QEMU/SYSTEMC COSIMULATION AT DIFFERENT ABSTRACTION LEVELS

Markus Becker
University of Paderborn/C-LAB
Fuerstenallee 11, 33102 Paderborn
markus.becker@c-lab.de

KEYWORDS

ABSTRACT
As today’s embedded software applications have a fast growing complexity early modeling and verification is crucial to save design costs and time-to-market as well as to increase the software quality. This is especially applies to safety critical real-time properties which require the use of dedicated Real-Time Operating Systems (RTOS). Modeling such systems is always a trade-off between simulation accuracy and simulation performance.

We present a mixed QEMU- and SystemC-based cosimulation environment for embedded real-time software [BZM10, MOZB10, BMX+10]. The environment comprises different levels of real-time software abstraction starting from a purely native source-level model for very early and fast simulations which can be refined towards an accurate but yet very efficient CPU specific model (see Fig.1). Modeling of timing aspects starts from coarse-grained source-level software segments with statically analyzed delay annotations towards dynamic cycle approximate estimations of the CPU specific binary code.

For this, our abstract Real-Time Operating System model in SystemC (aRTOS) [ZMG09][BZMU09] is combined with QEMU software emulator. For software refinement, we apply the two QEMU emulation modes. The QEMU system mode emulates a full CPU and its I/O for the execution of complete software stacks including operating system and drivers. In contrast, QEMU user mode emulates a CPU in user mode for the execution of single user programs on top of the host system. Existing QEMU/SystemC cosimulation environments are usually interfaced through the register accurate I/O emulation of QEMU’s system mode. To provide a smoother refinement from the purely native model towards the full system emulation, we introduce an intermediate step interfacing QEMU and SystemC at the RTOS API. For this, system calls from the QEMU user mode are trapped and handled by the native SystemC RTOS model. Thus, the execution of the actual RTOS and device drivers can be avoided postponing the decision for a concrete RTOS implementation by the additional advantage of a very fast cosimulation at the same time. We outline the efficiency of our approach by first results comparing simulation speed with accuracy.
REFERENCES


