A Simulator based on QEMU and SystemC for Robustness Testing of a Networked Linux-based Fire Detection and Alarm System

Massimiliano D'Angelo*, Alberto Ferrari*, Ommund Ogaard†, Claudio Pinello‡ and Alessandro Ulisse*
*Advanced Laboratory on Embedded Systems (ALES S.r.l.), Rome, Italy
Email: {massimiliano.dangelo,alberto.ferrari,alessandro.ulisse}@ales.eu.com
†Autronica Fire and Security, Trondheim, Norway
Email: ommund.ogaard@autronicafire.no
‡United Technologies Research Center (UTRC), Berkeley, California, USA
Email: Claudio.Pinello@utrc.utc.com

***Abstract*—In this paper, we present a simulation framework based on SystemC and QEMU applied to the robustness testing of the communication layers in the AutroSafe fire alarm system from Autronica Fire and Security. This is a distributed large scale fire detection and alarm system. The system is based on state-of-the-art redundant protocols and embedded Linux operating systems. To achieve good confidence about the correctness and the robustness of the system, a simulation environment is used to run the target software using virtual execution platforms with timing variability. These platform variations allow simulating a range of system conditions, hence several possible interactions among the nodes and the network protocols, to which the application must be resilient. QEMU is used to execute the target software, while accurate models of the network protocols and topologies are provided in a SystemC-based simulation framework called DESYRE. The novel contribution described in this work is the integration of these environments for the simulation of a networked system with control of time execution accuracy and variability, as well as the application to the robustness testing of the industrial application.**

I. Introduction

Virtual engineering techniques are increasingly popular in several application domains. Models of the system components (seen as "virtual components") are used to assess the behaviour and performance of a subsystem unit without the need of a physical prototype. This capability is usually exploited

- in the specification of the subsystem components to compare different design alternatives and select the best design option.
- in the verification flow, where the satisfaction of requirements can be assessed on a virtual prototype. This approach allows starting the verification process earlier in the design flow so as to detect and correct errors that would otherwise propagate along the chain up to the implementation. Moreover, with respect to hardware, it is typically easier to access and modify the internal state of the virtual prototypes, so a comprehensive fault injection campaign and fault tolerance assessment can be carried out.

The larger adoption of these techniques is correlated with a development of modeling languages and tools to satisfy the requirements of designers. In the simulation domain, there is an evolution from simulation tools focused on particular system aspects to modeling environments able to capture large portions of the system at different levels of refinement. The development of system level modeling languages (e.g., SystemC [1]) and techniques for the integration of heterogeneous models/tools (co-simulation, hosted simulation [2]) are only few clues of this trend. Full system simulators [3] can be positioned in the same picture.

The distinctive feature of a full system simulator is the modeling of a system platform at a level of refinement that allows the execution of the actual software and operating system running on the modeled architecture. The computational complexity of the models is kept reasonably low to not prevent the simulation of large systems. In the domain of networked architectures, a full system simulator provides a notable improvement with respect to a traditional network simulator, enabling the execution of the real application and protocol stack on the node in place of more abstract traffic generators and protocol models.

In this paper a full system simulation environment based on QEMU and SystemC is proposed. The simulator is applied to the robustness testing of the communication layers in the Autronica AutroSafe product, a large scale fire detection and alarm system. The system is distributed and networked. The network is based on a redundant Ethernet architecture, managed by a proprietary protocol running inside the nodes on top of a Linux operating system. The virtual platform is used to run the target software to achieve a good confidence about the correctness and robustness of the system. QEMU runs the Linux operating system and software components deployed on the nodes, whereas an accurate model of the redundant network is captured in SystemC in the Desyre [4] modeling framework. The testing process on the simulator has been performed in parallel to the traditional verification procedures based on physical testbeds. The comparison between

virtual and physical testing environments has highlighted both benefits and limitations of the proposed simulation platform.

This paper is structured as follows. Section II summarizes the related contributions in literature and highlights the distinctive elements of the work. Section III describes the industrial case study, whereas the simulator environment is depicted in section IV. The AutroSafe system model is described in Section V, together with numerical results to characterize the model accuracy and simulation times. Section VI details how the simulator has been used for robustness testing of the AutroSafe system, and compares its capabilities with traditional physical testbeds.

II. Related works and contribution

Simulation technologies that can support the full system simulation of networked embedded systems are already available. Wind River Simics [5] is a commercial product with promising features in terms of modeling and support for analysis. System models can be defined in different languages, among which SystemC TLM2.0, and can leverage a rich library of CPU architecture emulators. The Open Virtual Platform (OVP) [6] provides models of CPUs which can execute binaries, together with other components like memories and buses, which allow the description of a computational architecture. OVP models can be integrated with external simulators or imported in a SystemC TLM2.0 modeling environment. Other solutions for full system simulation include M5 [7] and UNISIM [8], but they are at a less mature stage.

Simulators based on the integration of QEMU with SystemC have been proposed in literature. QEMU provides an open-source emulation platform, which can be modified to fit into the modeling framework and enhanced to suit the modeling requirements. The execution of the target code is based on dynamic translation and is hence faster than traditional instruction set simulators. This enables the execution of operating systems and large application processes, as well as the description of models with several instances of QEMU, without incurring in prohibitive slow down of the simulation. In [9], the PCI/AMBA bus in QEMU is interfaced with SystemC to model hardware peripherals. The interaction between the modeling environments is based on synchronous transactions, initiated by QEMU. QEMU is frozen while the SystemC simulation performs the transaction. A similar approach is proposed in [10] to integrate a GPU model with QEMU. A more refined integration between QEMU and a SystemC model of a peripheral is proposed in [11], where the communication with the external module is transparent to the peripheral driver running in the virtual machine. In [12], QEMU and SystemC are combined to implement a fast cycle-accurate instruction set simulator. The instructions executed in QEMU are simulated at the cycle accurate level in SystemC. QEMU sends to SystemC all the information (instruction types, register values, memory accesses) about the instructions to be executed.

A possible approach to use QEMU for networked system modeling is to resort to the Virtual Distributed Ethernet (VDE) project [13], which provides models to build a virtual ethernet network among the virtual machines. The network is interfaced with QEMU using native capabilities of the virtual machine. The models enable the exchange of network frames among the machines, but capture the behaviour of the network at a very abstract level, with little or no notion of time.

The original contribution of this paper is twofold:

- A simulator based on SystemC and QEMU is proposed. Differently from the works available in literature, the simulator addresses a networked system, in which several QEMU instances are coordinated by the SystemC scheduler. Both the QEMU and SystemC components can generate events in their internal time representation, and a suitable approach is needed to mantain a chronological relationship among these events.
- An application of the simulator for robustness testing of a state of the art fire detection and alarm system is described. The usage of the simulator in an industrial context allows exercising the simulator capabilities to a larger extent than with ad-hoc case studies. The analysis of the verification process of a safety critical system points out the role, the benefits and the limitations of a virtual engineering platform with respect to traditional physical testbeds.

III. Industrial application: Detection and Alarm system

The AutroSafe [14] product family is the high-end fire detection system in the Autronica Fire and Security [15] product portfolio. It targets the on-shore market (buildings, industrial facilities), the maritime market (ships) and the petrochemical oil&gas market. Similarly to other fire detection systems, an AutroSafe installation is composed of a set of Control and Indicating Equipments (CIE). A CIE is responsible for controlling a particular area of the installation plant. It manages the inputs received by a set of near-by detection devices (e.g., smoke detectors) and may trigger the activation of alarm and protection devices (e.g., strobes, sounders, fire doors, fans). Moreover, a CIE may have a user interface (LCD screen and keyboard) to show the status of the system and to get inputs from a fire security operator (e.g., reset an alarm, modify the system configuration).

CIEs communicate with each other to share their local status and coordinate their actions in the event of a fire. A fire detection in a particular area of a building, for instance, may require the activation of alarm devices all over the building, possibly in a phased order. In the AutroSafe version that was tested, the interconnection among the CIEs is guaranteed by a redundant ethernet network, where all the network components are duplicated to achieve tolerance to faults. The redundant connection is managed by a proprietary protocol, AutroNet. This network architecture is one of the most relevant novelties introduced in this system.

The functionalities to perform detection, alarm and protection functionalities, as well as the AutroNet protocol, are all implemented by software components running inside a CIE on top of a Linux operating system. In terms of computational

capabilities, each CIE has an Atmel AVR32 microcontroller as main computational unit.

In the design process of a new fire detection and alarm system, the testing of the software components takes a large percentage of time on the overall project duration. Errors in the software implementation might compromise the system functionalities and its reliability, and would be costly to be fixed when several installations have been already put in place. In Autronica, the testing procedure involves several steps and is performed on different testing infrastructures. Testing in the early development stage is performed running the software components on a standard Linux PC, used as an emulator of a stand-alone CIE. It provides a convenient testbed for the developers to verify their implementation on the same machine where the components are being developed. Nevertheless, testing is limited at this stage to functionalities which can be exercised on an isolated node, and may require some modification of the original code to remove any dependency from particular features of the actual hardware platform.

A more refined testing infrastructure is composed of a set of test boards, connected by an Ethernet network. Compared to the actual CIE, each test board has a microcontroller with the same instruction set architecture, different board characteristics and lower overall performance. Moreover, each board provides two ethernet network interfaces to setup the redundant connection. This testbed can properly exercise all the software components which exchange data on the inter-CIE link, as well as the AutroNet protocol which manages the network redundancy. In Autronica, engineers usually have a small scale testbed on their desks to perform some preliminary testing over a networked system. In addition, a single larger testbed with 64 boards is available for extensive tests.

In the latest stage of the development process, testing is completed on an exact replica of a system installation, where the test boards are replaced by the hardware of the actual CIE. A CIE provides the whole set of peripherals that must be managed by the software components, which can hence be fully executed without the need for stubs.

Robustness testing

Robustness is defined as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions [16]. Dependable systems must be proven to be robust.

The key to robustness testing is to develop test cases and test environments where the robustness of a system can be assessed. Testing is usually focused on software components, which are more likely to fail when unintended conditions occur. The test cases must cover standard and exceptional, but legal, operating conditions. The system must work when fed with intended and unintended inputs, in different operating scenarios and architectural configurations. Safety critical systems may also require a certain degree of robustness to fault conditions.

The verification process in Autronica includes an assessment of the system robustness. Test procedures describe sequences of operations to be performed on the system, and the expected status of the system after each operation. The actions performed include powering up the nodes in different sequences and with random delays, rebooting or shutting down a subset of the nodes, disconnecting some cables. Moreover, the system inputs are fed by hardware emulators of the CIE peripherals, which may generate critical sequences of data.

The system model that will be described in the next sections complements the testing infrastructure of the AutroSafe system by providing the chance to alter to a larger extent the platform on which the AutroSafe software components are running. A model is based on an abstraction of the system platform, and mimics its behaviour up to a certain extent. While a good correlation with the real system is desirable to position the model in the reference operating condition, perturbations of the execution flow, due to model approximations or injected intentionally allow testing the robustness of the system. A virtual testing platform provides more opportunities to introduce variability, and guarantees a higher observability with respect to a physical testbed. System architectures can be easily reconfigured in the virtual domain to capture alternative configurations; faults that are difficult to be injected in a physical system can be more easily introduced in the system model. The availability of probing facilities in the simulator allows inspection of the status of all the components in the system, and identification of unintended behaviours that might be latent on the physical testbed.

IV. Simulation framework description

The simulator framework is built by integrating the QEMU virtual machine with the Desyre simulation framework. QEMU executes the operating system and the application processes running on a node of the system. Multiple instances of QEMU are interconnected by a model of the network architecture captured in the SystemC language in the Desyre framework. The integration has required the definition of a specific interface between the two environments, so as to enable data exchange and execution coordination. In particular, the internal times in Desyre and QEMU are synchronized to maintain the correct chronological relationship between the events in the two different timing domains. More details will be provided in the following subsections.

A. QEMU

QEMU is an open-source machine emulator and virtualizer. It can emulate several instruction set architectures; currenty, the set of emulated architectures contains x86, PowerPC, Sparc32/64, MIPS, ARM and Coldfire. Emulation is based on dynamic (run-time) translation of basic blocks of the target CPU instructions into the host instruction set. QEMU has two operating modes:

- Full system emulation. In this mode, QEMU emulates a full system, including a processor and various peripherals. A QEMU system platform can be used to execute different operating systems.

- User mode emulation. In this mode, QEMU can launch processes (not OSes) compiled for one CPU on another CPU, whereas compatibility is needed between the application and the hosting operating system where QEMU is running.

To implement the full system emulation, QEMU provides a set of predefined system platforms, characterized by a particular instruction set architecture and by a set of emulated devices. The list of system platforms includes (but is not limited to) a PC system based on the x86 emulator, a PREP or PowerMac PowerPC system with a PowerPC cpu, a UltraSPARC machine based on sparc64, several boards based on ARM. The emulated devices available for each platform can be binded to host devices, redirected to files, or configured to be accessible from other processes in the host system, e.g., by sockets.

B. Desyre

The Desyre framework [4] is a SystemC-based virtual prototyping environment. In this framework, the virtual prototype is composed of a set of modeling components, instantiated out of model libraries which may cover functional and architectural aspects of the system (e.g., protocol models, communication channels, computational platforms). The virtual prototype is specified as a hierarchical netlist with a set of parameterized configuration input files, compliant to the IP-XACT format [17]. These files define and instantiate the model components, and connect them together to specify the model of the entire system. To facilitate the generation of the system netlists for complex designs and for the design space exploration, Desyre provides an exploration language (EL) to describe in a more concise form, as parameter sets and parameter relations, the different configurations to be simulated. The EL specification is used to automatically generate and parameterize the IP-XACT file sets, representing the selected scenarios. The designer has the freedom to run selectively the simulation of a scenario or of all scenarios as a batch exploration. A simulation is performed in three phases:

1) Netlist elaboration and component loading;
2) Simulation;
3) Data post-processing and visualization.

The first two phases are repeated for each of the chosen scenarios. In the elaboration and loading phase, according to the designer's netlists, the model builder of DESYRE dynamically creates in memory the system model by parsing the IP-XACT files and loading the required SystemC models (compiled and present in the library). In simulation, the created system model is executed and the output traces are produced. In the data post-processing, traces are analyzed to aggregate and/or verify the performance and the functional data.

The dynamic technique for the model creation facilitates the design space exploration by 1) removing the need for the compilation of the different SystemC netlists (the flow is compiler free); 2) enabling the designer to quickly derive additional scenarios by parameterizing the EL or IP-XACT files.

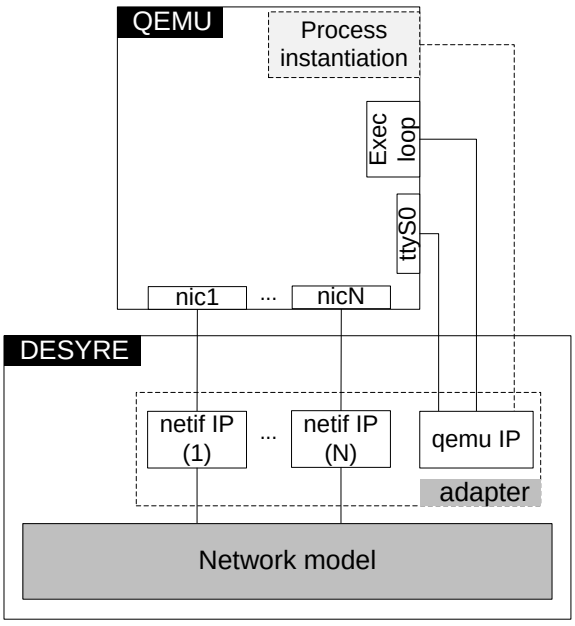


Fig. 1. Interface between QEMU and Desyre

C. Architecture of the Desyre - QEMU integration

Desyre and the QEMU instances run as distinct processes on a Linux host system. QEMU is configured in full system emulation mode, with a parametrized number of network interfaces and a serial port. The creation of the multi-process environment is managed through Desyre, and is based on a description of the system model in the IP-XACT metamodel. The interaction with an external QEMU process is managed by an adapter (Figure 1), composed of the following Desyre modules:

- A qemu IP, which is responsible for instantiating a QEMU process at the simulation start-up. During the simulation, the module schedules the execution of the QEMU instance and keeps the internal time of QEMU synchronized with the Desyre time. The module may also issue the execution of a command at a particular time in the console of the guest operating system, accessed by the emulated serial port (ttyS0 in Figure 1) connected to a socket. At the end of the simulation, it performs the shutdown of the virtual machine and collects the (eventual) logs generated inside the virtual machine and stored on the virtual disk. The configuration of QEMU (parameter values and emulated devices to be instantiated) is captured in the IP-XACT files, as parameters of the QEMU IP.
- A netif IP, which acts as an adapter between the emulated network interface card in QEMU and the model of the network in Desyre.

Notice that the partitioning of the functionalities in two different modules keeps the QEMU IP orthogonal from the particular model being interfaced with QEMU, and permits reusing this IP when different or additional parts of the systems (e.g., node peripherals) will be captured in Desyre.

The integration has required several modifications of the QEMU source code. The flow of execution of the target instructions has been modified to enable its suspension and resuming. More in detail, the virtual machine can be executed for an amount of virtual time ΔT before suspending: say t the internal time of the machine when execution is granted for ΔT time, the machine will suspend when its internal time will be $t + \Delta T$. The control of the execution flow is exposed by QEMU on a unix socket (exec loop in Figure 1) to which a QEMU IP is connected.

The emulated network card in QEMU can be natively redirected to a socket, where network frames can be exchanged with an external process. This mechanism is exploited, for instance, by VDE. However, the transmission of a frame in QEMU is modeled at a purely functional level, with no notion of time. Modifications have been introduced in the emulated card to expose the transmission delays assessed in the Desyre model to the virtual machine. In the modified QEMU, a delay is introduced in the virtual machine time between the request of a transmission and the interrupt which signals to the emulated CPU that the frame has been actually transmitted. The amount of time required for the transmission is assessed by Desyre and is notified to the virtual machine. Besides the transmission and reception of frames, the interaction between the emulated NIC and the Desyre model has been enhanced to synchronize the status and configuration of the cards (e.g., for an Ethernet network card, power on/power off state, promiscuous mode enabled/disabled, link state, multicast masks). The exchange of information is managed on the Desyre side by the netif IP, and is performed through a UNIX socket (nic1 to nicN in Figure 1). All the messages exchanged on the socket are tagged with a timestamp in the sender time.

The architecture of the simulator facilitates distribution of the processes involved in the simulation on a set of interconnected hosting machines, so as to build a distributed simulation platform. Communication among the simulation processes over the network may lead to longer simulation times. Techniques to mitigate performance penalties and enable large-scale highly-distributed simulations are under investigation by the authors in the context of the SPRINT EU project [18]. The first results on synchronization algorithms have been injected in the current version of the simulator, so as to allow the user to select the best trade-off between synchronization overhead and accuracy.

D. Time synchronization

The execution of each QEMU instance is controlled by Desyre to make time progress in the two environments in a consistent way. Two different approaches have been implemented in the QEMU IP to schedule the execution of the virtual machine:

- Fixed step execution. At time t_i, each QEMU instance is scheduled for execution for a fixed amount of time ΔT_f. When all the QEMU instances have reached time $t_i + \Delta T_f$, Desyre advances its time up to time $t_i + \Delta T_f$. At this point, a new scheduling round is started.

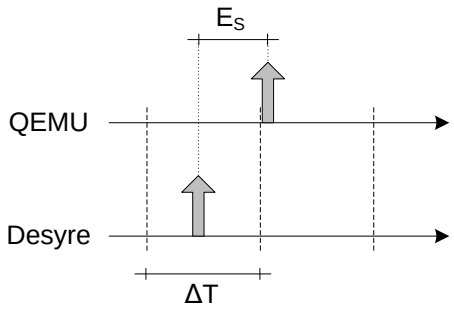


Fig. 2. Events generated by Desyre are scheduled at the beginning of the next scheduling slot in QEMU

- Variable step execution. At time t_i, the Desyre simulator retrieves the time of the next internal event t_{i+1}. QEMU is scheduled for execution for an amount of time $\Delta T_v = t_{i+1} - t_i$. When all the QEMU instances have reached time t_{i+1}, Desyre advances its time up to t_{i+1} (i.e., process the next event). The user can specify an upper and lower bound for ΔT_v: the upper bound avoids that the virtual machine and Desyre get largely out of synchronization; the lower bound prevents an excessive slow-down of the simulation.

The scheduling algorithm to be used can be selected by the simulator user.

Notice that in both scheduling algorithms, QEMU gets ahead in time with respect to Desyre. When QEMU passes a message to Desyre (e.g., a packet transmission), Desyre can properly schedule the processing of the message, because the message has been generated in the future with respect to its internal time. A message flowing on the other way will have a timestamp in the past with respect to the QEMU time, and will be processed at the beginning of the next scheduling slot (Figure 2). This process introduces an approximation (E_s) in the scheduling of the events (synchronization approximation), which can be reduced by using a smaller scheduling step (ΔT). The variable step solution, by adapting the value of ΔT to the time of the Desyre events, aims at stopping the virtual machine at the time instant when a message might be notified by Desyre to QEMU (the notification of a message is an event in Desyre). A numerical comparison of the two scheduling policies is provided in Section V-B.

V. System model

The architecture of the system model is depicted in Figure 3. A set of QEMU instances model the CIEs in the AutroSafe system. Since QEMU does not currently support the emulation of the AVR32 architecture, a x86 architecture emulator is used. The application processes of the AutroSafe system must hence be compiled for the x86 architecture to be executed on the simulator. These processes run in QEMU on top of a Linux operating system. The set of AutroSafe processes ported on the simulator includes the proprietary protocol, AutroNet, which manages the redundant connection. Each

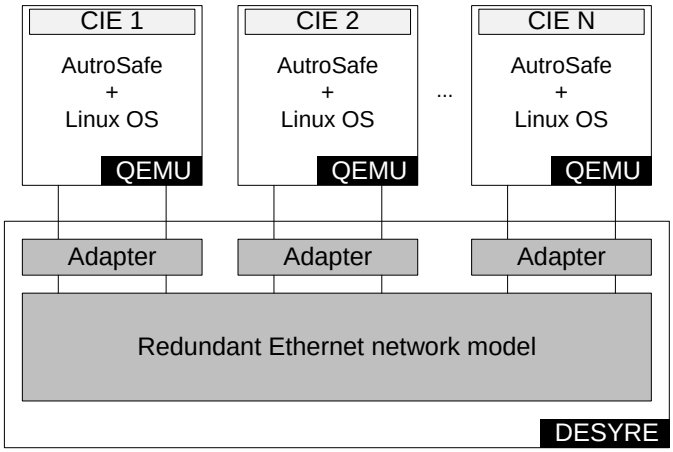


Fig. 3. Architecture of the AutroSafe system model

QEMU is configured to have two ethernet network interfaces. Some peripherals of the CIE are currently not captured by QEMU (e.g., interface for the loop of detectors and alarm devices). Desyre is exploited to build a model of a redundant ethernet network. The library of Ethernet network components in Desyre has been defined based on the IEEE802.3 standard, and covers both the MAC and PHY layers of the protocol. The models are defined at the TLM level of abstraction, and allow the simulator to achieve an accurate estimation of the most relevant network metrics, like latency, throughput and queue lengths.

An Ethernet frame flowing out of the network interface card of QEMU is injected through the adapter in the Ethernet network model, propagates through the network and is then passed to the QEMU virtual machine modeling the intended recipient.

A. Time model for software execution

The speed of execution in QEMU has been adjusted to roughly mimic the computational performance of the actual target platform. QEMU natively provides a parameter (icount) for specifying the amount of virtual time (in ns) that each instruction requires to be executed. Notice that the inverse of the icount gives the number of instructions per second (IPS) that the virtual machine can execute. When the icount parameter is set, the progress of the virtual time is correlated with the execution of the instructions.

The icount parameter has been calibrated to achieve the best correlation with the AVR32 CPU on the target. The CoreMark benchmark has been executed on the test board to get a reference value; then, the benchmark has been executed on QEMU to tune the icount value. The native icount value in QEMU is constrained to be a power of 2; to achieve a better resolution, QEMU has been modified to work with integer icount values. A time for instruction of 17 ns yields the benchmark score closest to the result on the test board. Section V-B will provide a comparison of the estimated execution times of the AutroNet process on the ”calibrated” QEMU with the measurements taken on the test board. Note that this calibration does not depend on the performance of the host but rather on the performance of the emulated (x86) platform.

B. Accuracy & performance assessment

The simulator has been exercised to characterize its correlation with the physical testbed and its performance in terms of ratio between simulation and simulated time. A system with two CIEs interconnected by a redundant network has been considered for comparison between the physical and virtual testbed. The physical testbed is composed of two test boards, connected by two Ethernet switches at 100 Mbit/s which serve the primary and secondary connections. The same system architecture has been modeled in the simulator.

Figure 4 compares the average execution time of the functions of the AutroNet process, assessed on the AVR32 microcontroller (physical testbed) and on QEMU with the icount parameter set to 17 ns (Section V-A). The values shown in the graphs are normalized with respect to the maximum function duration on the AVR32. Execution times have been measured by instrumenting the functions with the -finstrument-function option provided by GCC [19]. No better alternatives for function profiling are known to the authors that support both a x86 architecture and an AVR32 architecture. The perturbation introduced by the instrumentation on the function execution time is about 4 us. Binaries to be executed have been built with no optimization. Functions are partitioned according to their position in the call tree, where level 0 is the root (main function). Data about functions at level 2, level 3 and level 4 have been collected. A function at a particular level is identified on the graph by a function ID; a description of the associated function is not provided for non-discloure constraints. In this test, the simulator shows a good correlation with the physical testbed. The normalized error, computed as the difference between the execution time on QEMU and on the AVR32 and normalized with respect to the value on the AVR32, is always below 10 %. Similar correlation values are achieved by the other AutroSafe processes executed on the simulator.

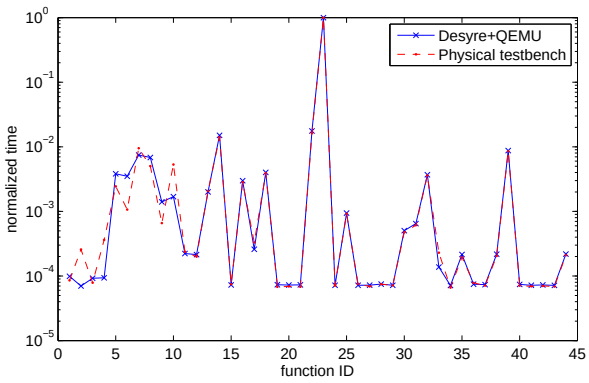
Figure 5(a) shows the maximum throughput achieved in a uni-directional communication between the two nodes in the system. The throughput has been measured with the IPERF benchmark [20], for both TCP and UDP transmissions over Ethernet frames with minimum and maximum payload. This test highlights a mis-correlation between the virtual and physical testbeds, where the virtual testbed achieves higher throughput values. The difference between them, expressed in percentage with respect to the actual throughput value, is below 70 % in almost all the scenarios, and reaches 230 % in the test with TCP and minimum payload. The same test has been repeated with a hub at 10 Mbit/s in place of the switches. The results in Figure 5(b) show a better correlation; the difference is below 20 % in almost all the scenarios, and reaches 55 % for the UDP transmission with minimum payload. This suggests that the different throughput values achieved by the two platforms are related to a mis-

correlation of the computational capabilities. The 10 Mbit/s scenario shows a better agreement because the throughput is bounded by the network, whereas the throughput at 100 Mbit/s expresses the different computational capabilities of the two platforms.

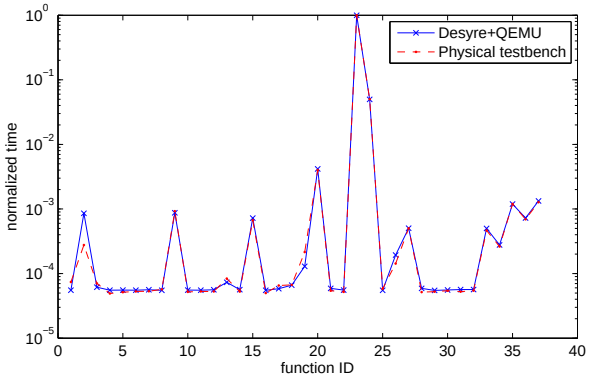
The results on the correlation of the function execution times and on the throughput provide a different assessment of the accuracy of the software time estimation of the simulator. This apparent inconsistency has been investigated carefully; the lack of a model of the cache in QEMU has resulted as the most likely cause of the observed behaviour. The score of the CoreMark benchmark used to measure the performance of the AVR32, and taken as the reference value to tune the QEMU icount, is highly affected by the cache; disabling the caching on the AVR32 reduces the score of more than one order of magnitude. The icount set to 17 ns hence models the average time for instruction when the AVR32 is exploiting the cache, and makes QEMU faster of the AVR32 when the target platform incurs frequent cache misses, as in the case of the reception of messages from the network [21]. The not very accurate model of the computational platform in QEMU will be assumed as an additional source of timing variability for the robustness test.

Table I shows the relation among the scheduling approach of the QEMU virtual machines, the ratio between the simulation time (real time) and the simulated time (time penalty), and the scheduling approximation (E_S) of the events notified by Desyre to QEMU (Section IV-D). A scenario with 16 nodes is considered. Each node runs the AutroSafe processes under test, and sends a message every 300 ms to all the nodes of the network and gets a reply back. The message is managed by AutroNet and flows on both the primary and secondary connections. The system activity is simulated for 5 minutes of virtual time, and includes the start-up process. The simulator is running on a workstation with 2x Quad-Core Intel Xeon @ 2.5 GHz and 48 GB of RAM.

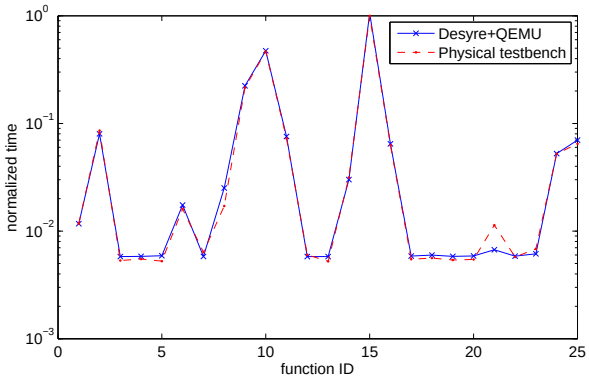
The dynamic step scheduling policy is compared with a fixed step approach, where the step size of the latter is set equal to the upper bound value of the former. The values 12 μs, 100 μs and 600 μs have been considered for comparison; the lower bound in the dynamic scheduling approach is always set to 2 μs. The real time required to run a simulation is affected by the size of the scheduling step, where a larger step provides a faster simulation at the cost of a higher error. Moving from the largest to the smallest scheduling step, the ratio between simulation time and simulated time moves from 2 (2 min of real time to simulate 1 min of system behaviour) to more than 70, whereas the error is reduced of about two orders of magnitude. The dynamic scheduling step usually requires longer simulation times, but provides a better accuracy with respect to the fixed scheduling step. The difference between the two approaches is emphasized in the scenario at 12 μs, where the adaptive behaviour increases the simulation time by about 25 %, but guarantees a reduction of the error of about 80 %. Simulation times for the dynamic step approach have also been measured in a scenario with 64 CIEs, in the same operating conditions. The configuration [2 μs, 100 μs] shows a time penalty ratio of 47, the configuration [2 μs, 600 μs] a penalty ratio of 10. The comparison of the results at 16 and 64 nodes suggests that simulation times scale almost linearly with the number of nodes in the system. Clearly, the average scheduling approximation error is not affected by the number of nodes being simulated.



(a)

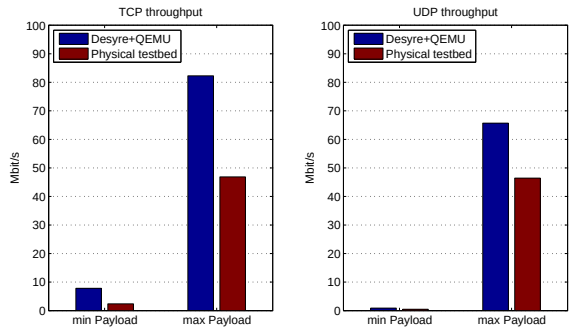


(b)

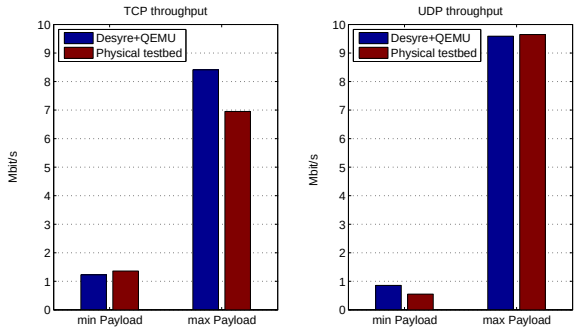


(c)

Fig. 4. Normalized execution time of functions of the AutroNet process at a) level 2, b) level 3, c) level 4 of the call tree (level 0 is the main function)



(a)



(b)

Fig. 5. Throughput on the point-to-point communication over a) a 100 Mbit/s switched Ethernet b) a 10 Mbit/s Ethernet with hubs

Scheduling	Time penalty	E_S
Dynamic time step [2 μs,12 μs]	87 x	1.4 μs
Fixed time step 12 μs	70 x	5.9 μs
Dynamic time step [2 μs,100 μs]	11 x	38.4 μs
Fixed time step 100 μs	9 x	57.3 μs
Dynamic time step [2 μs,600 μs]	2 x	351.8 μs
Fixed time step 600 μs	2 x	366.5 μs

TABLE I
SIMULATION TIME PENALTY AND SCHEDULING APPROXIMATION (E_S) FOR DIFFERENT SCHEDULING CONFIGURATIONS

VI. APPLICATION TO ROBUST TESTING

The simulator has been applied to the robustness testing of the communication layers in the AutroSafe fire alarm system. The AutroNet protocol and a set of software components which manage the status of the connection have been exercised on the simulator in different system configuration scenarios. At the current stage, the simulator provides a system model similar to the testbed. QEMU emulates the most relevant peripherals (network cards, serial ports) available on the test board, and all the configurations of the physical testbed (applications to be executed, network topologies) can be captured as well in the system model. Therefore, the test procedures defined for this physical testbed have been replicated on the simulator to speed up the maturation of the code. Notice that the simulator was not used as a substitute to the standard prototype and CIE testing for validation.

The behaviour of the software components has been observed in several operating conditions, like system start-up, starting the CIEs sequentially with a variable delay between them, in the event of CIE reboot, and under different traffic conditions (generated by a network test application running in QEMU on top of the AutroNet protocol). Different system scales have been considered, up to configurations with 64 CIEs. Moreover, the simulator infrastructure has been exploited to perform fault injection, introducing bit errors, packet drops, and connection drops on a variable subset of links. The software components have been analyzed by running the software through tools for memory checking, profiling and tracing, so as to support a review and optimization process of the source code.

A. Benefits of Virtual Prototyping

The usage of the simulator in the AutroSafe testing process has pointed out some benefits of virtualizing the testbed infrastructure. They are listed in the following:

- Larger exploration of system operating scenarios. The simulator can exercise the system in operating scenarios which are difficult to be replicated on a physical testbed. The injection of faults at the bit level, for instance, requires special hardware to be performed on the physical testbed. Events in the model of the system (e.g., the reboot of a node) can be precisely scheduled in time; it is hence possible to investigate the effect of two events occurring at the same exact time, or with a precise timing relation between them.
- Larger test automation. Testing procedures can be completely automated on the simulator, and run without human intervention. This guarantees repeatability of the tests, which can be useful for investigating the causes of a test failure. Moreover, it enables automatic regression testing either for robustness or system integration to support the software development.
- Increased observability. The usage of software models in place of hardware components allows inserting probes almost everywhere in the system. The features of the physical testbed prevent the execution of tools for dynamic software analysis, which currently have little or no support for the AVR32 architecture and have memory requirements which exceed the capacity of both the test board and of the final CIE. In the simulator, QEMU emulates a x86 architecture, which is supported by several tools; memory requirements can be fulfilled by setting the RAM of the virtual machine to a proper value.
- Extendability. The simulator has the potential to capture larger portions of the system. The introduction of models of the CIE IO interfaces and devices would bring the system model closer to the final system, and enable the simulator usage even in the latest verification phases.
- Flexibility. Each simulator instance can model different system configurations, in terms of, e.g., operating scenario, system scale, network topology. A configuration of the system model is stored in a set of XML files,

and can hence be quickly restored and shared among the simulator users. In addition, the modeling framework can accommodate any major change that the AutroSafe system might experience in the future; the library of models available in Desyre, for instance, will allow capturing alternative network architectures.
- Convenience. The virtual testing infrastructure can be easily replicated by installing the simulator on several workstations, without the need to buy dedicated hardware. Each developer can have a testing platform installed on its own machine, which can be easily accessed and can exercise the software to a larger extent.

VII. Conclusions and Future works

This work has proposed a simulation framework for networked embedded systems, based on QEMU and SystemC, which shows the capability to execute the same operating system and application processes running on the nodes of the network. The simulator has been used for robustness testing of a subset of the software components of the Autronica AutroSafe fire detection and alarm system. Numerical results on the accuracy and performance of the simulator have been provided. They show that the level of abstraction at which QEMU captures the computational platform makes simulation up to the maximum system scale affordable; simulation times can be tuned according to the required granularity of the synchronization between SystemC and QEMU. While the correlation between the actual architecture and the virtual machine in terms of software execution time is generally good (less than 10% error observed on application code) it may be poor in particular scenarios. In the industrial application, the simulator has provided a convenient testbed platform, with larger capabilities in terms of observability and exploration of operating scenarios than the physical testbeds.

Future enhancements of the simulator infrastructure will be devoted to support the design activity to a larger extent. The correlation between the simulator and a target platform will be improved, e.g., by the introduction of a cache model in QEMU, to increase the accuracy of the simulator and enable its usage for additional analysis (e.g., performance estimation). Moreover, Desyre supports design space exploration by providing a set of mechanisms such as parametric models and model composition functionalities to capture families of alternative system architectures, to simulate them and to compare their performance. The same flexible mechanisms can be used to capture particular system installations for feasibility assessment or to investigate post-installation issues.

A larger set of the AutroSafe software components is being ported for execution and testing into the virtual platform. The architecture of QEMU allows enhancing the virtual machine with models of peripherals that characterize a CIE, so as to solve dependencies of particular components with the actual hardware.

Acknowledgment

M. D'Angelo and A. Ferrari would like to acknowledge the support of the SPRINT EU project (grant agreement no: 257909).

References

[1] "SystemC," http://www.systemc.org.
[2] G. Döhmen, "SPEEDS Methodology - a white paper," http://www.speeds.eu.com/downloads/SPEEDS_WhitePaper.pdf.
[3] J. Engblom, "Full-System Simulation Technology," *Extended abstract in the proceedings of ESSES 2003 (European Summer School on Embedded Systems)*, September 2003.
[4] A. Mignogna, O. Ferrante, M. Carloni, and A. Ferrari, "A fully configurable rtos model for large scale distributed embedded systems simulations based on systemc," in *ASM 2011 - Applied Simulation and Modelling 2011*, 2011.
[5] "Wind River Simics," http://www.windriver.com/products/simics.
[6] "OVP - Open Virtual Platforms," http://www.ovpworld.org.
[7] "The M5 Simulator System," http://www.m5sim.org.
[8] "UNISIM: UNIted SIMulation environment," http://unisim.org.
[9] M. Monton, A. Portero, M. Moreno, B. Martinez, and J. Carrabina, "Mixed sw/systemc soc emulation framework," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, june 2007, pp. 2338 –2341.
[10] S.-T. Shen, S.-Y. Lee, and C.-H. Chen, "Full system simulation with qemu: An approach to multi-view 3d gpu design," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 30 2010-june 2 2010, pp. 3877 –3880.
[11] G. Di Guglielmo, F. Fummi, G. Pravadelli, M. Hampton, and F. Letombe, "On the functional qualification of a platform model," in *Defect and Fault Tolerance in VLSI Systems, 2009. DFT '09. 24th IEEE International Symposium on*, 2009, pp. 182 –190.
[12] T.-C. Yeh, G.-F. Tseng, and M.-C. Chiang, "A fast cycle-accurate instruction set simulator based on qemu and systemc for soc development," in *MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*, 2010, pp. 1033 –1038.
[13] "VDE - Virtual Distributed Ethernet," http://vde.sourceforge.net.
[14] "Autronica Fire and Security - AutroSafe 4 Product Brochure," http://www.autronicafire.com/utcfs/ws-445/Assets/AutroSafe4ProductBrochure.pdf.
[15] "Autronica Fire and Security," http://www.autronicafire.com.
[16] E. Iee, "IEEE Std 610.12-1990(R2002)," *IEEE Standard Glossary of Software Engineering Terminology*, 1990.
[17] "The SPIRIT Consortium's ESL-based IP-XACT 1.4 specification," http://www.spiritconsortium.org.
[18] "SPRINT EU Project - Software Platform for Integration of Engineering and Things," http://www.sprint-iot.eu.
[19] "GCC, the GNU Compiler Collection," http://gcc.gnu.org/.
[20] "IPERF," http://sourceforge.net/projects/iperf.
[21] W. Z. Aravind Menon, "Optimizing TCP Receive Performance," http://www.usenix.org/event/usenix08/tech/full_papers/menon/menon_html/paper.html.