A Heterogeneous Simulation and Modeling Framework for Automation Systems

Dogan Fennibay, *Member, IEEE,* Arda Yurdakul, *Member, IEEE,* and Alper Sen, *Senior Member, IEEE*

***Abstract*—Recently, new technologies have emerged in industrial automation platforms. A rapid modeling and simulation environment is required to integrate these new technologies with existing devices and platforms to reduce the design effort and time to market. System-level modeling is a popular design technique that provides early simulation, verification, and architectural exploration. However, integration of real devices with system models is quite challenging due to synchronization and hard real-time constraints in industrial automation. SystemC is the most commonly used system-level language in hardware–software codesign. However, SystemC lacks interfaces for the integration of system (virtual) models with real (physical) devices. We introduce the hybrid channel concept to clearly define the integration interface. Hybrid channel incorporates both real-to-virtual and virtual-to-real communication functions by solving synchronization issues while satisfying the real-time constraints. We successfully demonstrated the usability of our framework in industrial systems that utilize BACNet and Ethernet. We also developed a mathematical model that correctly estimates the results of our experiments. To the best of our knowledge, this is the first framework and mathematical model for SystemC in industrial automation domain.**

***Index Terms*—Hardware in the loop, real-time communication, real-time embedded systems, SystemC.**

I. Introduction

INDUSTRIAL automation systems are experiencing a paradigm shift due to incorporation of new technologies, such as embedded real-time devices and communication networks. It has been shown that the demand for plug and play mechatronic solutions increases significantly and the investments on automation systems that utilize hardware–software and communication infrastructures cannot be ignored in today's factory systems [1]. As the complexity of these systems increase, a rapid modeling and simulation environment is required to reduce the design and verification time. System-level modeling is a very effective way of reducing the development cycle while providing early prototypes and enabling architectural exploration and verification.

System-level modeling also allows reuse of different forms of intellectual properties (IPs), which is a common practice in the industry. Since industrial automation systems are increasingly connected with other IPs or industrial components, they should also benefit from system-level modeling techniques to reduce the development costs.

In traditional system-level modeling, all components need to be modeled. However, modeling has no added value for components that are already physically implemented because modeling is an abstraction mechanism and requires human effort. Therefore, techniques have to be developed for incorporating real devices with virtual models. Traditionally, there are communication mechanisms between virtual models. These mechanisms are defined through the constructs of modeling languages. Similarly, real implemented devices communicate with each other through physical mediums with predefined exchange data formats. However, there is no established mechanism that provides communication between real devices and virtual models (given in SystemC), and this paper fills this gap. Specifically, we present synchronization mechanisms between virtual models and real devices so as to achieve real-time communication.

We use SystemC, which is an IEEE standard (1666–2005) [2], for the system-level modeling of the industrial automation systems. SystemC allows modeling of complex systems at various abstraction levels. Therefore, it is one of the most commonly used modeling language in hardware–software codesign. However, SystemC does not have a library to connect real devices to virtual models. In this paper, we develop a hybrid channel mechanism for the design of industrial automation systems that incorporates both real devices and virtual models. We describe our contributions as follows.

1) We devise a coherent way of matching simulation time with real-time execution. This is because in heterogeneous simulation, the simulation time of virtual models does not match execution of real devices.
2) We develop mechanisms for connecting concurrent real devices to a set of virtual models. This is a requirement for the virtual model if it is driving multiple real devices in parallel. However, SystemC kernel operates sequentially and it cannot generate concurrent outputs from the virtual models.
3) We define techniques that allow virtual models to receive inputs from real devices. This is due to the fact that

Manuscript received December 1, 2011; revised February 13, 2012 and April 13, 2012; accepted April 17, 2012. Date of current version October 19, 2012. This work was supported in part by the Marie Curie European Reintegration Grant within the 7th European Community Framework Program, in part by the Bogazici University Research Fund 5483, and in part by the Turkish Academy of Sciences. This paper was recommended by Associate Editor P. Eles.

D. Fennibay is with Siemens AS, Istanbul 34870, Turkey (e-mail: dogan.fennibay@siemens.com).

A. Yurdakul and A. Sen are with the Department of Computer Engineering, Bogazici University, Istanbul 34342, Turkey (e-mail: yurdakul@boun.edu.tr; alper.sen@boun.edu.tr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2012.2199116

current SystemC libraries do not allow receiving inputs from external devices but only from virtual models.

4) We develop a mathematical model to estimate the validity of our heterogeneous framework.
5) We develop an experimental framework that utilizes hybrid channels in SystemC. We validate the effectiveness of our framework in industrial systems that employ BACNet and Ethernet.

The communication rate over hybrid channels can be as high as 10 kHz, that is enough to satisfy hard real-time constraints in industrial automation systems, as specified in [1].

The organization of this paper is as follows. Section II provides an overview of hardware–software codesign, which is followed by Section III covering related work. Section IV explains our solution and Section V gives information on the experimental evaluation of the proposed solution. Finally, we conclude by discussing the obtained results and listing the future work in Sections VI and VII.

II. Background

A. SystemC

SystemC is a language developed for system-level modeling (virtual models) used mostly for system-on-chip (SoC) systems. SystemC offers a clear set of modeling mechanism at varying abstraction levels.

SystemC has the concept of *processes* to model the concurrent activities of a system. Processes can be combined into *modules* to create hierarchies. *Events* (described in sc_event class) are the basic means of synchronization between processes. Events provide the *wait* and *notify* methods. Processes are the main concurrent execution units in SystemC. They have a notion of *sensitivity* to events, which can trigger for execution of processes. *Channels* are the formal means of communication between modules. Employing other means for intermodule communication harms the reusability of the model. Communication structures that can be modeled with a channel range from a very simple mutual exclusion mechanism to a very complex hierarchical communication structure such as a peripheral component interconnect bus. *Interfaces* provide a powerful ability to have interchangeable channels. *Ports* are connection points of modules to channels.

SystemC simulation kernel is a discrete event simulator [3]. SystemC kernel has a nonpreemptive, nondeterministic scheduler without a notion of priority for processes. A scheduled process is executed until it willingly gives up the execution resource via a *wait* method or it terminates (nonpreemptive). Among active processes, one of them is chosen for execution nondeterministically.

SystemC kernel execution consists of four main phases: initialize, evaluate, update, and time advance, as shown in Fig. 1. Evaluate and update phases form a delta cycle. Only an infinitesimal amount of time is assumed to pass in a delta cycle, so it is a “zero time advance” cycle. The changes of the evaluate phase operations on channels are not updated until the update phase. This mechanism allows the simulation kernel to model concurrent operations similar to hardware description language simulators.

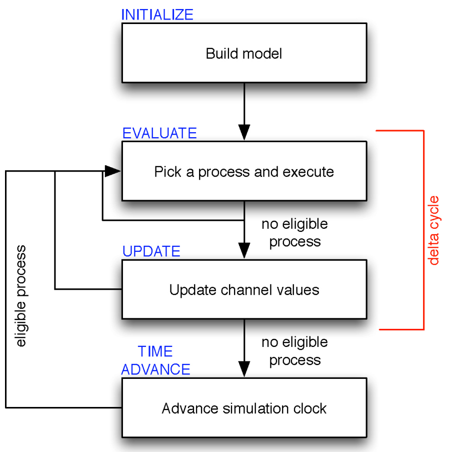

Fig. 1. SystemC kernel scheduler [3].

B. Real-Time Simulation

In a discrete-event simulator, the *simulation clock* is advanced in discrete-time intervals. At each interval, the simulator executes the processes that are triggered by the events that are scheduled. As soon as the execution of processes is finished, the simulation clock is advanced. Hence, the actual duration of two subsequent time intervals need not be the same. However, in real-time simulation, the time intervals must be physically the same (e.g., 1 ms). The clock for real-time simulation is defined as the *wall clock*. In order to have a system with real-time behavior, we need to establish the relationship given in (1) between the simulation and wall clocks, where $t_{S\text{new}}$ ($t_{W\text{new}}$) and t_S (t_W) denote the current and the starting values of the simulation clock (wall clock), respectively, [4] as follows:

$$\frac{t_{S\text{new}} - t_S}{t_{W\text{new}} - t_W} = 1. \tag{1}$$

III. Related Work

A. Hardware-in-the-Loop Techniques

There are several hardware-in-the-loop (HiL) solutions for different domains. We mention some of these techniques as they relate to our paper.

Both industry and academia propose solutions for providing HiL simulation in many domains, including aerospace and automotive [5]–[8], such as for engine controller units, antilock brakes, or space robot testing and validation. dSPACE [9] has a widespread reputation in these domains. The first two platforms use MATLAB/Simulink [10] or Modelica/Dymola [11], [12], which are known as powerful languages in continuous-time modeling of physical systems. However, these tools have engines and libraries to support discrete-event simulation [13]–[16].

In HiL solutions such as xPC Target [17] and Veristand [18], the model is executed on a dedicated system or on a Windows system, respectively. However, the modeling languages provided by these solutions have not been designed for hardware–software codesign purposes or for discrete-event simulation, so they lack the necessary constructs and mechanisms that are already present in SystemC. Our paper is orthogonal to such existing methods, as it is proposing to introduce the technique to the field of hardware–software codesign with a much more powerful modeling language, namely, SystemC.

SystemC simulation kernel is a powerful discrete-event simulator for hardware–software codesign. However, a HiL solution with SystemC has not been proposed before our paper. In the literature, programming temporally integrated distributed embedded systems (PTIDES) [19] has been proposed to fill the gap of discrete-event-based HiLs, but Ptolemy II, which is the modeling language used by PTIDES, has not been as widely adopted by the hardware–software codesign community as SystemC has been. Also, PTIDES uses a real-time operating system (RTOS), named PTinyOS, whereas our solution is implemented on a general-purpose operating system (GPOS) (i.e., Linux) with a real-time patch (i.e., RT-PREEMPT). PTIDES only uses an event-driven method to handle external events arriving from real devices. Our SystemC-based solution incorporates not only event-driven but also nonadaptive and adaptive polling methods to provide a better cosimulation environment. We experimentally show that for some cases, nonadaptive and adaptive polling methods can perform better than the event-driven method. Also, there are discrete-event simulators to simulate register transfer-level (RTL) models with SystemC models [20], both of which are virtual models. Balarin *et al.* [20] used property specification language, formal language, for describing transactors between different models. Transactors can be generated for connecting SystemC and RTL models as well as supporting standard co-emulation modeling interface protocol. However, there is no support for connecting actual real hardware components to virtual models, whereas we support this type of connection.

B. Frameworks for Industrial Automation

In the last decade, there have been significant studies to develop design platforms for intelligent industrial automation systems [21]. These automation systems rely heavily on a distributed computer-based infrastructure, where smart sensors and actuators, intelligent machines, robots, and other automation devices can interact using industrial protocols and take decisions in real time. In these systems, system-level communication, device synchronization, and the integration of new devices to the system are extremely challenging issues. Hence, there is a demand for sophisticated tools for the design of intelligent complex industrial automation systems.

In SIMOO-RT [22], an object-oriented framework is proposed for modeling a real-time industrial automation system. In this approach, objects are generated in Active-Objects/C++. It utilizes RT-UNIX operating systems (OSs) such as QNX. A system model with real devices cannot be developed with SIMOO-RT.

OOONEIDA [23] complies with IEC 61449 [24], which is a standard to design distributed control systems with intelligent devices. It proposes encapsulation of different types of IPs into reusable portable software modules called as function blocks. To achieve this, interfaces are created between various kinds of automation IPs such as devices, RTOSes, machines, systems, and industrial enterprises. However, in IEC 61499, the real-time properties of applications (e.g., reaction time) and resources (e.g., polling of data by function blocks, communication) are unspecified [25]. In OOONEIDA, real-time properties have to be handled via embedded controllers that are introduced to the system by encapsulation. There has also been a proposal on handling real-time issues in function blocks [25], but it has not been implemented yet. The encapsulation modules are developed in Java for the Eclipse and NetBeans integrated development environments.

In RI-MACS [26] project, a service-oriented architecture with real-time capabilities is proposed. Temporal behavior of each activity is isolated as much as possible by using dedicated hardware and software [1]. Linux kernel is modified to provide temporal isolation.

Our solution aims to design the entire real-time complex industrial automation systems with existing real devices and nonexisting-but-will-exist devices (i.e., virtual models). SystemC allows abstract modeling hence giving the designer flexibility in terms of developing various models. In HiL solutions, the virtual model does not necessarily need to be implemented; it can purely be used for testing or validating the already available real devices or can be used for developing newer generations of devices and testing them before they are produced. Even in the case of SoC design, it is a common practice to use SystemC for virtual model development and the actual hardware does not actually get produced directly through synthesis from this SystemC model.

C. Integrating Different Virtual Platforms

As there are no real devices involved, there is no need for real-time behavior, but executions of virtual environments have to be synchronized.

HetSC [27] integrates multiple models of computation in SystemC. It does not handle integration of a SystemC model with external environments. The work in [28] proposed a method for the component-oriented interoperation of real-time discrete-event system specification (DEVS) engines. DEVS is a model of computation that has its implementations in languages such as ECD++ and SystemC [29].

The work in [30] aimed to integrate quick emulator emulation environment and SystemC. It employed a SystemC module for representing the communication. An additional channel was necessary to connect the integration module to the rest of the model. However, this approach suffered from unnecessary doubling of effort, where the integration could be directly implemented with a hierarchical SystemC channel.

D. Integrating Real and Virtual Environments

A major decision point in the integration of real devices and virtual models is the level of abstraction. The communication between real devices and virtual models can range from pin level [31]–[33] to transaction level [30], [34]–[37]. Pin-level approaches offer great flexibility, as any communication protocol can be modeled on top of pin-level when necessary. However, with pin-level approach, modeling the communication protocol is costly and this also degrades accuracy and the execution speed of the whole model. Approaches that prefer transaction-level communication between real devices and virtual models allow to skip the modeling of the details of the communication protocol and focus on other issues such as emulation or bus modeling.

We first introduced the hybrid channel concept for hardware–software codesign in [38]. In that paper, we developed a heterogeneous system with a single real device. The input from the real device could be obtained via polling, where polling period had to be manually tuned. Since that solution was not able to efficiently handle complex industrial automation systems, we extended it in many directions in this paper. Specifically, we introduce techniques to drive multiple concurrent outputs from virtual models to real devices. We enable two automated mechanisms for receiving inputs from real devices: adaptive polling and event-driven solution. We provide a mathematical model to estimate the performance and scalability of the systems designed with our modeling framework. Also, we present experimental results with industrial automation systems that show the effectiveness of our approach.

E. Timing Concerns

Virtual chip [31] has a contribution in the timing management between real devices and virtual models. The device consists of the internal interface module (IIM), the operational buffer unit (OBU), and the external interface module (EIM). Behaviors of IIM and EIM are synchronized with the virtual model and the real device, respectively. OBU connects both interface modules and handles the timing difference via buffering methods.

Realtimify [39] provides real-time behavior for SystemC models. Basically, a module is added to the model to synchronize the virtual simulation to real-time with the objective to monitor the execution in real time. This approach is very lean and satisfactory for observing the model's execution in real time. However, it does not address the determinism issues faced during interaction with real devices. Additionally, the approach is intrusive as it requires changes in the model. It also relies on the nondeterministic SystemC scheduler. This results in uncertainty about when the model will be synchronized.

F. Deterministic Behavior

OS plays a critical role in obtaining deterministic behavior. RTOSs specialize in providing deterministic behavior [40], but they lack the variety of applications, I/O interfaces, and functionality provided by a GPOS. Linux with real-time improvements seems as a promising tradeoff. Real-time application interface [41], which is used in [42], is built on top of adaptive domain environment for OSs [43] and does time sharing with a Linux kernel. It provides real-time behavior by itself while leaving the resources to the Linux kernel for noncritical tasks. On the other hand RT_PREEMPT [44] employs a more direct method in which latency is decreased by increasing preemptibility throughout the Linux kernel.

IV. Hybrid Channels

Fig. 2 shows our hybrid channel solution. We define modeling platform as the combination of the simulation kernel, OS, and the computer hardware. Our solution both achieves real-time behavior and integrates real devices (real subsystems) and virtual models (virtual subsystems).

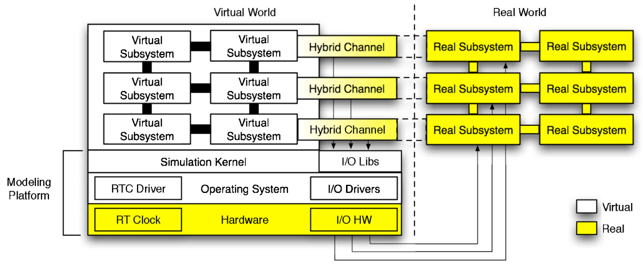

Fig. 2. Architecture of our hybrid channel solution.

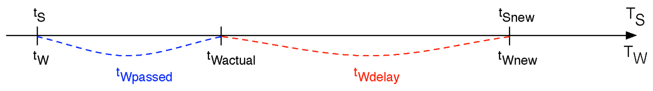

Fig. 3. Real-time visualization and patch to SystemC simulation kernel.

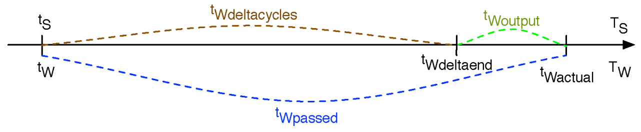

Fig. 4. Detail of Fig. 3 for the real-time duration $t_{W\text{passed}}$.

A. Achieving Real-Time Behavior

Fig. 3 details a single evaluate, update, and time advance phase of the SystemC simulation kernel in Fig. 1. In Fig. 3, the time increases from left to right and we use the real-time as the scale. Assume that at the end of the time advance phase, the simulation clock will be advanced from t_S to $t_{S\text{new}}$ by the SystemC kernel. Also assume that at the end of the time advance phase, the wall clock has advanced to $t_{W\text{actual}}$ due to the delta-cycle processing time $t_{W\text{passed}}$. In order to satisfy (1), we delay the execution of the virtual model simulation by $t_{W\text{delay}}$, thereby advancing the wall clock to $t_{W\text{new}}$. Note that this is possible when the delta-cycle processing time is less than the amount of time that will be advanced to in the virtual model (virtual model should execute at least as fast as the real-time device). Fig. 4 gives a detail view of the delta cycle in Fig. 3.

To reduce latency of the integrated system, we improve the preemptibility of the underlying OS with patches. We also disable further sources of latency, such as swap memory and power management. Additionally, the priority of threads executing the virtual model is increased to guarantee availability of resources.

B. Integrating Real Devices and Virtual Models

We extend SystemC channel concept to realize the hybrid channel functionality. We categorize the channel types and devise the class hierarchy shown in Fig. 5. dsc_hybrid_channel class at the base of the hierarchy distinguishes hybrid channels from other channel types in SystemC and implements the update real functionality, which will be explained in

TABLE I
COMPARISON OF TIMING ALTERNATIVES FOR MODEL OUTPUTS

Phase	Advantages	Disadvantages
Evaluate	Data do not wait at all. Glitches occurring at the end of delta cycles can be relayed to real devices immediately.	Data must not change in subsequent cycles, e.g., sc_fifo. Delta-cycle processing time increases.
Update	The final data from concurrent processes are used. Glitches occurring at the end of delta cycles can be relayed to real devices.	Data wait until the update phase. Delta-cycle processing time increases. Concurrent outputs are distributed over a larger window in real time.
Update_real (just before the time advance)	The final data from concurrent processes are used. Fewer output values are used. Concurrent outputs calculated by delta cycles are gathered together in real time.	Data wait until the time advance phase. Glitches occurring at the end of delta cycles are not relayed to real devices.

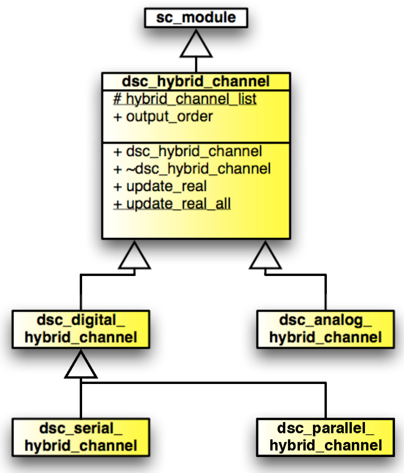

Fig. 5. Class diagram of hybrid channels.

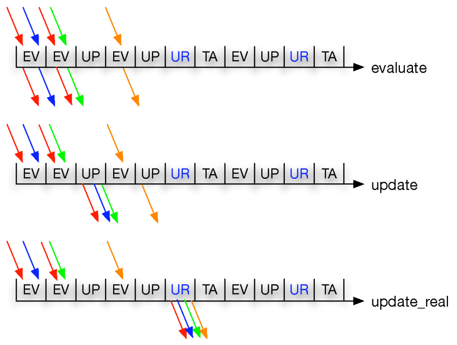
EV EV UP EV UP UR TA EV UP UR TA
evaluate
EV EV UP EV UP UR TA EV UP UR TA
update
EV EV UP EV UP UR TA EV UP UR TA
update_real

Fig. 6. Example set of outputs from virtual models to real devices using three different output methods.

the next section. In SystemC, a channel can inherit from sc_prim_channel (primitive channel) or sc_module (for hierarchical channels). As hierarchical channels offer a superset of primitive channel capabilities [3], we chose sc_module as the base of dsc_hybrid_channel. A hybrid channel can carry digital or analog data. This choice can be specified by choosing the appropriate subclass dsc_digital_hybrid_channel or dsc_analog_hybrid_channel. Digital channels can transfer data in a parallel way or in a serial way. Hence, two further subclasses are provided for specifying this characteristic.

1) *Interactions from Virtual Models to Real Devices:* Output values generated in the virtual models can be transferred to the real devices in three phases of the SystemC kernel: evaluate, update, or just before the time advance phase as shown in Fig. 1 and Table I. The constraints of the channel model dictates the best phase for the transfer as we describe below.

a) *Evaluate*: The data can be transferred as soon as it is produced. For example, a first in first out (FIFO) channel whose current value will not be affected by values in later delta cycles can be transferred in the evaluate phase.

b) *Update*: There might be several processes that affect the final value of an output variable. In that case, the data should not be transferred to the real device until the final stable value is reached. Otherwise, if multiple successive delta cycles change the data in the channel, real devices can observe this. For example, a signal channel whose actual value is established at the end of a delta cycle can be transferred in the update phase.

c) *Update_real (just before the time advance)*: Due to the sequential nature of SystemC simulation kernel (it merely simulates concurrency), concurrent outputs cannot be transferred to the real devices simultaneously. When output values are transferred in the evaluate or update phases, real devices observe them at time points distributed in $t_{W\text{delta cycles}}$ time as shown in Fig. 4. Delaying the transfer until the time advance phase will gather the outputs in $t_{W\text{output}}$ time, which is much smaller than $t_{W\text{delta cycles}}$. So, outputs that are simultaneous with reference to the simulation clock are generated in a smaller time window. This option has another advantage of reducing simulator effort because the number of I/O operations are reduced. As a disadvantage, delta-cycle changes are not observable by real devices in this scheme. However, delta-cycle changes need not always be relayed to the real devices. In other words, a real system that has to be insensitive to the transient behavior of the signals at its inputs benefits from this method.

Fig. 6 shows an example of using the above-mentioned three different phases for transferring outputs from virtual models to real devices for a simulation that involves multiple time advance phases. We assume that the output values are generated only in the evaluate (EV) phase. Outputs can be transferred in evaluate (EV), update (UP), or update real (UR) phases. The time advance (TA) phase always pairs with exactly one UR phase, and is solely used for advancing the time. The arrows on the top of the timelines denote the output-event-generation phases in the virtual model and the arrows below the timeline denote the phases where such outputs are transferred to the real devices. Arrow colors denote for which device an output is generated. Note that the number of arrows at the top and the bottom can differ as outputs sent to the same real device can be combined together.

SystemC offers methods for transferring the output values in evaluate or update phases. Our work further provides the method update real that is called at the time advance phase prior to the advance of the simulation clock. The model developer can choose the appropriate output timing for the hybrid channel.

If the hardware used by the hybrid channel is capable of receiving or producing concurrent I/O from or to the virtual models or real devices (e.g., a digital I/O device with multiple output channels), then update_real mechanism can be easily exploited to achieve accurate real-time and concurrent behavior.

If there is no hardware support, then ordering has to be applied to the hybrid channels so as to reduce the transfer delay between subsequent channels. We define the set of concurrent outputs as the critical output subset. There can be more than one critical output subset. The hybrid channels in a critical output subset have to have successive order numbers. Additionally, all pending outputs have to be sorted just before $t_{W\text{delta end}}$ in update_real.

2) *Interactions from Real Devices to Virtual Models:* SystemC kernel does not have a mechanism for receiving external events such as the ones generated by a real device. Time is always advanced according to internal events and operations of the kernel. A mechanism is needed to incorporate external events. We solve this by implementing three mechanisms: a) nonadaptive polling; b) adaptive polling; and c) event-driven simulation.

a) *Nonadaptive polling*: SystemC thread processes are used in order to poll the external inputs and relay this information to internal sc_events. SystemC kernel becomes aware of the external events and the rest of the model can use sc_event to check for inputs. Fig. 7 shows an example of this mechanism. The actual input receipt is done asynchronously by an OS thread, whereas it could also be done by the SystemC poll thread synchronously. The polling period (poll_period) is a fixed parameter in this mechanism. The input latency introduced by polling will range in [0; poll_period] and on the average, it will be poll_period/2. Therefore, increasing the poll_period will increase the input latency, and decreasing poll_period will increase the performance demands on virtual model simulation.

b) *Adaptive polling*: If the interval between external events is changing during the model execution, the poll_period can be changed dynamically to adapt to these changes. For instance, the model of a network device may need to adapt its poll_period accordingly when the network traffic increases or decreases. Adaptive polling also helps in the case where the model developer does not have an *a priori* knowledge on the environment of the model and the correct poll_period is unknown at the time of the development. A control loop employing a proportional, integral, and derivative (PID) controller is used to accommodate for these changes. poll_period is the control variable and input_interval is measured independently by the unit handling the inputs. The control loop tries to converge the difference between the input_interval and the poll_period to 0. To achieve this, the poll_period is increased or decreased. The coefficients of the PID controller (proportional K_P, integral K_I, and derivative K_D) are parameterizable. The assignment of these coefficients is a standard PID controller tuning task and the same techniques can be applied here.

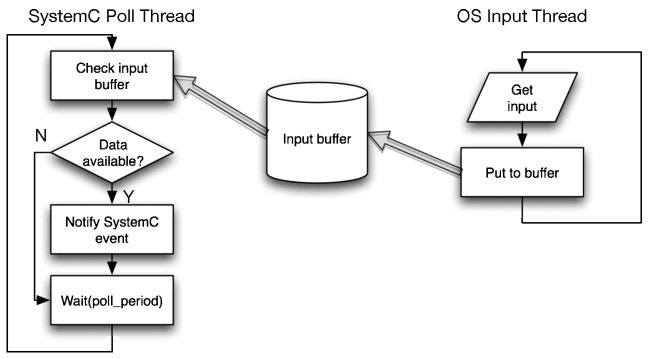

Fig. 7. Nonadaptive polling mechanism for incorporating external events from real devices to virtual models.

c) *Event-driven simulation*: The real-time patch to the SystemC kernel (Section IV-A) can be extended to eliminate the need for polling to incorporate external events as shown in Fig. 8. The changes are added after the completion of delta cycles and before the time advance phase, hence the first box in the figure. The unconditional waiting of $t_{W\text{delay}}$ in Fig. 3 is replaced by a conditional waiting on ext_event, shown as wait(ext_event, $t_{W\text{delay}}$) in the figure. An sc_event is used to represent each external event. This sc_event is added to the external event list and the ext_event is signaled (notified) by the OS input thread (input handler). In case there is no timeout, then the SystemC scheduler waiting on ext_event wakes up, adjusts the SystemC simulation clock according to the current wall clock time, and schedules timed notifications for all the external events sc_events in the external event list including the external event that triggered the wait condition. These timed notifications will trigger the threads responsible for processing external inputs in the SystemC virtual model. If the OS input thread does not get an input, hence no ext_event, the *wait* will end due to timeout and the execution will proceed to the next delta cycle.

Table II shows a comparison of methods for incorporating external events to the simulation model.

C. Mathematical Model of Execution Performance

We define the following variables for modeling the real-time patch of our SystemC simulation kernel according to Fig. 1.

1) A: Total number of time advance phases in the simulation.
2) U_i: Total number of update phases before the ith time advance phase.

TABLE II
COMPARISON OF METHODS FOR INCORPORATING EXTERNAL EVENTS FROM REAL DEVICES TO VIRTUAL MODELS

Mechanism	Advantages	Disadvantages
Nonadaptive polling	Simple implementation. Fastest execution. No complex OS constructs.	Tuning necessary for poll_rate. Tradeoff: I/O latency versus simulation performance.
Adaptive polling	No complex OS constructs.	Tuning necessary for PID coefficients.
Event-driven simulation	No tuning necessary.	Intricate implementation, uses complex OS constructs.

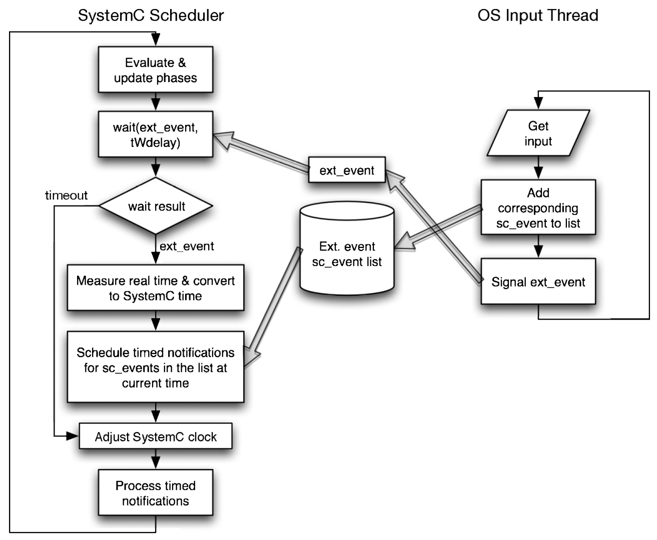

Fig. 8. Event-driven mechanism for incorporating external events from real devices to virtual models.

3) E_{ij}: Total number of evaluate phases before the jth update phase, which comes before the ith time advance phase.
4) O_i: Total number of outputs to real devices before the ith time advance phase.
5) H: Total number of hybrid channels.
6) P: Total number of SystemC thread processes.

The SystemC kernel scheduler, hence the simulation of the virtual model, should satisfy the following constraints to achieve real-time behavior.

Constraint 1) The relationship in (1) between the simulation clock T_S and the wall clock T_W has to be maintained.

Constraint 2) Concurrent output generations with regard to the simulation time T_S have to occur within a maximum output window of $t_{W\text{ow}}$ with regard to the real-time clock T_W. This allows the real devices connected to the virtual model to observe these virtual model outputs as concurrent. This constraint can be further strengthened by adding narrower critical output windows of $t_{W\text{cow}_s}$ for specific subsets of output channels.

1) *Constraint 1: Real-Time Simulation for Hardware–Software Codesign:* Constraint 1 means that $t_{W\text{delay}_i}$ must remain nonnegative since a negative delay is not implementable. This implies that the following inequality must hold in Fig. 3:

$$t_{W\text{passed}_i} \leq t_{S\text{new}_i} - t_{S_i}. \qquad (2)$$

Using Fig. 4, (2) can be further be detailed as follows:

$$t_{W\text{delta cycles}_i} + t_{W\text{output}_i} \leq t_{S\text{new}_i} - t_{S_i}. \qquad (3)$$

Fig. 1 shows that the real-time patch of SystemC kernel scheduler (that takes $t_{W\text{delta cycles}}$ time as per Fig. 3) consists of multiple delta cycles and a delta cycle consists of multiple evaluate phases and exactly one update phase. Equation (4) shows this relationship as follows:

$$t_{W\text{delta cycles}_i} = \sum_{j=1}^{U_i} \left[\left(\sum_{k=1}^{E_{ij}} t_{W\text{evaluate}_{ijk}} \right) + t_{W\text{update}_{ij}} \right] \qquad (4)$$

where $t_{W\text{evaluate}_{ijk}}$ is the sum of runtimes of processes that are active in the evaluate phase k before the update phase j that comes before the time advance phase i. $t_{W\text{update}_{ij}}$ is the sum of channels' update times that were written during the previous evaluate phases. If an upper bound can be guaranteed for $t_{W\text{evaluate}_{ijk}}$, $t_{W\text{update}_{ij}}$, and U_i, then the inequality (2) can be guaranteed statically because the total number of thread processes in the model, P, is always an upper bound for the number of processes executed in an evaluate phase, E_{ij}. However, SystemC is based on C++, so it supports all programming structures of this language, including very complex ones. Furthermore, the inputs from real devices arriving via hybrid channels can change the execution path of threads. Therefore, it is nearly impossible to determine the runtimes statically or guarantee upper bounds for these variables. Common industry practice of profiling, i.e., runtime analysis, should be chosen here. As an addition to profiling for the special case of SystemC, measuring the ratio $t_{W\text{passed}_i}/(t_{S\text{new}_i} - t_{S_i})$ at each time advance phase is used to monitor if the inequality (2) is held.

2) *Constraint 2: Concurrent Outputs:* The generation of an output from the virtual model to the real devices is done in three steps as described below.

a) *Output value computation*: Computation of the data to be output.
b) *Output trigger*: The trigger for starting the output generation.
c) *Output generation*: The actual realization of the output so that the real devices can receive it.

$t_{W\text{output}_i}$ is the sum of output production runtimes of hybrid channels at the ith time advance phase. This can vary a lot for different types of output hardware. The following assumptions are introduced for simplification.

a) All output generations are delegated to asynchronous output threads that employ sufficiently large output buffers to guarantee decoupling from the simulation's

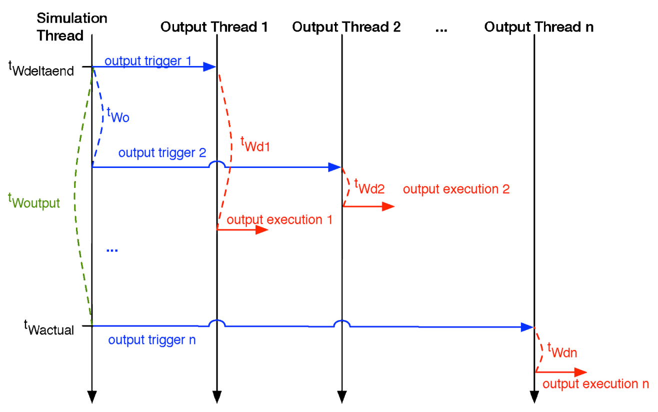

Fig. 9. Concurrent output generations from virtual models to real devices using n output threads in the update real phase for the ith time advance phase.

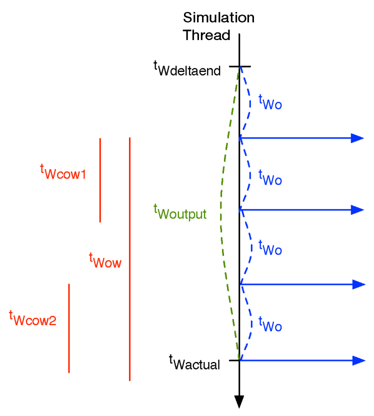

Fig. 10. Concurrent output windows.

generation so that the simulation thread does not wait on output threads. These threads are created in the initialization phase of the simulation in order to avoid latencies caused by the thread creation.

b) The triggers of output threads from the simulation thread is implemented via similar OS mechanisms, e.g., notifying a condition variable.

c) The number of concurrent output productions at a specific simulation time does not exceed the number of processing cores in the modeling platform.

d) All hybrid channels use the update real method for output timing.

Under these assumptions, it becomes safe to claim that all output triggers take the same amount of time, t_{Wo}, in the simulation thread. The total number of hybrid channels H in the model can be used as an upper bound for the number of output generations at a time advance phase. If there are multiple output generations pending on a hybrid channel at a time advance phase, they can be still regarded as one, because: a) the trigger to output generation is still single, and b) multiple output value computations imply a sequential relationship, so the concurrency is not actually required for the output productions of one hybrid channel. We can derive (5) as follows:

$$t_{W\text{output}_i} = \sum_{u=1}^{O_i-1} t_{Wo} = (O_i - 1) \cdot t_{Wo} \leq (H-1) \cdot t_{Wo}. \quad (5)$$

Fig. 9 outlines the generation of outputs using the update real phase, detailing the $t_{W\text{output}}$ in Fig. 4. Output value computation is done in the evaluate and update phases, i.e., before $t_{W\text{delta end}_i}$. Output trigger is done in the simulation thread, which always takes t_{Wo}. Output generation is done in asynchronous output threads and may therefore take various amount of times, denoted as t_{Wd_u} for each output thread u.

Now we can discuss satisfaction of Constraint 2. As Fig. 9 shows, concurrent output triggers are distributed over $t_{W\text{output}_i}$, which should be lower than the output window $t_{W\text{ow}}$ described above. So the constraint can be formulated as follows:

$$t_{W\text{output}_i} \leq t_{W\text{ow}}. \quad (6)$$

Equation (5) can be used to get an upper bound for $t_{W\text{output}_i}$, so (7) can be used as an estimator for (6). Here, H can be derived statically from the model and t_{Wo} can be measured as a characteristic of the modeling platform as follows:

$$(H-1) \cdot t_{Wo} \leq t_{W\text{ow}}. \quad (7)$$

Satisfying a narrower time window of $t_{W\text{cow}_s}$ for a subset s of output channels (see Fig. 10 for an example) is achieved by specifying a fixed order of output generations, where output channels in the subset are ordered successively. This requires that the subsets are exclusive, i.e., do not share an output channel. Thereby, the time between two output generations in the subset is set to t_{Wo}. For a subset s consisting of m_s channels, the constraint can be formulated as follows:

$$(m_s - 1) \cdot t_{Wo} \leq t_{W\text{cow}_s}. \quad (8)$$

As seen in Fig. 9, the real point in time where the output is observable by real devices depends also on the runtime of the output thread t_{Wd_u}. However, this does not imply an inaccuracy of the model from the real system because several forms of delay are also present in a real system as propagation delay, signal setup time, etc. The same fact is also true for input channels.

Another point regarding the output timing is the hold time for channels with parallel semantics (dsc_parallel_hybrid_channel). As new values overwrite old values, there should be a minimum time window for the value to remain unchanged, so that the real devices can get the value. However, this is strongly dependent on the model, and the model developer should and can specify this hold time by inducing a wait in SystemC. Our method will guarantee that the value will be held for at least this duration.

3) *Illustrative Example:* Let there be an embedded system under development where the model is connected to the real devices via six hybrid channels: A, B, C, D, E, and F. A and C are connected to real subsystem 1, B, E, and F are connected to real subsystem 2, and D is connected to real subsystem 3. Fig. 11 shows the setup.

As concurrent outputs from the model to single devices should be produced within a narrow time window, two critical output subsets of the set of all hybrid channels are defined.

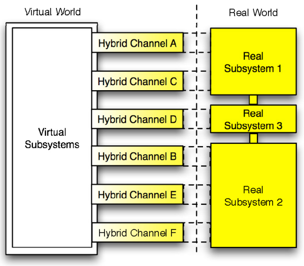

Fig. 11. Setup of illustrative example.

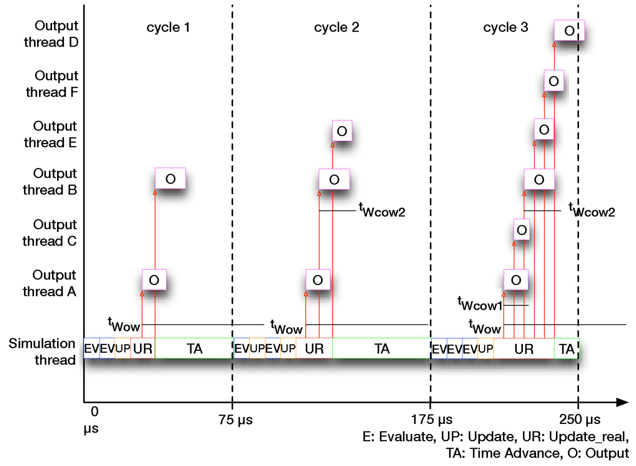

Fig. 12. Sample execution of the illustrative example.

Hybrid channels A and C belong to the critical output subset 1 (with number of channels $m_1 = 2$) and B, E, and F belong to the critical output subset 2 (with number of channels $m_2 = 3$). Critical output subsets 1 and 2 have the critical output windows $t_{W\text{cow}_1} = 10\,\mu\text{s}$ and $t_{W\text{cow}_2} = 15\,\mu\text{s}$, respectively. There is also the general output window $t_{W\text{ow}} = 50\,\mu\text{s}$ for all hybrid channels.

To satisfy the critical output window constraints, a strict ordering of all hybrid channels is forced by assigning them integers for ordering as follows: A: 10, C: 10, B: 20, E: 20, F: 20, and D: 30. All outputs are triggered during the update real phase and the output generation is delegated to asynchronous output threads. Let the time of one output trigger be $t_{Wo} = 5\,\mu\text{s}$.

Fig. 12 shows an excerpt of the execution of the example system. In all cycles, $t_{W\text{passed}_i}$ has remained below $t_{S\text{new}_i} - t_{S_i}$, i.e., in the first cycle, we measured that $t_{W\text{passed}_1} = 36\,\mu\text{s} \le t_{S\text{new}_1} - t_{S_1} = 75\,\mu\text{s}$, in the second cycle, $t_{W\text{passed}_2} = 51\,\mu\text{s} \le t_{S\text{new}_2} - t_{S_2} = 100\,\mu\text{s}$, and in the third cycle, $t_{W\text{passed}_2} = 63\,\mu\text{s} \le t_{S\text{new}_3} - t_{S_3} = 75\,\mu\text{s}$. As a result, the time advance phase can always be executed.

The following observations can be made about the behavior of concurrent outputs.

a) In the first cycle, only one channel from each critical output subset has an output, so critical output windows need not be checked. Only general output window can be checked, and it is satisfied: $(O_1 - 1) \cdot t_{Wo} = 5\,\mu\text{s} \le t_{W\text{ow}} = 50\,\mu\text{s}$.

b) In the second cycle, two channels from the critical output subset 2 have new output values, so the critical output window 2 is checked and it is satisfied too: $(m_2-1)\cdot t_{Wo} = 10\,\mu\text{s} \le t_{W\text{cow}_2} = 15\,\mu\text{s}$. Additionally, general output window is also satisfied: $(O_2 - 1) \cdot t_{Wo} = 5\,\mu\text{s} \le t_{W\text{ow}} = 50\,\mu\text{s}$.

c) In the third and last cycle, all hybrid channels have an output, so both critical output windows are checked and they are satisfied: $(m_1 - 1) \cdot t_{Wo} = 5\,\mu\text{s} \le t_{W\text{cow}_1} = 10\,\mu\text{s}$ and $(m_2 - 1) \cdot t_{Wo} = 10\,\mu\text{s} \le t_{W\text{cow}_2} = 15\,\mu\text{s}$. Additionally, general output window is satisfied too: $(O_3 - 1) \cdot t_{Wo} = 5\,\mu\text{s} \le t_{W\text{ow}} = 50\,\mu\text{s}$. In the last cycle, output generations continue after the end of the cycle. This is expected and acceptable because outputs are executed in asynchronous output threads.

V. Experimental Results

The proposed method has been evaluated using three set of experiments. The first set of experiments helps to measure the performance of our method in terms of real-time behavior and I/O performance, respectively. The second set aims to analyze the behavior of concurrent outputs. The last experiment is the design of a new industrial embedded system, which is tested with real devices and transaction-level SystemC models.

A. Implementation Details

SystemC simulation kernel is executed on the Linux OS kernel. The real-time patch has been applied to SystemC in sc_simcontext::simulate code, where time is advanced. Additionally, the code for calling update real methods of all channels of type dsc_hybrid_channel has been inserted at the same place.

Fig. 13 shows a simple example of the proposed hybrid channel. It has a SystemC interface sc_signal_inout_if on one side, and a handle to the I/O driver on the other side. The pin uses parallel communication, so it inherits from the class dsc_parallel_hybrid_channel.

Two more complex examples realize communication over Ethernet as shown in Fig. 14:

1) sc_hybrid_eth_in for data from Ethernet to SystemC model;
2) sc_hybrid_eth_out for the reverse direction.

In order to minimize the time during virtual model execution ($t_{W\text{passed}}$ in Fig. 3), I/O operations have been delegated to the OS threads. For instance, in sc_hybrid_eth_in, an OS thread recv_thread does the actual reception from Ethernet, and then a SystemC thread gets the data to the virtual model. Similarly, actual transmission is done in the OS thread send_thread, which is triggered by a SystemC thread. Queues hold the incoming or outgoing data for the transfer between threads. Ethernet is a kind of serial communication, so these classes inherit from dsc_serial_hybrid_channel. Similar to the Ethernet hybrid channel, we have also built a UDP/IP hybrid channel sc_hybrid_udp capable of both input and output.

Management of delta cycles is handled differently in pin and Ethernet channels. Pin channel implements a SystemC signal, so the value can change in successive delta cycles and it makes sense to delay the output until the final value is established for the current simulation time. On the other hand, Ethernet

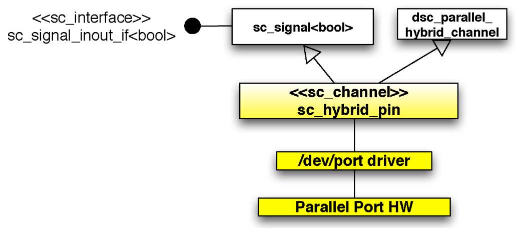

Fig. 13. Hybrid pin channel.

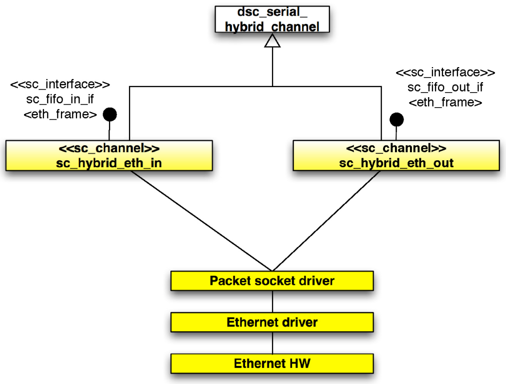

Fig. 14. Hybrid Ethernet channels.

channels are FIFO channels, and each written value should be transferred to the output regardless of later operations so that the output device can start processing the data as soon as it receives it [3]. Thus, sc_hybrid_pin generates the actual output in update_real, i.e., just before the time advance phase, while sc_hybrid_eth_out sends the data right away to the OS thread at the evaluate phase.

To improve determinism, Linux kernel's preemptibility has been increased via the RT_PREEMPT patch [44]. SystemC thread and OS threads for the I/O operations have been set to real-time scheduling and their priorities have been set to a priority directly below the interrupt handling threads. Since a computer with swap memory has been used in the experiments, all memory pages belonging to the simulation process have been locked in memory to avoid latencies due to page faults. The thread stack has been extended beyond the maximum point used, in order to avoid page faults due to stack growth.

Our software modeling platform consists of Linux 2.6.31.6-rt19 with RT_PREEMPT mode turned on and SystemC version 2.2.0 with our real-time patch. CPU load is generated via multiple instances of an infinitely spinning shell script. For the round-trip time (RTT) experiment, a PC with dual Intel Quad-Core Xeon processors running at 3.4 GHz is used and for the BACnet broadcast management device (BBMD) experiment a PC with Intel Pentium 4, 3.2 GHz HT is used.

B. I/O Performance

RTT experiment is used to measure the I/O performance of our proposed system in isolation. Fig. 15 shows the setup of the experiment. Here, the SystemC model functions as a frame replier, which sends the incoming frames back. The SystemC module sc_eth_mirror is responsible for sending the received frames back. sc_hybrid_eth_in and sc_hybrid_eth_out relay

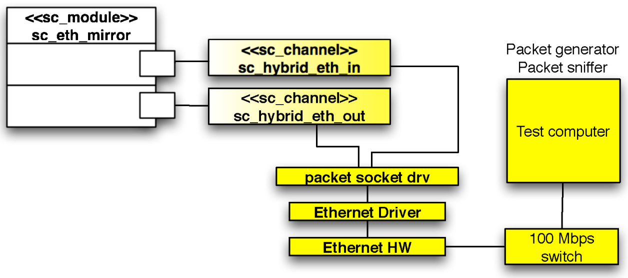

Fig. 15. Experiment setup of RTT measurement.

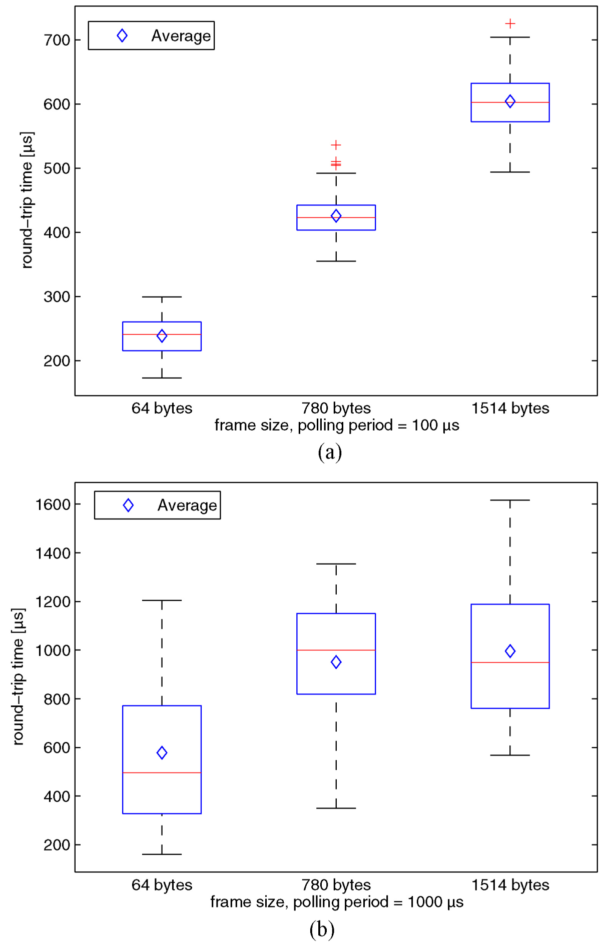

Fig. 16. RTT results with nonadaptive polling for (a) 100 μs and (b) 1000 μs.

the frames between the Ethernet interface and the SystemC model. The measurement is then done on initial sender's side. Three proposed methods for incorporating external events to the simulation are evaluated in three experiments: nonadaptive polling, adaptive polling, and event-driven solution. Each experiment is carried out with three different frame lengths, i.e., 64, 780, and 1514 bytes. In each run, 100 samples are taken. The performance of the three methods is shown as box plots in Figs. 16–18, respectively.

In nonadaptive polling method, polling period directly affects the performance. To demonstrate this fact, we did experiments with two different polling periods: 100 μs and 1000 μs. Fig. 16 shows that the polling time has a more significant

TABLE III
PARAMETERS OF RTT EXPERIMENT WITH ADAPTIVE POLLING

Parameter	Value(s)
K_P	-0.1
K_I	-0.05
K_D	0
Max. polling_period	10 ms
Min. polling_period	10 μs

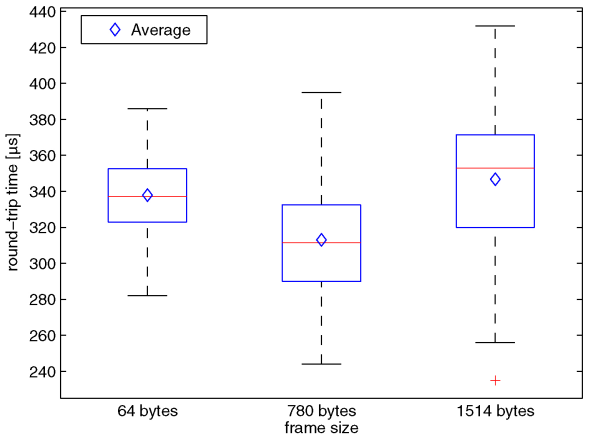

Fig. 17. RTT results with adaptive polling.

effect than the frame size. It has also been observed that polling time results in high jitter of RTT because the response time is directly dependent on the current phase of the polling cycle. Furthermore, frame size has a linear effect on RTT, as most operations—copy, Ethernet propagation—are affected linearly. For cyclic communication, cycle times of 1 ms may require low polling cycles, 100 μs in this experiment, since measured values exceed this value in other cases. However, at periods of 10 ms and higher, no further tuning is necessary for satisfactory performance.

Parameters for adaptive polling include a maximum and minimum value for the polling period and the coefficients K_P, K_I, and K_D of the PID controller. Our experimental parameter values are listed in Table III. The results are shown in Fig. 17. The results depend strongly on the behavior of the PID controller, so fine-tuning of PID coefficients may yield much better results; however, this fine-tuning is costly in terms of effort. Longer frames increase the transmission time but also decrease the polling period. We have also observed that communication cycles of 1 ms are possible.

Event-driven solution performs worse than adaptive polling for large frames (Fig. 18). Event-driven solution's RTT values grow mostly linearly with the frame size. Maximum RTT is not effected by the frame size, probably this occurs due to an inherent latency in the OS constructs, which are more complex in comparison to nonadaptive polling or adaptive polling. Event-driven solution has the big advantage of not requiring any tradeoff between I/O latency and simulation performance or any tuning.

C. Concurrent Outputs

The behavior of concurrent outputs in real-time has been evaluated in these experiments. Two proposed measures, i.e.,

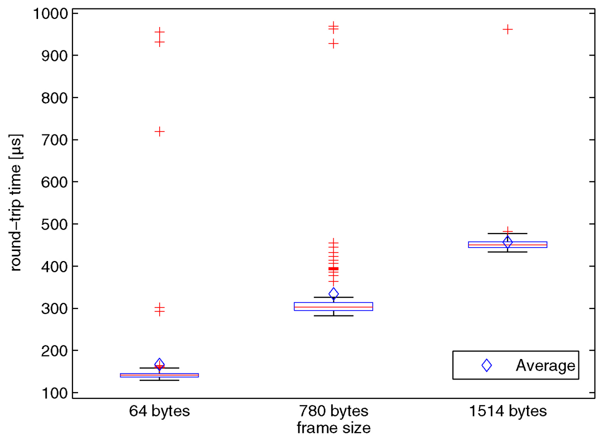

Fig. 18. RTT results with event-driven solution.

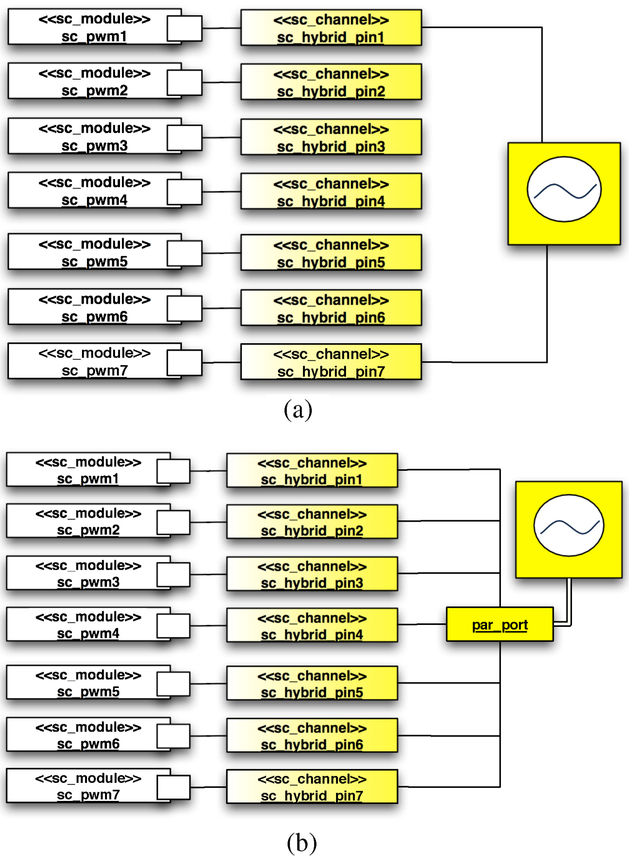
<<sc_module>> sc_pwm1
<<sc_module>> sc_pwm2
<<sc_module>> sc_pwm3
<<sc_module>> sc_pwm4
<<sc_module>> sc_pwm5
<<sc_module>> sc_pwm6
<<sc_module>> sc_pwm7
<<sc_channel>> sc_hybrid_pin1
<<sc_channel>> sc_hybrid_pin2
<<sc_channel>> sc_hybrid_pin3
<<sc_channel>> sc_hybrid_pin4
<<sc_channel>> sc_hybrid_pin5
<<sc_channel>> sc_hybrid_pin6
<<sc_channel>> sc_hybrid_pin7
(a)
<<sc_module>> sc_pwm1
<<sc_module>> sc_pwm2
<<sc_module>> sc_pwm3
<<sc_module>> sc_pwm4
<<sc_module>> sc_pwm5
<<sc_module>> sc_pwm6
<<sc_module>> sc_pwm7
<<sc_channel>> sc_hybrid_pin1
<<sc_channel>> sc_hybrid_pin2
<<sc_channel>> sc_hybrid_pin3
<<sc_channel>> sc_hybrid_pin4
<<sc_channel>> sc_hybrid_pin5
<<sc_channel>> sc_hybrid_pin6
<<sc_channel>> sc_hybrid_pin7
par_port
(b)

Fig. 19. Experiment setup for concurrent outputs. (a) Without hardware support. (b) With hardware support.

ordering outputs ($H = 7$ and $m_s = 2$) and using hardware support, if available, have been studied in two experimental setups shown in Fig. 19.

In both setups, seven digital output signals are written by the model to the parallel port. Without hardware support, all channels write their values to the parallel port without combining it with other channels. When using hardware support, all output channels send their data first to another class called par_port, which combines all values and writes to the hardware at a single point in time. We achieved 100% concurrency with

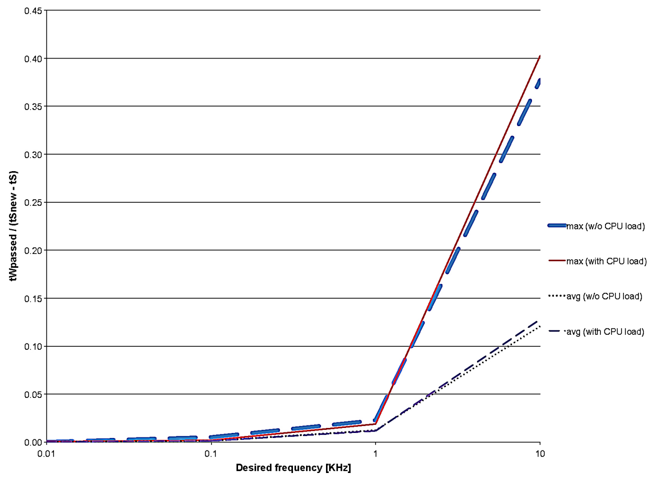

Fig. 20. PWM behavior for different desired frequencies.

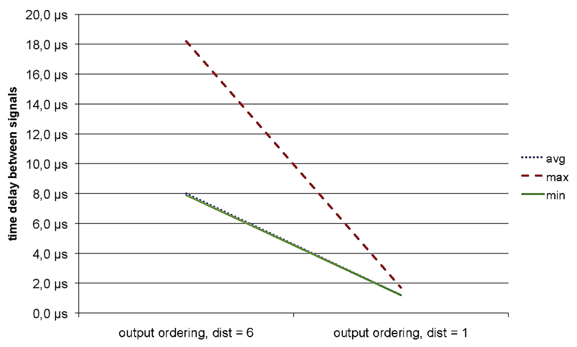

Fig. 21. Real-time difference between signals pin1 and pin7 of Fig. 19(a).

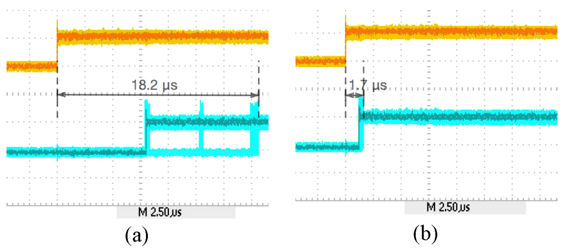

Fig. 22. Waveforms of two concurrent signals in real time (persistence = infinite). (a) With five channels in between. (b) Successive.

hardware support. On using output ordering without hardware support, the time delay between concurrent signals can be reduced 6.7 times on average (Figs. 21 and 22). Maximum delay is reduced 10.7 times, probably because longer delay is more likely to be caught by the maximum system latency.

D. Evaluation of the Mathematical Model

We verified the proposed mathematical model using the above experiment results. The first part of the model (Constraint 1) dealing with the real-time execution can be verified using the PWM experiment. As seen in Fig. 20, the ratio $t_{W\text{passed}}/(t_{S\text{new}} - t_S)$ derived directly from (2) is reflected in the output signal. Hence, Constraint 1 is satisfied.

The second part of the mathematical model (Constraint 2) dealing with concurrent outputs can be verified using the results in Fig. 21 explained above. We know from (7) and (8)

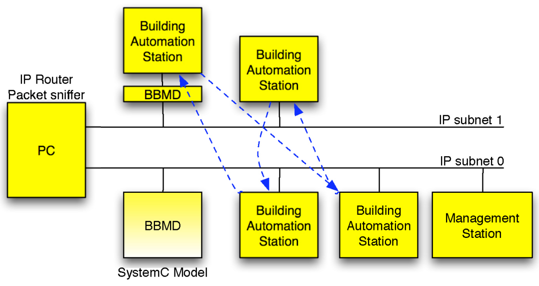
Building Automation Station
Building Automation Station
IP Router Packet sniffer
BBMD
PC
IP subnet 1
IP subnet 0
BBMD
SystemC Model
Building Automation Station
Building Automation Station
Management Station

Fig. 23. Experiment setup of BBMD.

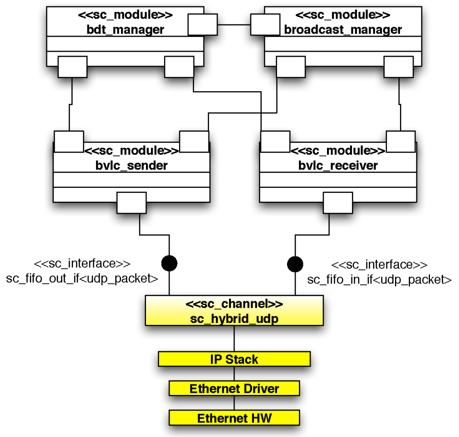
<<sc_module>> bdt_manager
<<sc_module>> broadcast_manager
<<sc_module>> bvlc_sender
<<sc_module>> bvlc_receiver
<<sc_interface>> sc_fifo_out_if<udp_packet>
<<sc_interface>> sc_fifo_in_if<udp_packet>
<<sc_channel>> sc_hybrid_udp
IP Stack
Ethernet Driver
Ethernet HW

Fig. 24. Model of BBMD.

that the relationship $(H-1)/(m_s-1) \leq t_{W\text{ow}}/t_{W\text{cow}_s}$ needs to be satisfied for the concurrent output experiment. This is because in this experiment, there is only one set of concurrent outputs. Also, in this experiment, $H = 7$ and $m_s = 2$, which implies that $t_{W\text{ow}}$ must be at least six times larger than $t_{W\text{cow}_s}$. We observe from Fig. 21 that $t_{W\text{ow}}/t_{W\text{cow}_s} = 6.7$, which satisfies the relationship above. Hence, Constraint 2 is verified.

E. Industrial Automation System Experiment

Fig. 23 shows the setup of BBMD experiment. BACnet is a communication protocol used widely in building automation networks [45]. BBMD is a subset of the BACnet protocol and is used in BACnet networks running over the Internet protocol (IP) to ensure that the broadcast packets used by BACnet are distributed correctly to all IP subnetworks constituting the BACnet network. In this experiment, an untimed transaction-level model (TLM) of a new generation BBMD has been created, which may then be extended to a fully BACnet-compliant device. Our method allowed us to test this new-generation BBMD model with old-generation real BBMD devices. The test network consists of two IP subnets comprising four Siemens S7-300 building automation stations, a management station for monitoring other stations, and a PC acting as an IP router and a packet sniffer. Each IP subnet needs one BBMD. One of the building automation stations has been configured to act also as a BBMD for one subnet. For the other subnet, the BBMD model has been connected to the network using our method. In addition to the traffic between the management station and the building automation stations, there is peer-to-peer traffic among building automation stations, shown with arrows in Fig. 23.

Our TLM of the new-version BBMD has performed very well in the experiment. It has outperformed the existing real BBMD in terms of response time and packet drop rates. Under a traffic burst of 2000 incoming packets per second, our model has not dropped any packet while the real BBMD has dropped 67% of packets. As the model was untimed, the accuracy of response time has not been an evaluation criterion and the model has outperformed the real BBMD in average response time up to 80 times.

VI. Discussion

We addressed the lack of a structured communication mechanism between the virtual models and the real devices with the concept of hybrid channel in SystemC. This concept allows minimal interference with SystemC model development and also provides a very clear interface via SystemC's native mechanisms. Hybrid channels are also very generic tools to implement every kind of communication between the real devices and virtual models.

In order to let virtual models behave according to the real-time in an accurate manner, a real-time patch has been added to the SystemC kernel. This patch is nonintrusive to the SystemC model, i.e., it allows all models running on top of the patched simulation kernel to behave according to the real-time in a transparent manner. A deterministic behavior for industrial applications has been achieved even with inexpensive off-the-shelf components like the standard parallel port and Ethernet hardware.

Polling posed a difficult tradeoff between the I/O performance and the simulation performance, both of which are important. Adaptive polling and a event-driven simulation kernel capable of incorporating external events have been devised as a remedy. Polling is still an option for simple configurations. Adaptive polling can be used when complex OS constructs have to be avoided to reach lower I/O latencies. Event-driven solution can be used even more widely after ensuring better behavior of complex OS constructs, e.g., by using a native RTOS instead of Linux with real-time improvements.

The mathematical model has also been developed and verified in the scope of this paper. This model can foremost be used as a first test to see whether our method is applicable to a given SystemC specification. The empirical data needed by the model can be obtained through profiling on the virtual model platform. Additionally, the mathematical model can also be used in order to understand bottlenecks and improvement points in the virtual model. The mathematical model can also be adapted to improve SystemC kernels, where parallelism is employed to increase simulation's performance.

The level of determinism gained via RT_PREEMPT solution of Linux is satisfactory for our purposes. API-transparency is a big advantage of this approach. However, this method remains largely manual. The modeling platform needs to be tested and tuned until a satisfactory result is obtained. Real-time Linux [44] is constantly being improved, and new tools are being developed so the manual approach is fading in favor of more structured ones, which favors our choice to use RT_PREEMPT.

Only models able to run at least as fast as real-time can be targeted with our approach. Thus, industrial applications domain with TLMs is feasible with the current state of the art. Nowadays, there are a lot of studies in the fields of increasing computation power of processors and increasing execution speed via parallel execution or optimization. Further advances in these fields will multiply the possible domains to use the proposed method.

VII. Conclusion

We presented a rapid modeling and simulation environment to integrate virtual models with real devices. We used the popular SystemC language for virtual models. We addressed issues related to integrating virtual models and real devices with that of a novel hybrid channel concept. Our experimental results validated the effectiveness of our framework on industrial automation systems. We also developed a mathematical model to estimate the validity of the heterogeneous framework.

References

[1] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A real-time service-oriented architecture for industrial automation," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 267–277, Aug. 2009.

[2] *IEEE Standard SystemC Language Reference Manual*, IEEE Standard 1666-2005, 2005.

[3] T. Groetker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Boston, MA: Kluwer, 2002.

[4] R. Fujimoto, *Parallel and Distributed Simulation Systems*. New York: Wiley-Interscience, 2000.

[5] S. Nagaraj and B. Detrick, "HIL and RCP tools for embedded controller development in hybrid vehicles," in *Proc. IEEE VPPC*, Sep. 2009, pp. 896–902.

[6] J.-C. Lee and M.-W. Suh, "Hardware-in-the loop simulator for ABS/TCS," in *Proc. IEEE Int. CCA*, Aug. 1999, pp. 652–657.

[7] F. Gu, W. S. Harrison, D. M. Tilbury, and C. Yuan, "Hardware-in-the-loop for manufacturing automation control: Current status and identified needs," in *Proc. IEEE Int. CASE*, Sep. 2007, pp. 1105–1110.

[8] J. Piedboeuf, F. Aghili, M. Doyon, Y. Gonthier, E. Martin, and W.-H. Zhu, "Emulation of space robot through hardware-in-the-loop simulation," in *Proc. Int. Sym. AI Robot. Automat. Space*, Jun. 2001, pp. 1–8.

[9] dSPACE. (2012) [Online]. Available: http://www.dspaceinc.com

[10] Simulink. (2012) [Online]. Available: www.mathworks.com/products/simulink

[11] Modelica. (2012) [Online]. Available: https://modelica.org

[12] Dymola. (2012) [Online]. Available: http://www.3ds.com/products/catia/portfolio/dymola/overview

[13] SimEvents. (2012) [Online]. Available: www.mathworks.com/products/simevents

[14] G. Lipovszki and P. Aradi, "Simulating complex systems and processes in LabVIEW," *J. Math. Sci.*, vol. 132, no. 5, pp. 629–636, 2006.

[15] T. Beltrame, "Design and development of a Dymola/Modelica Library for discrete event-oriented systems using DEVS methodology," M.S. thesis, Dept. Comput. Sci., Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland, 2006.

[16] V. S. Prat, A. Urquia, and S. Dormido, "ARENALib: A Modelica Library for discrete-event system simulation," in *Proc. Int. Modelica Conf. MODELICA*, 2006, pp. 539–548.

[17] xPC Target. (2010) [Online]. Available: http://www.mathworks.com/products/xpctarget

[18] Veristand. (2012) [Online]. Available: http://www.ni.com/veristand

[19] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler, "Execution strategies for PTIDES, a programming model for distributed embedded systems," in *Proc. IEEE RTAS*, Apr. 2009, pp. 77–86.

[20] F. Balarin and R. Passerone, "Specification, synthesis, and simulation of transactor processes," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 10, pp. 1749–1762, Oct. 2007.

[21] C. E. Pereira and L. Carro, "Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control," *Ann. Rev. Contr.*, vol. 31, no. 1, pp. 81–92, 2007.

[22] L. Becker and C. Pereira, "SIMOO-RT: An object-oriented framework for the development of real-time industrial automation systems," *IEEE Trans. Robot. Automat.*, vol. 18, no. 4, pp. 421–430, Aug. 2002.

[23] V. Vyatkin, J. Christensen, and J. Lastra, "OOONEIDA: An open, object-oriented knowledge economy for intelligent industrial automation," *IEEE Trans. Ind. Inform.*, vol. 1, no. 1, pp. 4–17, Feb. 2005.

[24] *Open Standard for Distributed Control and Automation*, IEC Standard 61499, 2010.

[25] V. Vyatkin, "The IEC 61499 standard and its semantics," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 40–48, Dec. 2009.

[26] R. Checcozzo, F. Rusina, L. Mangeruca, A. Ballarino, C. Abadie, A. Brusaferri, R. Harrison, and R. Monfared, "RI-MACS: An innovative approach for future automation systems," *Int. J. Mechatron. Manuf. Syst.*, vol. 2, no. 3, pp. 242–264, 2009.

[27] F. Herrera and E. Villar, "A framework for embedded system specification under different models of computation in systemC," in *Proc. DAC*, 2006, pp. 911–914.

[28] M. Moallemi, G. Wainer, F. Bergero, and R. Castro, "Component-oriented interoperation of real-time DEVS engines," in *Proc. Conf. SpringSim*, Apr. 2011, pp. 127–134.

[29] F. Madlener, H. Molter, and S. Huss, "SC-DEVS: An efficient systemc extension for the DEVS model of computation," in *Proc. Conf. DATE*, Apr. 2009, pp. 1518–1523.

[30] M. Monton, A. Portero, M. Moreno, B. Martinez, and J. Carrabina, "Mixed SW/SystemC SoC emulation framework," in *Proc. IEEE ISIE*, Jun. 2007, pp. 2338–2341.

[31] N. Kim, H. Choi, S. Lee, S. Lee, I.-C. Park, and C.-M. Kyung, "Virtual chip: Making functional models work on real target systems," in *Proc. Des. Automat. Conf.*, 1998, pp. 170–173.

[32] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura, "A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ emulator and FPGA emulator with shared register communication," in *Proc. Des. Automat. Conf.*, 2004, pp. 299–304.

[33] SynaptiCAD. (2002) [Online]. Available: http://www.syncad.com/pr_pinport_release.htm

[34] L. Benini, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Virtual in-circuit emulation for timing accurate system prototyping," in *Proc. IEEE Int. ASIC/SOC Conf.*, Sep. 2002, pp. 49–53.

[35] C. Koehler, A. Mayer, and A. Herkersdorf, "Chip hardware-in-the-loop simulation coupling optimization through new algorithm analysis technique," in *Proc. Int. Conf. Mixed Des. Integr. Circuits Syst. MIXDES*, Jun. 2009, pp. 412–416.

[36] U. Nageldinger, A. Pyttel, and H. Kleve, "System simulation speedup combining systemC models and reconfigurable hardware," in *Proc. SpeAC Workshop*, 2004, pp. 1–24.

[37] R. Ramaswamy and R. Tessier, "The integration of systemC and hardware-assisted verification," in *Proc. Int. Conf. FPL App.*, 2002, pp. 1007–1016.

[38] D. Fennibay, A. Yurdakul, and A. Sen, "Introducing hardware-in-loop concept to the hardware/software co-design of real-time embedded systems," in *Proc. IEEE ICESS*, Jun. 2010, pp. 1902–1909.

[39] M. Trams. (2005). *Realtimify: A Small Tool for Real Time SystemC Simulations* [Online]. Available: http://www.digital-force.net/download.php?file=publications/systemc_realtimify.pdf

[40] M. Chitnis, P. Gai, G. Lipari, P. Pagano, and A. Romano, "Rapid prototyping suite of IEEE 802.15.4-compliant sensor networks," in *Proc. IEEE Int. Conf. MASS*, Oct. 2007, pp. 1–3.

[41] P. Mantegazza, E. Bianchi, L. Dozio, S. Papacharalambous, S. Hughes, and D. Beal, "RTAI: Real time application interface," *Linux J.*, vol. 2000, no. 72, pp. 1–10, 2000.

[42] B. Lu, X. Wu, H. Figueroa, and A. Monti, "A low-cost real-time hardware-in-the-loop testing approach of power electronics controls," *IEEE Trans. Ind. Electron.*, vol. 54, no. 2, pp. 919–931, Apr. 2007.

[43] K. Yaghmour. (2001). *Adaptive Domain Environment for Operating Systems*, pp. 1–7 [Online]. Available: http://www.opersys.com/adeos

[44] *Real-Time Linux Wiki* (2009) [Online]. Available: http://rt.wiki.kernel.org

[45] *BACnet: A Data Communication Protocol for Building Automation and Control Networks*, ASHRAE Standard 135-2001, 2004.

Dogan Fennibay (M'08) received the B.S. degree in computer and industrial engineering from Istanbul Technical University, Istanbul, Turkey, in 2006, and the M.S. degree in computer engineering from Bogazici University, Istanbul, in 2010.

In 2005, he was an Embedded Software Engineer with Siemens AS, Istanbul. Since 2011, he has been a Software Architect involved in embedded industrial systems. His current research interests include real-time systems, system-level modeling, industrial communication, and embedded operating systems.

Arda Yurdakul (M'91) received the B.S., M.S., and Ph.D. degrees in electrical and electronics engineering from Bogazici University, Istanbul, Turkey in 1992, 1994, and 1999, respectively.

She is currently an Associate Professor with the Department of Computer Engineering, Bogazici University. Her current research interests include embedded systems, reconfigurable computing, system-level and high-level design automation, and mathematical modeling.

Alper Sen (SM'09) received the B.S. and M.S. degrees in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 1995 and 1997, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, in 2004.

He is currently an Assistant Professor with the Department of Computer Engineering, Bogazici University. He was a Technical Staff Member with Freescale Semiconductor, Austin, and an Adjunct Faculty Member with the University of Texas at Austin, until 2009. His current research interests include verification of hardware and software systems, parallel programming, embedded systems, and system-level designs.