $ ../../build/ARM/gem5.opt conf/tlm_slave.py
   ```

3. **Run the simulation:**
   
   ```
   $ build/examples/slave_port/gem5.sc m5out/config.ini
   ```
$ build/examples/slave_port/gem5.sc m5out/config.ini -e 200000 -d TrafficGen

[...]

0 s (=) : sc_main Start of Simulation
info: Entering event queue @ 0. Starting simulation...
5 ns (=) : system.cpu LinearGen::getNextPacket: r to addr 0, size 4
5 ns (=) : system.cpu Next event scheduled at 10000
10 ns (=) : system.cpu LinearGen::getNextPacket: w to addr 4, size 4
15 ns (=) : system.cpu Received retry
15 ns (=) : system.cpu LinearGen::getNextPacket: r to addr 8, size 4
16675 ps (=) : system.cpu Received retry
75 ns (=) : system.cpu Received retry
75 ns (=) : system.cpu LinearGen::getNextPacket: r to addr c, size 4
76038 ps (=) : system.cpu Received retry
135 ns (=) : system.cpu Received retry
135 ns (=) : system.cpu LinearGen::getNextPacket: r to addr 10, size 4
136068 ps (=) : system.cpu Received retry
195 ns (=) : system.cpu Received retry
195 ns (=) : system.cpu LinearGen::getNextPacket: w to addr 14, size 4
196098 ps (=) : system.cpu Received retry
Exit at tick 200000, cause: simulate() limit reached

➤ The binary excepts various options:
  • -e end of simulation at tick
  • -d set a gem5 debug flag
Usecases
The Orchestration Path at CFAED [2]

A programming stack for wildly heterogeneous systems including:

- dataflow programming models
- dataflow compiler
- adaptive runtime systems
- capability-based OS
- model checker

➤ A flexible simulation platform is required to try new designs and technologies.
Building Heterogeneous MPSoCs: gem5 as a Tile
gem5 in Synopsys Platform Architect
gem5 in Synopsys Platform Architect
gem5 in Synopsys Platform Architect: Trace Analysis

- **Baseline**: Cursor = 0
- **Current**: Cursor = 996528 ns
- **Diff**: 996528 ns

**PE0 (Kernel)**

- TLM Port Trace
  - PE0.lnoc_socket
  - PE0.lnoc_socket
  - Configure...

**PE1 (Worker)**

- TLM Port Trace
  - PE0.lnoc_socket
  - PE0.lnoc_socket
  - Configure...

**PE2 (Worker)**

- TLM Port Trace
  - PE0.lnoc_socket
  - PE0.lnoc_socket
  - Configure...

**PE8 (RAM)**

- TLM Port Trace
  - PE0.lnoc_socket
  - PE0.lnoc_socket
  - Configure...
### gem5 in Synopsys Platform Architect: Trace Analysis

#### Details

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARDWARE PE08.1 noc_socket</td>
<td></td>
</tr>
<tr>
<td>Formatted Value</td>
<td>W addr=0x0 data={}</td>
</tr>
<tr>
<td>Generator Name</td>
<td>Tlm2WtTrans</td>
</tr>
<tr>
<td>Port Name</td>
<td>HARDWARE PE08.1 noc_socket</td>
</tr>
<tr>
<td>Stream Name</td>
<td>HARDWARE PE08.1 noc_socket</td>
</tr>
<tr>
<td>Time Begin</td>
<td>996588 ns</td>
</tr>
<tr>
<td>Time End</td>
<td>996608 ns</td>
</tr>
</tbody>
</table>

#### TLM Port Trace

```
<table>
<thead>
<tr>
<th>PE08.1 noc_socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE08.1 noc_socket</td>
</tr>
<tr>
<td>PE08.1 noc_socket</td>
</tr>
<tr>
<td>Configure...</td>
</tr>
</tbody>
</table>
```

---

**VP Explorer:** pid=6178, /home/cmenard/synopsys/workspace/export

**Cursor:** 996528 ns

**Graph:**
- Timelines: 1 ms, 1500 us, 2 ms
- PE8 (RAM)
Coupling gem5 with DRAMSys [3], [4]

DRAMSys is a design space exploration framework for DRAM and memory controller It includes:
- Power model
- Thermal model
- Retention error model.

Linux boot (without thermal model) using the DRAMSys model → slowdown of 1.9×

This slowdown mostly comes from detailed DRAMSys model
Coupling gem5 with DRAMSys Continued
Coupling gem5 with DRAMSys Continued
Coupling gem5 with DRAMSys Continued
Coupling gem5 with DRAMSys Continued
Connecting gem5 to Non-SystemC Simulators

JNI Wrapper

SW:
- Bare Metal
- Linux
- Ubuntu
- RTOS

CPU
- L1I
- L1D

I/O

DRAM

Shadow Device

IRQ

SystemC Model
(e.g. other ISA)

CAN BUS Simulator

FERAL Simulation Framework [6]

Fraunhofer IESE

Simulink Model

...
Scenarios with FERAL

- Coupling of different Simulators with different models of computation (e.g. Simulink with gem5 or SystemC)
- gem5 compiled as C++ library and wrapped in JNI wrapper

- Development of software concepts
  - Simulation of systems of systems
  - Combining different levels of abstraction

- Software Testing:
  - Normal software testing instruments source code or binary → Intrusive
  - gem5 is instrumented instead
  - Supervised testing of concurrent software
  - Fault Injection
  - Coverage
Summary and Outlook

- Full interoperability between gem5 and SystemC
- Fully compliant to the SystemC standard
- It is part of the gem5 repository!

→ Next steps:
  → replace the entire simulation kernel and communication System by SystemC/TLM (?)
  → Remove step for .ini generation (?)
  → Suggestions? We are open!

Thank you!
References


