2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

Introducing Hardware-in-Loop Concept to the Hardware/Software Co-design of Real-time Embedded Systems

Dogan Fennibay*†, Arda Yurdakul† and Alper Sen†
*Corporate Technology, Siemens AS, Kocaeli, Turkey
†Department of Computer Engineering, Bogazici University, Istanbul, Turkey
Email: {dogan.fennibay, yurdakul, alper.sen}@boun.edu.tr

***Abstract*—As the need for embedded systems to interact with other systems is growing fast, we see great opportunities in introducing the hardware-in-the-loop technique to the field of hardware/software co-design of embedded systems. This technique reduces the need to develop models for existing hardware and increases the accuracy of the overall system. This work is especially important now that complexity and time-to-market constraints demand early simulation, verification, and architectural exploration of systems. We introduce the hardware-in-the loop technique to the field of hardware/software co-design of industrial embedded systems using SystemC as the modeling environment. We conceptualize the hybrid channel to clearly define the communication between real and virtual (modeled) subsystems. We patch the SystemC kernel for hard real-time execution and we improve the underlying operating system to guarantee an upper bound for the overall system latency. We have performed tests to measure the performance of our method in terms of response time and determinism. We have achieved a stable operating frequency of 10 KHz and an I/O performance of sub-millisecond round-trip time over Ethernet. Moreover we have developed a non-timed transaction-level model of a BACnet Broadcast Management Device (BBMD) and connected it with real devices to see our method's performance in a real-life environment. Our model outperformed the competing real system up to 80 times in maximum response time. We deem the results very promising for the future of our method.**

I. Introduction

System-level modeling is a relatively new approach in the development process (from architectural exploration to verification) since systems are getting more integrated [1]. In this trend, hardware and software are also getting closer, hence hardware/software co-design is increasingly employed and system-level modeling comprises modeling of hardware and software together. Systems under development are usually complex. As a result, current design trend is the employment of models and modular/component-based strategies.

Two additional trends in embedded systems are that (1) they are increasingly connected with other embedded systems and (2) they increasingly contain off-the-shelf components to shorten the time-to-market and reduce development costs. We refer to other embedded systems and off-the-shelf components as *real subsystems*. These trends imply that real subsystems also have to be modeled even though their implementations exist. Modeling of such elements has no added value as it multiplies the intrinsic disadvantages of modeling, i.e. inaccuracy due to abstraction and additional effort.

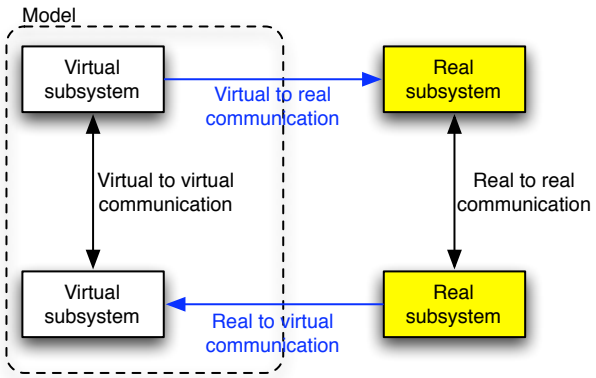

Fig. 1. Types of communication in a hardware-in-the-loop setup

We believe that real subsystems should not be modeled in the systems under development because they are already implemented and these implementations should be integrated with the system-under-development. A well-known method which is used in the test of implemented embedded systems is hardware-in-the-loop [2]. We introduce the same concept to hardware/software co-design of embedded systems. We define novel communication mechanisms between virtual and real subsystems to implement hardware-in-the-loop (Figure 1) method for hardware/software co-design. We also introduce real-time behavior for virtual subsystems, as most communication mechanisms between subsystems rely on a common notion of time such as a timeout mechanism in a communication protocol.

Real-time behavior consists of synchronizing the clocks of the virtual subsystems with the clocks of the real subsystems and achieving determinism in the overall system. Non-deterministic behavior is present in all computing platforms with an operating system and decreases the accuracy of the model by introducing latency and by increasing the response time randomly. Achieving determinism in the overall system requires bounding such latencies to limit the effects on the

978-0-7695-4108-2/10 $26.00 © 2010 IEEE
DOI 10.1109/CIT.2010.325

accuracy of the models.

We deem SystemC as the most appropriate platform to introduce the hardware-in-the-loop technique to hardware/software co-design, because it is widely used and standardized [3]. In this study, we introduce the *hybrid channels* concept by extending the SystemC channel construct to clearly specify the communication between real and virtual subsystems. Additionally, we adapt the SystemC kernel to execute the simulation in real-time and improve the underlying operating system to limit the system latency. We also add special mechanisms to properly manage timing of communication between virtual and real subsystems.

We evaluate our method in the domain of industrial applications, more specifically industrial communication. The basic reasons are: (1) there exist industrial communication standards that have very strict hard real-time constraints such as PROFINET [4], [5]; (2) the data exchange rate in industrial communication ranges up to 10 KHz, and this is achievable with current computing systems executing the models; and (3) system-level modeling is starting to be a design trend in this domain. Our method can also be applied directly to the design of SoCs if computation power of the modeling platforms becomes sufficient for executing the complex SoC models in real-time. There is a vast amount of research to speed up simulation of SystemC models via parallelization or optimization. In addition, the increasing use of system-level modeling for software, which has lower execution speed, will also be an application area for our method.

This paper is organized as follows: Next section presents the related work, it is followed by a background section summarizing the preliminaries. We present our solution in Section IV and our experimental evaluation in Section V. Final section concludes the work and sets directions for future research.

II. Related Work

A. Current hardware-in-the-loop techniques

As a well-established technique, hardware-in-the-loop has well-established tools. Most famous among them are MathWorks' solutions xPC Target [6] and Real-Time Windows Target [7], where the model is executed on a dedicated system or on a Windows system, respectively. However, the modeling language provided by these solutions has not been designed for hardware/software co-design purposes, so it lacks the necessary constructs and mechanisms that are already present in SystemC. Our work is orthogonal to such existing methods, as we are proposing to introduce the technique to the field of hardware/software co-design with a much more powerful modeling language, namely SystemC.

B. Integration of different environments

There have been several studies regarding the integration of different environments, i.e. enabling different modeling environments to interact with each other and also enabling interactions between models and real systems.

1) Integrating different virtual platforms: These methods introduce concepts and software mechanisms to integrate different modeling platforms. As there are no real subsystems involved, there is no need for real-time behavior, but the need for synchronizing executions of virtual environments.

HetSC [8] accomplishes to integrate multiple models of computation in a SystemC model together, which allows more accurate modeling of complete systems. This study only deals with integration of parts of a SystemC model and not integration of a SystemC model with external environments.

The work in [9] aims to integrate QEMU emulation environment and SystemC. It employs a SystemC module instead of a SystemC channel for representing the communication, which we believe would cause design difficulties as SystemC foresees use of channels for communication purposes. In [9], an additional channel is necessary to connect the integration module to the rest of the model. However, this is a double effort. The integration can be directly implemented with an hierarchical SystemC channel.

2) Integrating real and virtual environments: A major decision point in the integration of real and virtual subsystems is the level of abstraction. The communication between real and virtual subsystems can range from pin-level to transaction-level. The works in [10]–[14] are examples of the pin-level communication, and the works in [9], [15]–[18] are examples of the transaction-level communication. It should be noted that not all approaches address the hardware-in-the-loop method; [11], [15], [17], [18] propose connecting models to real subsystems only to increase the overall execution performance by using the real subsystems as coprocessors.

Each work has interesting properties that show the variety of possible answers to the question of which level of abstraction to use for the communication. The hardware-in-the-loop framework proposed by Underwood [12] does the integration at the analog signal level via A/D, D/A converters, while Virtual Chip [10] and PinPort [14] use pins directly. Work in [11] employs register-level transfers, which can be considered as pin-level. Pin-level approaches offer great flexibility, as any communication protocol can be modeled on top of pin-level when necessary. However, when the subsystem using the communication protocol is in focus rather than the protocol itself, modeling the protocol will be costly, it will also decrease accuracy and the execution speed of the whole model unnecessarily.

Approaches that prefer transaction-level communication between real and virtual subsystems allow to skip the modeling of the details of the communication and focus on other interesting parts. Virtual In-Circuit Emulator [15] and Chip Hardware-in-the-Loop Simulation [16] use the remote debugging interface of a microprocessor to integrate it to the simulation model. This approach works well for microprocessors, however it does not address other hardware-in-the-loop configurations. The approach in [9], is more flexible in that aspect, it is already able to model two bus systems, namely Advanced Microcontroller Bus Architecture (AMBA) and Peripheral Component Interconnect (PCI).

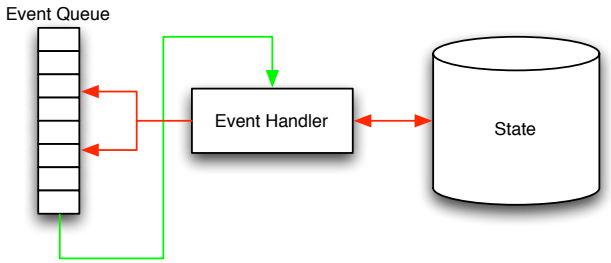

Fig. 2. Basic architecture of a discrete event simulator

C. Timing concerns

Virtual Chip has a contribution in the timing management between real and virtual subsystems. The authors devise a three-level device integrating the model and the real system. The device consists of the Internal Interface Module, the Operational Buffer Unit and the External Interface Module. With respect to the timing, behaviors of the Internal and the External Interface Module are synchronized with the model and the real subsystem, respectively. Operational Buffer Unit connecting both interface modules handles the timing difference via buffering methods [10].

Realtimify is an approach to real-time execution of SystemC models. Basically, a module is added to the model which synchronizes the simulation's execution to real-time with the objective to monitor the execution in real-time [19]. The approach is very lean and satisfactory for observing the model's execution in real-time. However, it does not address the issue of determinism as interaction with real subsystems besides human interaction is not targeted. Additionally the approach is intrusive as it requires changes in the model. Finally, it relies on the non-deterministic SystemC scheduler to execute the synchronization, which results in uncertainty about when the model will be synchronized.

D. Deterministic behavior

Operating system plays a critical role in the issue of determinism. Real-time operating systems (RTOS) specialize in providing deterministic behavior, but they lack the variety of applications, I/O interfaces and functionality provided by a general purpose operating system (GPOS). Linux with real-time improvements seems as a promising tradeoff. Real-time Application Interface (RTAI) [20] which is used in [13] is built on top of Adaptive Domain Environment for Operating Systems (ADEOS) [21] and does time sharing with a Linux kernel. It provides real-time behavior by itself while leaving the resources to the Linux kernel for non-critical tasks. On the other hand RT_PREEMPT [22] employs a more direct method in which latency is decreased by increasing preemptibility throughout the Linux kernel.

III. Preliminaries

SystemC simulation kernel is a discrete event simulator (Figure 2) [1], the *simulation clock* is advanced in discrete

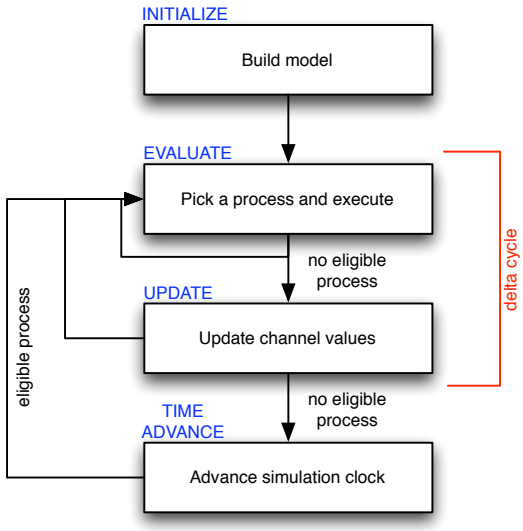

Fig. 3. Flow of SystemC scheduler [1]

time intervals and changes in the model only happen at discrete points in time. An *event* happens at a point in time, specifies changes to the *state* of the simulation and adds/removes elements to/from the *event queue*. Event queue holds an ordered list of events according to their timestamps. *Event handler* processes events in order and executes changes specified by events. The simulation clock is part of the state and is advanced according to the timestamp of the next event in the event queue. [23].

It should be stressed that a SystemC event is not the same thing as an event in a discrete event simulator. A SystemC event only represents an occurrence in the simulation model, so that processes in the model can notify others via SystemC events or wait on SystemC events. To avoid confusion, we will always refer to a SystemC event with *sc_event*, and use only *event* for referring to events in a discrete event simulator.

Two different clocks are involved in a discrete event simulator: the *simulation clock* representing the virtual clock of the model and the *wall clock* (a.k.a. real-time clock) representing the physical time passing during the execution of the simulation model [23]. Real-time simulation consists of establishing the relationship given in Equation (1) between the simulation clock T_S and the wall clock T_W, so that the advance of simulation clock is bound to the wall clock [24].

$$\frac{T_S - T_{Sstart}}{T_W - T_{Wstart}} = 1 \tag{1}$$

SystemC kernel's execution consists of four main phases: initialize, evaluate, update and time advance (Figure 3). Evaluate and update phases form a *delta-cycle*. A delta cycle is a zero time advance cycle where only an infinitesimal amount of time is assumed to pass. The result of operations done in the evaluate phase are not updated until the update phase in order to allow arbitrary order of execution of operations

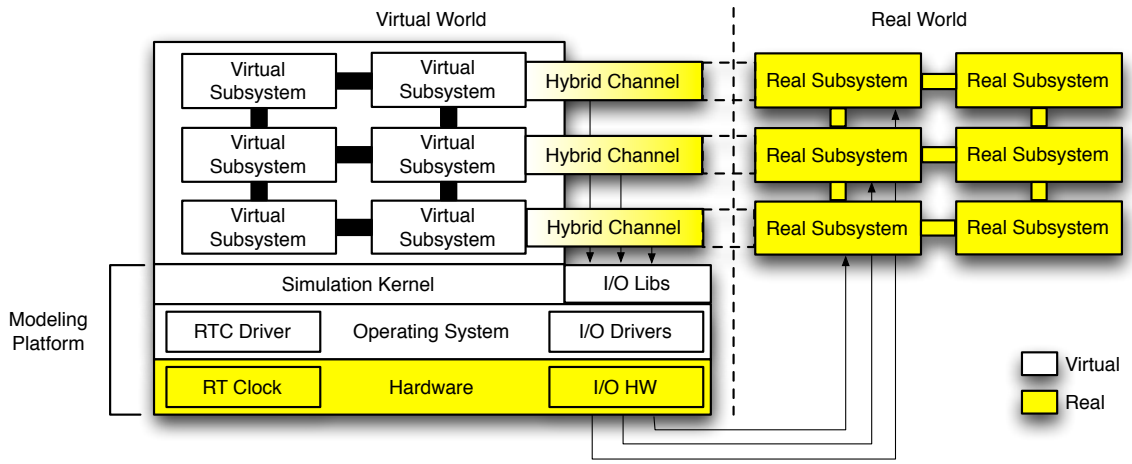

Fig. 4. Architecture of our solution

scheduled to run concurrently. This method allows modeling concurrent operations in a serial execution thread, but it also poses a difficulty for real-time simulation as there is no zero time advance in real-time.

As mentioned, SystemC uses an event-driven simulation kernel. However, all events are in the simulation model. There is no interface for an external event, e.g. new data arriving via a hybrid channel to the model, in the current implementation [1]. For example, if the event queue contains events with timestamps 00:02 and 00:06; the simulation clock will be advanced from 00:02 to 00:06 directly. So, if data has arrived from real subsystems to the model at 00:04, it will first be received at 00:06 by the model. This issue decreases accuracy of the model and should be addressed.

Another factor affecting accuracy is the nondeterministic behavior of the system. Due to inherent latencies, all computing platforms demonstrate a deviation from real-time, so the response time of the platform cannot be determined 100%. Assuring deterministic behavior consists of guaranteeing an upper bound for the system latency, which can be defined as the time needed for a system to react whenever there is an action on it. Real-time schedulers guarantee that routines handling actions are executed when necessary, but regions preventing schedulers to manage resources such as interrupt service routines, a.k.a. non-preemptible sections increase system latency. Solutions proposed in [22] and [20] decrease the system latency by increasing preemptibility.

IV. Hardware-in-the-Loop with SystemC

Figure 4 shows the architecture of our solution. Virtual subsystems, i.e. models, run on top of a simulation kernel, which runs on a GPOS. GPOS runs on a computer hardware. We call the triplet formed by the simulation kernel, the operating system and the computer hardware as the *modeling platform*. Our solution addresses two aspects of the problem: (1) achieving real-time behavior and (2) integrating real and virtual worlds.

A. Achieving real-time behavior

We patch the simulation kernel at the point where the time advance is done (Figure 5). At this point, simulation clock is still t_S and is about to be advanced to t_{Snew}, while wall clock has advanced from t_W to $t_{Wactual}$ due to the delta cycle processing time. In order to satisfy Equation (1), our patch delays the execution of the simulation by an amount of t_{Wdelay}, which advances the wall clock to t_{Wnew}.

To assure determinism, we improve the underlying operating system's preemptibility with patches and disable further sources of latency such as swap memory and power management. Additionally, we increase the priority of threads executing the model to guarantee availability of resources. Single-threaded execution model of SystemC guarantees that the contention among threads can only happen in hybrid channels, which can employ additional threads. So we designed the hybrid channels carefully to avoid excessive contention.

B. Hybrid channels: integrating real and virtual subsystems

We extend the channel concept of SystemC so as to realize the hybrid channel functionality, i.e. realizing the communication between the real and virtual subsystems. Furthermore, we categorize channel types and devise the class hierarchy shown in Figure 6. *dsc_hybrid_channel* class at the base of the hierarchy serves for distinguishing hybrid channels from other channels and implements the *update_real* functionality, which we will explain in the next paragraphs. In SystemC, a channel can inherit from *sc_prim_channel* (primitive channel) or *sc_module* (hierarchical channel). As hierarchical channels offer a superset of primitive channels' capabilities [1], we choose *sc_module* as the base of *dsc_hybrid_channel*. A hybrid channel can carry digital or analog data, which can be specified by choosing the appropriate subclass *dsc_digital_hybrid_channel* or *dsc_analog_hybrid_channel*. Digital channels can transfer

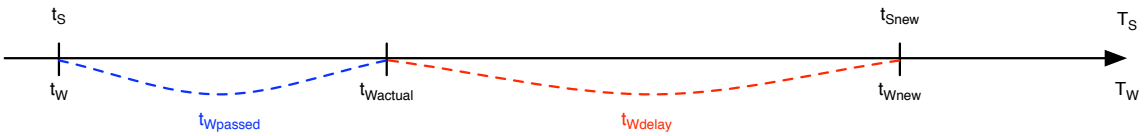

Fig. 5. Real-time patch to simulation kernel

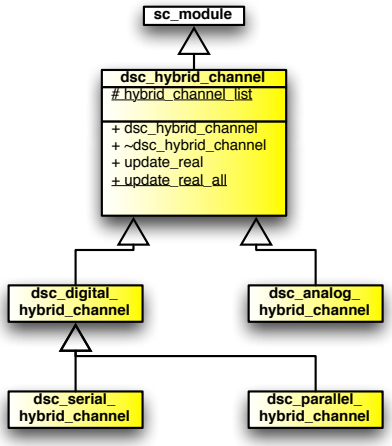

Fig. 6. Class diagram of hybrid channels

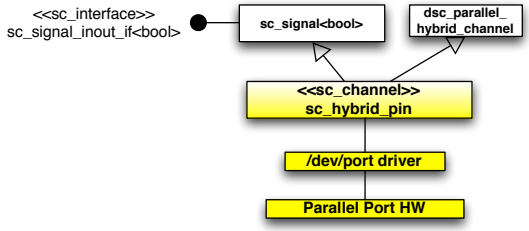

Fig. 7. Hybrid pin channel

data in a parallel way, e.g. the parallel port, or in a serial way, e.g. Universal Serial Bus (USB). Hence we provide two further subclasses for specifying this characteristic.

1) Interactions from virtual to real subsystems: Output values determined in the virtual subsystems can be transferred to the real subsystems in evaluate, update or time advance phases (Figure 3). The constraints of the channel model dictates the best phase for the transfer:

- Evaluate: The data may be transferred as soon as it is produced. (e.g. a fifo channel whose current value will not be affected by values in later delta-cycles.)
- Update: There might be several processes that affect the final value of an output variable. In that case, the data should not be written to the real subsystem until the final stable value is reached. If multiple successive delta-cycles change the data in the channel, real subsystems can observe this. (e.g. a signal channel whose actual value is established at the end of a delta-cycle)
- Time advance: Due to the sequential operation of SystemC simulation kernel, concurrent outputs cannot be transferred to the real subsystems simultaneously. When output values are transferred in the evaluate or update phase, real subsystems observe them at time points distributed in $t_{Wpassed}$ shown in Figure 5. Delaying the transfer until the time advance phase will gather the output points in time together at $t_{Wactual}$. So, outputs that are simultaneous regarding to the simulation clock are generated in a smaller time window regarding the wall clock. This option has another advantage of reducing simulation's execution effort, because the number of I/O operations are reduced. As a disadvantage, delta-delay changes are not observable by real subsystems in this scheme.

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

2) Interactions from real to virtual subsystems: SystemC kernel does not have a mechanism for receiving external events. Time is always advanced according to internal events and operations. We use SystemC thread processes to poll the external interfaces and relay this info to internal sc_events. This way the SystemC kernel becomes aware of the external events. However, an additional latency due to polling is introduced.

V. Evaluation

We evaluated our method experimentally in three cases. Two small experiments helped to measure the performance of our method in terms of real-time behavior and I/O performance. Last experiment was a case of design of new industrial software tested with real subsystems already at transaction-level model (TLM) phase.

A. Implementation details

SystemC simulation kernel was executed on Linux operating system kernel. We did the real-time patch to SystemC in *_simcontext::simulate*, where time is advanced. Additionally, at the same place we also inserted the code for calling *update_real* methods of all channels of type *dsc_hybrid_channel*.

Figure 7 shows a simple example of the hybrid channel we are proposing. 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 is a parallel communication, so inherits from the class *dsc_parallel_hybrid_channel*.

Two more complex examples realized the communication over Ethernet: *sc_hybrid_eth_in* for data from Ethernet to SystemC model, and *sc_hybrid_eth_out* for the reverse direction. In order to minimize the time during simulation's execution ($t_{Wpassed}$ in Figure 5), we delegated I/O operations to operating system threads. For instance in *sc_hybrid_eth_in*, *recv_thread* does the actual reception from Ethernet, then a SystemC thread gets the data to the model. Similarly, actual transmission is done in the operating system thread *send_thread*, which is triggered by a SystemC thread. Queues hold the incoming/outgoing data for the transfer between threads. Ethernet is a kind of serial communication, so these classes inherit from *dsc_serial_hybrid_channel*. On the same principles as the Ethernet hybrid channel, we also built a UDP/IP hybrid channel *sc_hybrid_udp* capable of both input and output.

We introduced polling mechanisms in hybrid Ethernet channels in order to incorporate external events in the SystemC model. For *sc_hybrid_eth_out* an external event is the end of transmission, which means an available slot in the outgoing queue. *poll* function checks the outgoing queue with a period of $POLLPERIOD\_US\mu s$ and notifies the sc_event if there is a free slot. *sc_hybrid_eth_in* has a similar process, but it checks if there is an element in the incoming queue, and then sets the sc_event. This way, the external event is bound to an internal event, which allows the rest of the operation to be handled via SystemC mechanisms.

Management of delta cycles was done differently in pin channel 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 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 [1]. Thus, *sc_hybrid_pin* generates the actual output in *update_real*, i.e. at the beginning of time advance phase, while *sc_hybrid_eth_out* sends the data right away to the operating system thread, which makes the transmission in parallel to the simulation's execution.

To improve determinism, Linux kernel's preemptibility was increased via the RT_PREEMPT patch [22]. SystemC thread and operating system threads doing the I/O operations were set to real-time scheduling and their priorities were set to a priority directly below the interrupt handling threads. Because a computer with swap memory was used in these experiments, all memory pages belonging to the simulation process were locked in memory to avoid latencies due to page faults. Finally, the thread stack was extended beyond the maximum point used, in order to avoid page faults due to stack growth.

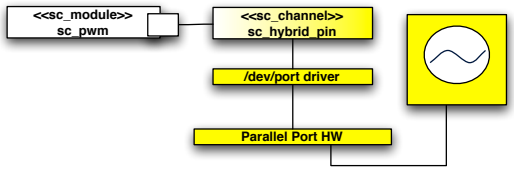

Fig. 8. Experiment setup of PWM

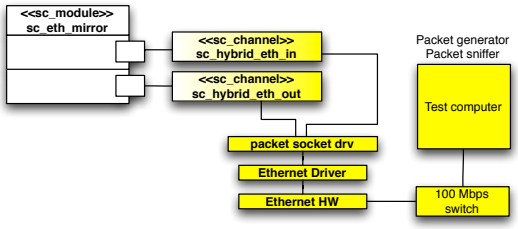

Fig. 9. Experiment setup of RTT measurement

B. Experiment setup

Pulse width modulation (PWM) experiment (Figure 8) consists of generating a square wave and examining the jitter in the output signal. Square wave was generated by a SystemC module *sc_pwm* with 50% PWM duty cycle and relayed to the parallel port of the computer via the hybrid channel *sc_hybrid_pin*. An oscilloscope was employed to examine the quality of the generated signal. The persistence parameter of the oscilloscope was set to infinite, in order to keep all of past waveforms and observe the maximum jitter. Parameters of the PWM experiment were desired frequency (0.01, 0.1, 1, 10 and 100 KHz), and presence of additional CPU load (yes/no). The evaluation criterion was the ratio of maximum jitter to the desired period. Additionally, we also measured the ratio $(t_{Snew} - t_S)/t_{Wpassed}$ (Figure 5) at each time advance to separately see the simulation's performance apart from the I/O performance.

Figure 9 shows the setup of the round-trip time (RTT) experiment. Here, the SystemC model functioned as a frame replier, which sends the incoming frames back. The SystemC module *sc_eth_mirror* was responsible for sending the received frames back. *sc_hybrid_eth_in* and *sc_hybrid_eth_out* relayed the frames between the Ethernet interface and the SystemC model. The measurement was then done on the initial sender's side. Parameters of the RTT experiment were frame length (64, 780 and 1514 bytes), and polling period (100 and 1000 μs). Evaluation criteria were maximum, minimum and average values for RTT. One hundred samples were taken in each run.

Figure 10 shows the setup of BACnet Broadcast Management Device (BBMD) experiment. BACnet is a communication protocol used widely in building automation networks [25]. 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

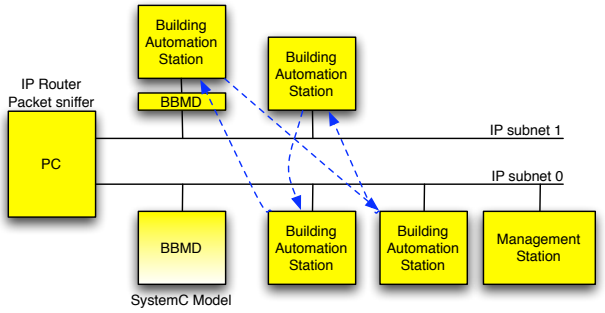

Fig. 10. Experiment setup of BBMD

distributed correctly to all IP subnetworks constituting the BACnet network. In this experiment we created a non-timed TLM of a new BBMD, which might be then extended to a fully BACnet-compliant device. Our method allowed us to test this model with real partners already in TLM phase. We built a test network 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. We configured one of the building automation stations to act also as a BBMD for one subnet. For the other subnet we connected the BBMD model to the network using our method. In addition to the traffic between the management station and the building automation stations, there was peer to peer traffic between building automation stations shown with arrows in Figure 10.

Software platform of modeling consisted 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 was generated via multiple instances of an infinitely spinning shell script. For the PWM and RTT experiments we used a PC with dual Intel Quad-Core Xeon processors running at 3.4 GHz and for the BBMD experiment we used a PC with Intel Pentium 4 3.2 GHz HT.

C. Experiment results

PWM experiment's results are given in Figure 11. The signal was stable up to 10 KHz. At this rate jitter became significantly high, however it was still below half of the period, so the square wave was still generated at the desired frequency, hence it may be still considered acceptable. At 100 KHz no meaningful jitter could be measured, as the waveform was largely corrupted (Figure 12). At stable frequencies, load had only a minor effect although the CPU was loaded 100%. This showed that the real-time operating system scheduler is working fine. Moreover, the internal measurement of the maximum value of $(t_{Snew} - t_S)/t_{Wpassed}$ matched the external measurement, so we can conclude that the simulation performance was the main factor affecting the performance rather than the I/O performance. Finally, the average value of $(t_{Snew} - t_S)/t_{Wpassed}$ showed that there is still unused capacity for higher frequencies in terms of computation power, however jitter prevented this capacity from being utilized.

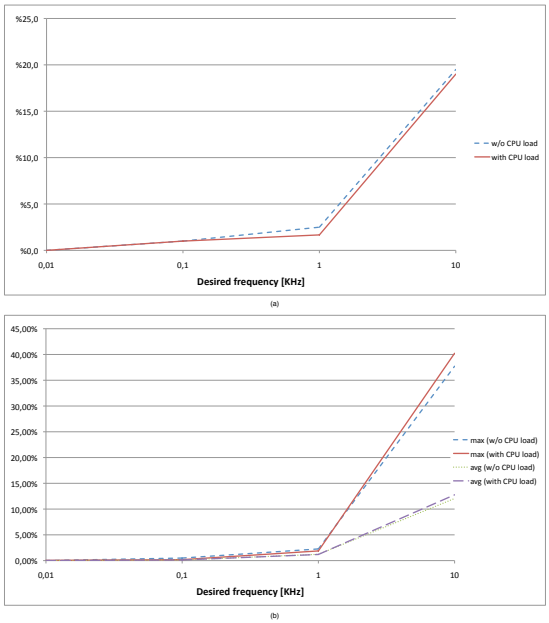

Fig. 11. PWM: For different desired frequencies, (a) Max jitter / desired period (b) $(t_{Snew} - t_S)/t_{Wpassed}$

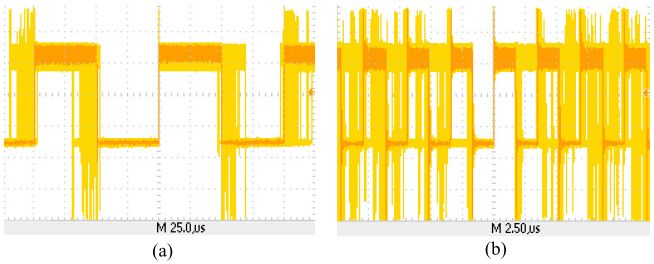

Fig. 12. Waveforms of (a) 10 KHz and (b) 100 KHz desired frequencies (persistence = infinite)

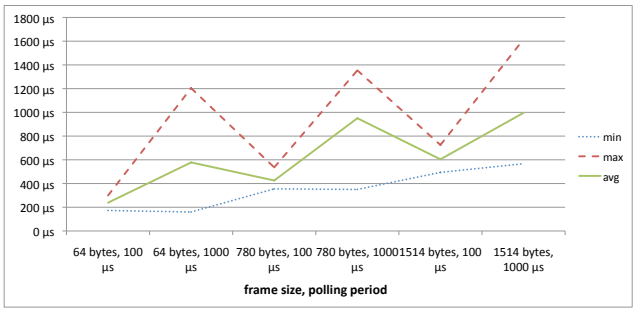

Fig. 13. RTT's max/average/min measurement for (frame size, polling time)

In RTT experiment, Figure 13 shows that the polling time had a more significant effect than the frame size. We also observed that polling time resulted in high jitter of RTT, because the response time was directly dependent on the current phase of the polling cycle. Furthermore, frame size had a linear effect on RTT, as most operations - copy, Ethernet propagation

- were affected linearly. For cyclic communication, cycle times of 1 ms may need low polling cycles since measured values exceeded this value in other cases. However, at periods of 10 ms and higher, no further tuning is necessary for satisfactory performance.

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

VI. Conclusion

In this study we have developed a hardware-in-the-loop concept for hardware/software co-design of real-time embedded systems, implemented with SystemC and evaluated experimentally. Our encapsulation of the communication between the real and virtual subsystems in a SystemC *hybrid channel* allows minimal interference to 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 real and virtual subsystems.

We achieved a deterministic behavior for industrial applications even with commonly available off-the-shelf components like standard parallel port and Ethernet hardware. We believe that with further advances in increasing simulation performance and computation power, the possible domains to use our method will multiply.

The solution devised for the incorporation of external events (polling) is satisfactory as a first result, however it imposes a difficult tradeoff: simulation speed vs. I/O latency, both of which are important. It should be listed under future work to accomplish this in a truly event-driven way, without resorting to polling. Another point for future study is the scalability, i.e. how our method will scale to models with higher number of processes, modules, channels, hybrid channels etc.

Acknowledgment

Alper Sen was supported by a Marie Curie European Reintegration Grant within the 7th European Community Framework Programme.

References

[1] T. Groetker, S. Liao, G. Martin, and S. Swan, *System design with SystemC*. Boston/Dordrecht/London: Kluwer Academic Publishers, 2002.

[2] M. Bacic, "On hardware-in-the-loop simulation," in *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05*, 2005, pp. 3194–3198.

[3] *IEEE Standard SystemC Language Reference Manual*, IEEE Std. 1666, 2005.

[4] *Industrial communication networks - Fieldbus specifications*, IEC Std. 61 158, 2007.

[5] *Industrial communication networks - Profiles*, IEC Std. 61 784, 2007.

[6] MathWorks. (2010) xpc target. [Online]. Available: http://www.mathworks.com/products/xpctarget/

[7] ——. (2010) Real-time windows target. [Online]. Available: http://www.mathworks.com/products/rtwt/

[8] F. Herrera and E. Villar, "A framework for embedded system specification under different models of computation in systemc," in *DAC '06: Proceedings of the 43rd annual Design Automation Conference*. New York, NY, USA: ACM, 2006, pp. 911–914.

[9] M. Monton, A. Portero, M. Moreno, B. Martinez, and J. Carrabina, "Mixed SW/SystemC SoC Emulation Framework," in *IEEE ISIE, 2007.*, 2007, pp. 2338–2341.

[10] N. Kim, H. Choi, S. Lee, S. Lee, I. Park, and C. Kyung, "Virtual chip: making functional models work on real target systems," in *Proceedings of the 35th annual conference on Design automation*. ACM New York, NY, USA, 1998, pp. 170–173.

[11] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura, "A fast HW/SW co-verification method for SoC by using ac/c++ sim. and FPGA emu. with shared register comm." in *Proc. DAC*, 2004.

[12] R. Underwoood, "An Open Framework for Highly Concurrent Hardware-in-the-Loop Simulation," *Master's thesis, University of Misouri-Rolla*, 2007.

[13] 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 Transactions on Industrial Electronics*, vol. 54, no. 2, pp. 919–931, 2007.

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

[15] L. Benini, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Virtual in-circuit emulation for timing accurate system prototyping," in *ASIC/SOC Conference, 2002. 15th Annual IEEE International*, 2002, pp. 49–53.

[16] C. Koehler, A. Mayer, and A. Herkersdorf, "Chip Hardware-in-the-Loop Simulation (CHILS) - Embedding Microcontroller Hardware in Simulation," in *Proceedings of the 19th IASTED International Conference on Modelling and Simulation*. Acta Press Inc,# 80, 4500-16 Avenue N. W, Calgary, AB, T 3 B 0 M 6, Canada, 2008.

[17] U. Nageldinger, A. Pyttel, and H. Kleve. (2004) System simulation speedup combining systemc models and reconfigurable hardware. [Online]. Available: http://speac.fzi.de/WORKSHOP2/Speac_Paris_2004_01.pdf

[18] R. Ramaswamy and R. Tessier, "The integration of systemc and hardware-assisted verification," in *FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2002, pp. 1007–1016.

[19] 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

[20] P. Mantegazza, E. Dozio, and S. Papacharalambous, "RTAI: Real time application interface," *Linux Journal*, vol. 2000, no. 72es, 2000.

[21] K. Yaghmour, "Adaptive domain environment for operating systems," *Opersys Inc.*, 2001.

[22] (2009) Real-time linux wiki. [Online]. Available: http://rt.wiki.kernel.org/index.php/Main_Page

[23] T. J. Schriber and D. T. Brunner, "Inside discrete-event simulation software: how it works and why it matters," in *WSC '04: Proceedings of the 36th conference on Winter simulation*. Winter Simulation Conference, 2004, pp. 142–152.

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

[25] *BACnet - A Data Communication Protocol for Building Automation and Control Networks*, ASHRAE Std. 135-2004, 2004.