Timing Aspects in QEMU/SystemC Synchronization

Davide Quaglia[1], Franco Fummi[1] Maurizio Macrina[2], and Saul Saggin[2]

[1] University of Verona, 37134, Italy
[2] EDALab s.r.l., 37134, Italy

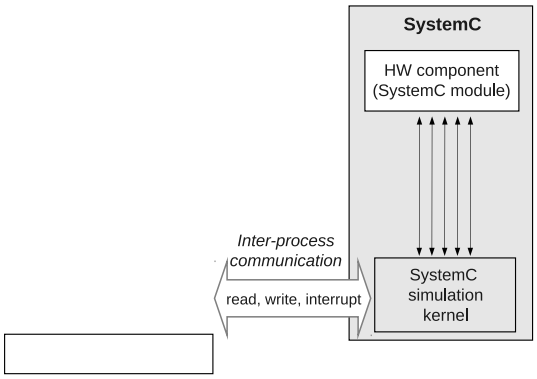
Abstract. Modeling complex embedded platforms requires to co-simulate one or more CPUs, connected to hardware devices, running applications on top of an operating system. This work presents a QEMU-based co-simulation framework in which hardware devices can be either mapped on the actual hardware of the co-simulation host (e.g., an Ethernet chip) or modeled in SystemC, when they represent ad-hoc IP-cores under design. The work presents the timing aspects to be addressed to synchronize QEMU, executing software components, and a SystemC simulation kernel.

1 Introduction

In the design of ever complex embedded systems, a major task is handling several platforms consisting of different processors and operating systems as well as a large amount of HW devices such as memory, DSPs, and I/O interfaces. CPU emulation, as the one provided by QEMU [1], can be used to reproduce the behavior of target processors while HW description languages and their simulation environments such as SystemC [2] can be used for the simulation HW devices under design. The reasons of the choice are: *1)* QEMU already supports the use of host-mapped devices *2)* SystemC supports the HW description at many abstraction levels, and *3)* QEMU source code is available and easy to understand and modify.

Figure 1 shows the framework described in this work. QEMU hosts both the user application and the operating system; they access the hardware components by using device drivers and interrupt service routines which read and write device registers through the memory-mapped I/O. The SystemC HW registers are mapped on specific memory locations and read/write operations on this locations are re-directed to SystemC. Also interrupt signals generated by SystemC models are sent to QEMU.

Communication between QEMU and SystemC simulator is established through inter-process communication primitives (i.e., a socket) between a new component of QEMU, named *QEMU-SystemC Wrapper*, and a modified version of the SystemC simulation kernel; the exchanged messages have the purpose not only to transmit data and interrupt signals but also to keep the simulation time synchronized between the simulation kernels.


SystemC
HW component
(SystemC module)
Inter-process
communication
read, write, interrupt
SystemC
simulation
kernel

```
while( T < Tmax )
{
  read messages coming from the other peer
  and enqueue them (together with local events)

  pick all the next events from the queue
  with the same timestamp and execute them (for
  each executed event which is old, issue
  a warning notification to the user)

  for each executed event for which a message has
  been sent to the peer, wait for its response

  update T to the timestamp of the executed event
}
```

Fig. 2. Pseudo-code of the synchronization algorithm (Approach 1).

alignment of simulation times but it requires that just one tool is running at each time. The next approaches allow concurrent execution at the cost of more complex algorithms; therefore, they can fully exploit multi-core architectures.

2.2 Approach 1

Figure 2 shows the pseudo-code of the first approach of the synchronization algorithm; it is implemented both in the SystemC kernel and in the QEMU Wrapper. Both instances of the algorithm start by reading messages coming from the other peer tool. The corresponding events are put in a time-sorted queue. In SystemC kernel this queue also contains internally-generated events either involving internal modules or requiring to send a message to the peer. Then, the peers execute all the queued events belonging to the same time. An event to be executed could be old if it comes from the other peer whose simulation time is lower. In this case, its timestamp is updated to the current time before execution and a warning notification is issued to the designer since simulation output may not be correct. Finally, both tools wait for an acknowledge for each request of a read or write operation and increase the simulation time. In case of read command, the acknowledge also bears the requested value. If a tool receives a request with a lower simulation time, it sends immediately the acknowledge. If the request has a higher simulation time, the receiver serves it when the same simulation time is reached; since the acknowledge is delayed the tool which made the request is forced to wait thus leading to its realignment.

2.3 Approach 2

The second approach is based on the first one but introduces a further mechanism to re-align simulation times. If a tool has a higher simulation time than the one

of the received command, it sends immediately the acknowledge and issues a warning as in Approach 1; then it blocks itself after having asked the requester to be notified when it reaches the same simulation time.

2.4 Approach 3

To use this synchronization approach each tool has to know the timing of future events involving data exchange with the other tool. This approach is based on the definition of some synchronization checkpoints. The first checkpoint is fixed in the setup phase, tools exchange the time of the next event involving co-simulation, i.e., read/write/interrupt operations. A special message (called NEXT in the following) is used. The tool receiving a NEXT message records the time of the other-peer next event in its event queue.

After this setup phase the tools start execution till the first checkpoint. At the checkpoint, the enqueued NEXT message blocks the receiver waiting for the read/write message of the other tool. The other tool sends the corresponding message, computes the next co-simulation event, sends a new NEXT message and waits for a NEXT message coming from the co-simulation peer to update its queue with the new checkpoint. On the other side, after handling the event, the tool waits for the NEXT event message of the peer, computes its next co-simulation event and sends a NEXT event message.

Once tools have updated their event queue with the new NEXT messages, the next synchronization point is defined and both events executed till the next checkpoint.

In case of QEMU, the evaluation of the next co-simulation event requires the analysis of the code to be executed looking for instructions involving read/write operations on shared locations. The presence of the cache of the already translated code simplify this task. In case of SystemC, only the next event in the queue is known and it could not be a co-simulation event. For this reason, a *virtual clock* has been introduced to generate periodic synchronization events inside SystemC simulator; if the next event is not a co-simulation event, the virtual synchronization event is used. The virtual clock frequency is set by the user in SystemC at the beginning of the co-simulation session; the higher is the frequency, the slower is the simulation but a too low frequency could lead to misalignment of the simulation times. The right trade-off depends on the simulation model.

References

1. QEMU Emulator, http://fabrice.bellard.free.fr/qemu.
2. "IEEE Std 1666 - 2005 IEEE Standard SystemC Language Reference Manual," *IEEE Std 1666-2005*, pp. 1–423, 2006.