Outline

➢ Introduction
➢ Objectives
➢ Virtual Platforms and SystemC
➢ Checkpointing for SystemC
➢ Conclusions
Introduction – Virtual Platforms

- Functional models of physical platforms
- Target SW unable to distinguish virtual platform from real HW
- Run SW or OS on Virtual HW
- Develop SW for non-existing HW
- Simulate complex system interconnectivity
Introduction – Virtual Platforms
Introduction – Virtual Platforms
Virtual Platform Design

- Hardware
- Software
- Integration & Test

© WindRiver (Virtutech)
Virtual Platform Design

© WindRiver (Virtutech)
Generic Virtual Platform Diagram

User | User | User
---|---|---
Target Operating System | Target Drivers | Boot code

Memories | Devices | ISS

VP Back-end functions

Virtual Platform
Introduction – SystemC & TLM-2

- SystemC language for systems description (HW mainly)
- OSCI simulator
- TLM-2 standardizes communication model
  - Sockets to emulate any memory-mapped bus
- De facto standard for system modeling
Introduction – VP Languages

➢ Different Virtual Platforms uses different languages (C, C++, DLM, ...) and APIs
➢ HW engineers know (or should) SystemC, not other languages
➢ Different VP → Rewrite own models
➢ No interoperability
Objectives

➢ Add SystemC-TLM to any Virtual Platform
  • Different strategies to add two simulators
  • Add SystemC-TLM support to an open-sourced VP
➢ Add checkpointing support to SystemC
  • C++ not checkpointable
  • Overcome limitations
Generic Virtual Platform Diagram

Virtual Platform

VP Back-end functions

ISS

SystemC Devices

Sync

SystemC

Target Drivers

Boot code

Target Operating System

User

User

User

Memories

Devices

ISS

Devices

Sync
Virtual Platforms and SystemC

- Link together two simulators
- Synchronization strategy
- Generic bridge
- Support for generic TLM-2 devices
  - LT, AT, DMI...
Link together two simulators

Virtual Platforms and SystemC

SystemC
Synch
VP
Virtual time
Virtual Platforms and SystemC

SystemC

Synch

VP

Virtual execution

Execution time
QEMU-SC

- Transition from RTL to TLM
- Generic fabric bus (TLM) → any architecture
- QEMU master and SystemC slave (simulation)
  - QEMU manages simulation
- Focus on few SystemC devices
QEMU-SC

Application

Linux

Driver

VP

SC_Link

SC_Bridge

TLM Socket

SystemC module
Synchronization

➢ Only synchronize when needed (sc_start())
  • When SystemC devices are accessed
  • When pending event in current simulation time
    – SystemC events list
  • When I/O to/from SystemC device
    – Capture or notify all I/O in SystemC device
  • QuantumKeeper asks to
    – To adhere to standard
Communication

- QEMU works with zero-delay communication
  - CPU accesses one device and the device responds immediately
- Fit to LT devices
- Need to manage AT devices
  - Special synchronization
  - Finish all protocol phases before return to VP
Communication - LT

VP → SC Bridge
operation()

SC Bridge → LT Device
b_transport()

LT Device → VP
operation()
Communication - AT

VP

SC Bridge

operation()

AT Device

BEGIN_REQ

BEGIN_RESP

END_REQ

END_RESP

operation()

nb_transport_fw()

nb_transport_bw()

nb_transport_bw()

nb_transport_fw()
Managing external I/O from/to SC device:
- Devices use QEMU callbacks for I/O
- Bridge knows when a I/O is performed
  - Synchronize simulators

Continue SystemC simulation when:
- VP time arrives to first SC event
- Every quantum time (TLM-2)

Advance SystemC time until transaction ends
QBox

- Change simulation manager
  - QEMU becomes simulator slave
- **SystemC** manages simulation
- QEMU is a TLM-2 Initiator module
- Easy integration
- Focus on many SystemC models
QBox

SystemC simulation

Wrapper

TLM Socket

QEMU

Target module
QBox internal architecture

- QEMU Wrapper
  - manage_int()
  - sc_write()
  - sc_read()
  - sc_dmi_write()
  - sc_dmi_read()

- SC_Link
- PCI Bus
- QEMU

- Ghost PCI dev 1
- Ghost PCI dev 2
QBox complex example
Test & results

- Different tests and examples
  - Validate implementation
  - Extract performance metrics

- Results for performance
  - Relatives to same system in native language
    - C in QEMU & Qbox
QEMU-SC Test System

- **MPEG-2 decoder**
- **Linux**
- **Driver**
- **SC_Link**
- **SC_Bridge**
- **QEMU**
- **MPEG accelerator**
- **SystemC iDCT acc.**
- **IRQ**
- **Reg. File**
- **DMI Pointer**
- **TLM Socket**
- **irq_event**

**Connections:**
- IRQ from MPEG accelerator to Reg. File
- DMI Pointer from SC_Bridge to Reg. File
- TLM Socket from QEMU to SC_Bridge

**Software Components:**
- **QEMU**
- **Linux**
- **Driver**
- **MPEG-2 decoder**

**Hardware Components:**
- **MPEG accelerator**
- **SystemC iDCT acc.**
- **Reg. File**
- **DMI Pointer**
- **IRQ**

**Other Notes:**
- The diagram illustrates the interaction between software and hardware components in a QEMU-SC test system.
QEMU-SC Results

- No acc.
- No acc. w/ drivers
- QEMU style 1 acc.
- SystemC 1 acc.
- QEMU style 2 acc.
- SystemC 2 acc.
- SystemC skel. (1 acc.)
- SystemC skel. (2 acc.)
QEMU-SC Results

- No acc.
- No acc. w/ drivers
- QEMU style 1 acc.
- SystemC 1 acc.
- QEMU style 2 acc.
- SystemC 2 acc.
- SystemC skel. (1 acc.)
- SystemC skel. (2 acc.)

Penalty 8% ~ 14%
QBox Test System

- QEMU Wrapper
  - MPEG-2 decoder
  - Linux Driver
- QEMU
- TLM Socket
- SignalSocket (IRQ)
- GreenRouter
- SystemC iDCT acc.
- IRQ
- Reg. File
  - irq_event
QBox Results

- No acc.
- No acc. w/ drivers
- QEMU style 1 acc.
- SystemC 1 acc.
- QEMU style 2 acc.
- SystemC 2 acc.
- SystemC skel. (1 acc.)
- SystemC skel. (2 acc.)
QBox Results

- No acc.
- No acc. w/ drivers
- QEMU style 1 acc.
- SystemC 1 acc.
- QEMU style 2 acc.
- SystemC 2 acc.
- SystemC skel. (1 acc.)
- SystemC skel. (2 acc.)

Penalty ~ 100%!!!
QBox Test System

Penalty ~ 100%!!!

QEMU Wrapper

MPEG-2 decoder

Linux

Driver

QEMU

GreenRouter

SystemC iDCT acc.

IRQ

Reg. File

irq_event

TLM Socket

SignalSocket (IRQ)
Complex QBox system
Conclusions – SystemC

- Joined SystemC to two Virtual Platforms
- Tested two different strategies for joining two simulators
  - QEMU-SC
  - QBox
- Minor performance impact SystemC bridge
- Published as open-sourced projects
  www.greensocs.com
## Conclusions – SystemC

<table>
<thead>
<tr>
<th></th>
<th>Simulation Manager</th>
<th>Penalty</th>
<th>System for</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEMU-SC</td>
<td>Yes</td>
<td>10%</td>
<td>SW</td>
</tr>
<tr>
<td>QBox</td>
<td>No</td>
<td>25~30%</td>
<td>HW</td>
</tr>
</tbody>
</table>
Future Work

➢ Automagic configuration of QBox systems
  • Manage map-address in QEMU, Router, BIOS, etc.
➢ Enhance QEMU time management
  • Hard to measure virtual time in QEMU
➢ Add multiple instances from QEMU
  • Current QBox library allows one
➢ Explore QEMU user mode
  • Simplified version, only ISS, run applications
Future Work

➢ Merge SystemC methods into QEMU kernel
  • Remove OSCI simulator and write an API
    SystemC <-> QEMU
➢ Merge QEMU functionality into OSCI kernel
  • Make QEMU a truly SystemC ISS
➢ Both merges increase simulation speed due to removed synchronization
Proposal

➢ Join efforts and teams to develop
  • just one QEMU & SystemC virtual platform
Thank you!

Questions?