System Architecture Performance Modeling with SystemC

9th European SystemC Users Group Meeting

February 17th, 2004

Soeren Sonntag
Raimar Thudt
Tino Bertram
Outline

1. Field of application
2. Motivation for system architecture performance modeling
3. Simulation model overview
4. Simulation performance
5. Issues regarding SystemC
6. Future Steps
7. Conclusions
Field of Application: Interworking Scenario

- CPE
- access net
- core net

ATM over DSL

system architecture simulation
System Architecture Simulation – Why?

- Used for performance modeling of large designs
- Abstract models → simple
- Very high simulation speed
- Allows simulation during concept phase
- Detection of bottlenecks in design
- Easy evaluation of performance of different design architectures
- Easy verification

- But: Not possible with hardware description languages like VHDL or Verilog
System Architecture Simulation – How?

- SystemC 2.0.1
  - Function blocks represented by `sc_module`
  - Communication via `sc_channel`

- Top-down refinement starting at system level
- Refinement only for “critical” modules
SystemC Models

- Behavioral description
- Modules have only function, necessary to evaluate performance
- Modules consume time
- No hardware signals
- No clocks!

→ Use of queuing models
SystemC Queuing Model

- Source
  - generate request

- Queue
  - store determined amount of requests
  - order requests according to a given queue discipline (FIFO, LIFO, Round Robin, ...)

- Server
  - dedicated functionality
  - dedicated service time per request

- Sink
  - perform statistical analysis
Interworking Function – Overview
SystemC Model Refinement

- Model with high abstraction

```c
SC_MODULE(CAM) { // pseudo code
  ...
  void serve() {
    in->read(request);
    wait(service_time);
    out->write(automagically_created_data);
  }
  ...
};
```

- Model with less abstraction

```c
... 
void serve() {
  in->read(request);
  do_something_reasonable();
  wait(service_time);
  out->write(reasonable_data);
}
... 
```
## Simulation Performance

- **Simulation environment:** 2.4 GHz Linux PC

<table>
<thead>
<tr>
<th></th>
<th>model with high abstraction</th>
<th>model with less abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>model contains</strong></td>
<td>several hundreds of sources with exponentially distributed traffic (Imix)</td>
<td>154 sc_modules, 678 sc_channels</td>
</tr>
<tr>
<td><strong>lines of code</strong></td>
<td>ca. 11,000</td>
<td>ca. 15,000</td>
</tr>
<tr>
<td><strong>difference</strong></td>
<td>without real RAM access</td>
<td>with RAM access</td>
</tr>
<tr>
<td><strong>performance</strong></td>
<td>1 : 300  (5 min)</td>
<td>1 : 1800  (1/2 hour)</td>
</tr>
<tr>
<td></td>
<td>(one second of chip operation in 300 (1800) seconds of wall-clock time)</td>
<td></td>
</tr>
</tbody>
</table>

A huge trade-off between architectural correctness and simulation performance.
Issues Regarding SystemC Due to Abstraction (1)

- Issue: No event queue available
  (only first event survives)

- Example:

```c
event.notify(10, SC_NS);
event.notify(20, SC_NS);
wait(event);                // returns at 10 ns
wait(event);                // never returns
```

- Severity: Medium
Issues Regarding SystemC Due to Abstraction (2)

- Issue: Processes do not know by which event they have been triggered

- Example:

  ```
  ... // pseudo code
  sc_port<int, 512> in; // multiport with 512 channels
  ...
  sensitive << in;
  ...
  wait();
  // Which channel bound to the port caused the event?
  ```

- Severity: High

- Workarounds exist, but implementation into SystemC kernel might be more efficient
Future Steps & Conclusions

Future steps
- Extended use of SystemC for system architecture performance modeling and simulation
- Use of SystemC for multi-level simulation (but still above register-transfer level)

Conclusions
- SystemC is a good choice for simulation at a high level of abstraction
- Issues exist, but could be fixed more or less easily
- SystemC is in our case faster than certain commercial tools
Questions?
Thank you for your attention!

Soeren Sonntag  soeren.sonntag@infineon.com
Raimar Thudt    raimar.thudt@infineon.com
Tino Bertram    tino.bertram@infineon.com

Never stop thinking.