Towards Hardware Embedded Virtualization Technology:

Architectural Enhancements to an ARM SoC

Paulo Garcia
Centro Algoritmi
University of Minho
Guimarães, Portugal
pgarcia@dei.uminho.com

Tiago Gomes
Centro Algoritmi
University of Minho
Guimarães, Portugal
tgomes@dei.uminho.com

Filipe Salgado
Centro Algoritmi
University of Minho
Guimarães, Portugal
fsalgado@dei.uminho.com

Joao Monteiro
Centro Algoritmi
University of Minho
Guimarães, Portugal
jmonteiro@dei.uminho.com

Adriano Tavares
Centro Algoritmi
University of Minho
Guimarães, Portugal
atavares@dei.uminho.com

ABSTRACT

Embedded virtualization possesses inherent challenges which differentiate the domain from traditional virtualization application fields such as server and desktop computing. Standard software virtualization solutions have a negative impact, not only on memory footprint and performance, but also on determinism and interrupt latency which are critical for the embedded real-time domain. Thus, efficient embedded virtualization requires domain-specific software and hardware support.

This paper presents work in progress results of hardware-based Hypervisor implementation. The use cases of embedded virtualization are analyzed, justifying the reasoning for hardware-supported virtualization. Architectural and micro-architectural improvements to an ARM v5TE processor are described, demonstrating the performance advantages, and compared against ARM Virtualization Extensions, identifying respective vulnerabilities and providing alternative solutions which enable higher flexibility, minimizing virtualization costs. The research roadmap towards a hardware-complete Hypervisor, based on the presented results, is described.

Categories and Subject Descriptors

C.3 [**Special Purpose and Application Based Systems**]: Microprocessor/microcomputer applications; Real-time and embedded systems; D.4.7 [**Organization and Design**]: Real-time systems and embedded systems

General Terms

Performance, Design, Experimentation

VtRES 2013 Workshop on Virtualization for Real-Time Embedded Systems
August 21st, 2013, Taipei, Taiwan

Keywords

Virtualization, Embedded Systems, FPGA, ARM

1. INTRODUCTION

The development of embedded systems has always been subjected to very tight constraints, e.g, power consumption, memory footprint [1]. With the emergence of complex embedded systems which increasingly display characteristics from general purpose computing, while still constrained by real time requirements (e.g., automotive systems, smartphones), as well as increasing security and safety concerns in the safety-critical domain [2], the designer's task has become exponentially more difficult. Embedded virtualization has emerged as a solution to address all the previously mentioned concerns [3]. Virtualization enables the co-existence of Virtual Machines (VM), allowing systems to partition concerns (real time vs rich applications, safety isolation, etc). However, the design of Virtual Machine Monitors (VMM or Hypervisor) for the embedded domain is a non-trivial task, especially in the case of safety-critical systems [4].

Efficient embedded virtualization is an emerging concern which spawned research at several levels. On the software side, the problem of CPU and I/O virtualization, which incurs in high overheads, has been significantly addressed in the literature. A multi-core virtualization layer has been presented in [5]. The approach identifies the problem of Memory Management Unit (MMU) virtualization, which causes significant overhead if emulated by software, and thus addresses the issue by implementing a virtualization composition kernel which enables guest operating systems to operate in kernel space, emulating only a minimal set of instructions. This approach is based upon new processor generations which fall outside the traditional virtualization model proposed by Popek and Goldberg [6]; further work on formal requirements for virtualization in new hardware generations has been addressed in [7] more recently. Focusing on I/O virtualization, which is typically the biggest performance bottleneck, much work has been developed in order to optimize virtualized systems. In [8], a virtual machine dedicated to I/O scheduling is used to coordinate requests among all other partitions. In [9], a different approach is

used, where a VMM allows partitions to directly access I/O, eliminating the need for software emulation; the Hypervisor merely coordinates accesses. Both approaches fall within the current trends in I/O virtualization presented in [10]: namely, full and para-virtualization, software emulation and bypass (direct) I/O. These approaches improve performance at the cost of determinism and higher development time, unsuited to the embedded domain, while the work presented in this paper attempts to keep software development time short by providing adequate hardware support without impact on real-time execution. This approach follows the roadmap expected on the hardware side: several processor architectures, namely ARM, the embedded market leader, already provide some sort of virtualization support on new-generation processors [11]. This support eliminates several ARM virtualization issues previously identified in [12]. The catapulting of Field Programmable Gate Arrays (FPGA) from mere prototyping to fully-fledged deployment platforms [13] has opened the way to novel hardware-software co-design approaches, namely for embedded virtualization technology. Projects such as BASTION [14] and HAVEN [15] demonstrate the advantages of this technology.

Most of the presented related work addresses limitations in embedded virtualization, which can be overcome if hardware support is available. In most cases, a great deal of Hypervisor software is required to cope with virtualization issues, negatively impacting performance and determinism. The presented work is concerned with architectural features, such as ARM virtualization support, which can simplify the design and increase the efficiency of embedded virtualization. Specifically, this paper presents work in progress towards the development of an embedded co-designed Hypervisor; it presents virtualization extensions to an ARM v5TE architecture and micro-architecture to reduce the required VMM software. This virtualization technology is described, detailing how processor, memory, peripheral and interrupt virtualization is performed and contrasted to ARM Virtualization Extensions (VE), and the preliminary results of the performance impact are presented. The main contribution of this work is the specification of architectural and micro-architectural features at System-on-Chip (SoC) level (encompassing processor, memory system and peripherals) to enable high-performance virtualization, paving the way for hardware implementation. The remaining of this paper is organized as follows: Section II presents the rationale behind the envisioned architecture for hardware embedded virtualization technology. Section III presents the current state of the research, describing the modifications performed on the SoC to reduce VMM software base. Section IV presents preliminary results on performance impact of the described technology, based on specific VMM functionalities. Section V presents the conclusions obtained from the currently available results and describes the roadmap for future research.

2. VIRTUALIZATION USE-CASES FOR EMBEDDED SYSTEMS

Virtualization in the embedded systems domain is an answer to a set of challenges quite different from the ones encountered on the desktop and server domains. Hence, the requirements and design hurdles are also substantially different and must be addressed by adequate, embedded-oriented solutions.

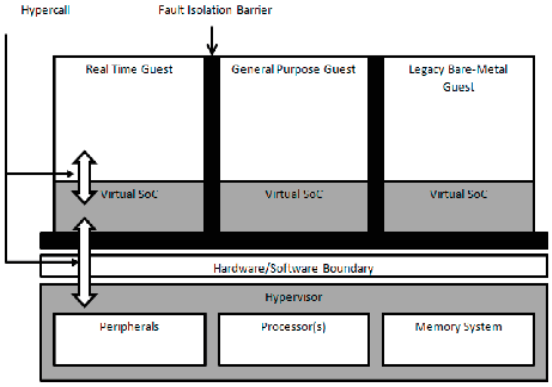

Figure 1: Hardware-Based Virtualization System View

In the server domain, the de facto application field for virtualization, VMMs are used for [3, 16]:

(a) **Service consolidation**, merging several physical machines onto a virtualized one;

(b) **Load balancing**, migrating VMs between hosts on-the-fly for efficient Quality of Service (QoS);

(c) **Power management**, by exploiting live migration to shut down low-load hosts;

On the embedded domain, however, the use-cases for virtualization technology differ:

(d) **Consolidation** of legacy software stacks with modern, feature-rich OSs [17, 18] (e.g., integrating legacy control systems with human-machine interface infrastructures);

(e) **Co-existence** of GPOSs and RTOSs [18, 19, 20] (e.g., designing systems with multiple contradictory requirements, such as smartphones);

(f) **Functionality and temporal partitioning**, in order to address security and safety concerns [21], namely fault isolation [22], and verification effort [23, 24];

Taking into account inherent characteristics of embedded systems, a case can be made against the need for use cases (a), (b) and (c): embedded systems are typically stable throughout deployment time, except for occasional firmware upgrades. Therefore, use case (a) does not apply. Use cases (b) and (c) assume multiple hosts; in the embedded domain, multi-core systems are typically heterogeneous [25]; application specificity allows load and power management to be coped with at design time. In the case of homogeneous multi-core platforms, typically used for computationally-intensive applications [26], use cases (b) and (c) may apply, but they will most likely conflict with real time requirements, thus should be carefully handled by the Hypervisor's core partitioning mechanisms [27] (i.e., finding the best tradeoff between static, semi-static and dynamic partitioning architectures).

Analyzing use-cases (d), (e) and (f), a set of conditions for embedded virtualization, namely in the safety-critical domain, is obtained: apart from efficient CPU, memory and

I/O virtualization, an embedded VMM must cope with real-time constraints; an embedded VMM must provide fault isolation between partitions and; an embedded VMM must not be a single point of failure. Embedded virtualization which does not comply with these requirements will cause: (1) Real-time failure. A virtualized RTOS will not be able to meet deadlines if VM time-partitioning which does not encompass real-time requirements is applied. (2) Fault propagation. A faulty partition may corrupt other partitions or the VMM itself, if no proper spatial-partitioning at all levels (processor, memory and peripherals) is employed. (3) Single points of failure. A corrupt Hypervisor (due to software bugs/malware or hardware soft errors) will cause the entire system to fail.

In function of these conditions, requirements for embedded virtualization may be postulated:

(1) An embedded VMM must be a Type-0 Hypervisor [28].

(2) As much as possible, Hypervisor software should be replaced by Hypervisor hardware.

(3) RTOSs must be at least partially paravirtualized (even in fully virtualizable architectures) or, alternatively, managed by the Hypervisor in a very specific way.

Fig. 1 depicts the envisioned architecture. The rationale for these requirements is an analysis of how virtualization is performed, under the light of the set of conditions previously specified for embedded virtualization.

A Type-2 Hypervisor runs on top of a GPOS. Therefore, it is incapable of providing the real-time requirements of the embedded domain. A Type-1 monolithic Hypervisor directly controls the hardware, thus representing a single point of failure. Type-1 microkernel Hypervisors (sometimes referred to as Type-1.5) such as Xen [29], rely on a specific guest OS to manage the hardware, again implementing a single point of failure. A Type-0 Hypervisor offers only the bare minimum to virtualize guest OSs, without support to hardware control. As such, a Type-0 implementation presents the smallest single point of failure out of VMM implementation options.

Minimizing the VMM software base to Type-0 means each guest OS will control hardware independently. While for processor virtualization, this can be easily coped with through ISA extensions for virtualization, the same is not true for peripherals. Virtualization hardware must be employed at system level to guarantee time and space partitioning throughout the entire SoC. This paradigm also allows for fault tolerance capabilities by hardware, minimizing Hypervisor vulnerability without degrading performance, as in the case of software fault tolerance. Although a hardware implementation is less flexible than a software one, this work assumes two postulates: (1) several architectural capabilities (such as ARM Virtualization Extensions and the features presented in the following section) do not decrease flexibility, since they merely provide mechanisms to simplify and expedite VMM software execution (thus they scale well when the number of partitions increases) and; (2) the use of FPGAs and Intellectual Property (IP) Cores to develop embedded systems opens the possibility for application-specific hardware support, where scalability is not an issue and hardware development time is amortized over implementations.

Assuring that real time guests are able to meet their deadlines requires partition scheduling specific to the guests' needs.

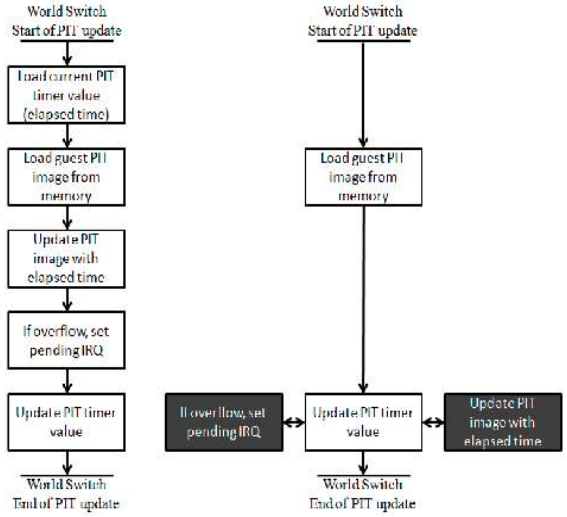

Figure 2: PIT timekeeping (a) without virtualization support; (b) with virtualization support (shaded regions indicate hardware operation)

In the simplest case, co-existence of one general purpose OS and one real time OS (which will be the most predominant case in consumer electronics), the embedded Hypervisor must guarantee that scheduling the general purpose partition will not interfere with the real time operation. In order to do this, either: (a) the RTOS is paravirtualized, using Hypercalls to inform the VMM of its scheduling needs or; (b) the VMM monitors the RTOS state (in a temporally non-intrusive way) in order to detect periods of idle task running, during which the GPOS can run. In the case of multiple real-time partitions, compositional scheduling [30] must be performed by the VMM, in order to ensure all guests are able to meet their deadlines, if schedulability cannot be performed statically; otherwise, traditional Time Division Multiple Access (TDMA) suffices [3, 30]. Compositional scheduling falls outside the scope of this paper.

3. TYPE-0 EMBEDDED VIRTUALIZATION TECHNOLOGY

This section presents the hardware architecture that supports a Type-0, hardware-biased Hypervisor. Due to its ubiquity in the embedded domain, the ARM architecture was chosen as a development target. More specifically, an ATMEL AT91SAM9XE, which implements the ARM v5TE instruction set, was cloned and implemented on a Virtex 5 FPGA in order to test the VMM architecture. Using a softcore (open-code) implementation was required in order to modify key processor internal components (e.g., decode logic) which would not have been possible with a COTS product.

Although the in-house implementation is (most likely) micro-architecturally different than the commercial one, the architecture is 100% compatible. Hence, results may not correspond exactly to equivalent implementations on the commercial one, but it is the authors' beliefs that micro-architectural differences will correspond to minor variations. For the purpose of fairness, all presented results are based on tests run on the in-house clone, with and without the

virtualization extensions.

Hardware embedded virtualization technology cannot be seriously considered, unless compared to ARM's virtualization extensions. The following technology specifications are compared to ARM's VE in terms of: how functionalities are supported by hardware, while ARM VE provides software support; and on performance improvements to ARM VE limitations identified in the literature [11].

3.1 CPU Virtualization

Identically to ARM VE, the proposed architecture extends the ARM modes with a special Hypervisor mode, more privileged than any other, and an instruction to enter this mode. Hypercalls are implemented through this instruction, which is considered privileged, trapping directly to guest OS if used in User mode. Hypervisor mode possesses its own group of banked registers, as well as additional control registers which contain the guest partition number (which may be required for future extensions to address peripherals or access the virtually tagged caches) and the exception handling register. The exception handling register allows configuring processor exceptions (e.g., undefined instruction) to trap to guest or to VMM. This register can be loaded upon each world switch in order to provide guest specific control, thus does not limit scalability or flexibility.

3.2 Memory Virtualization

ARM Virtualization Extensions supports two stage address translation: guest virtual to guest physical and guest physical to host physical, using a total of four levels of page tables. For the embedded systems use-cases (small, fixed number of VMs), memory segmentation between guests is likely to suffice. Therefore, the implemented virtualization technology simplifies guest segmentation by providing dedicated registers. No guest memory access traps to Hypervisor directly; instead, all accesses are mapped to the corresponding guest's memory segment by using the VMM registers Guest Address Base and Guest Address Top. Only on a segment violation is the Hypervisor invoked, trapping to a new VMM-only exception, Guest Memory Fault. This approach reduces the overhead in address translation compared to the architecture without virtualization support, and also eliminates the need for second page table walk presented by ARM VE. With the virtualization technology, a guest memory access can be translated without accessing the main memory for the VMM segment descriptor on each guest access (only at the starting point of each guest's time slice). Like the exception handling register, the segment control registers can be loaded upon each world switch in order to allow flexible segmentation. Number of segments is only limited by available physical memory and guests' needs. This approach is identical to KVM' s implementation for the Power Architecture [31], which benefits from having physically contiguous memory assigned to guests. Access to memory mapped peripherals is treated in a specific way.

3.2.1 Peripheral and Interrupt Virtualization

At this point, peripheral virtualization support has only been applied to the AT91SAM9XE's Advanced Interrupt Controller (AIC) and Periodic Interval Timer (PIT). The PIT is one of the system peripherals, responsible for the OS tick.

The PIT was extended with an additional Timer counter, accessible only in Hypervisor mode, which provides the VMM's tick. The HyperPIT can trigger an interrupt which bypasses the AIC, feeding the ARM core directly and causing a transition to Hypervisor mode, jumping to the VMM Scheduler interrupt vector. The standard PIT Timer counter was extended with update logic. In all non-Hypervisor modes, it behaves as traditionally. When writing the counter in Hypervisor mode, the loaded value is automatically added with the HyperPIT value, thus simplifying guest timekeeping when the VMM restores a guest context. If the value surpasses the overflow limit, a PIT interrupt is immediately dispatched to the guest. Whenever the ARM core transits to Hypervisor mode, the guest PIT components are automatically halted. This time-keeping behavior is depicted on Fig. 2. The Advanced Interrupt Controller was also extended with a Hypervisor protected register, the HyperIMR (Interrupt Monitoring Register), designed to facilitate interrupt virtualization. The AIC behavior when an interrupt is pending depends on the current value of the interrupt mask register (controlled by the currently running partition) and the value of the HyperIMR, which dictates how the AIC handles each interrupt source. If interrupt source x is pending and:

(1) AIC_IMR[x] clear and HyperIMR[x] clear: the interrupt is intended to another partition only, and will be handled when the Hypervisor schedules that partition. No action is performed by the AIC:

(2) AIC_IMR[x] clear and HyperIMR[x] set: the interrupt is intended to another partition, and must be serviced as soon as possible (real time requirement). The AIC causes the core to transit to Hypervisor mode, to the VMM Scheduler interrupt vector, so the Hypervisor can schedule the correct partition to handle the interrupt as soon as possible. This possibility may complicate temporal partitioning, but may be useful for certain applications which do not require strict temporal separation.

(3) AIC_IMR[x] set and HyperIMR[x] clear: the interrupt is intended for the current partition only, so the AIC triggers an interrupt which is handled by the running partition immediately.

(4) AIC_IMR[x] set and HyperIMR[x] set: the interrupt may be intended to one of several partitions, so the AIC causes the core to transit to Hypervisor mode, to the VMM Partition Interrupt vector, where software

Table 1: SYNTHESIS RESULTS - ARMV5TE CORE, 16KB ICACHE, 8KB DCACHE, MMU, PIT, AIC, DDRII CONTROLLER, USART, SD CARD CONTROLLER

Synthesis Results	Without Virtualization Technology	With Virtualization Technology
Slice Registers	25549 (36%)	26234 (37%)
Slice LUTs	43993 (63%)	44211 (63%)
Block RAM	108 (72%)	108 (72%)
Embedded DSPs	15 (23%)	15 (23%)
Clock Frequency	41.966	41.966

can decide to which partition the interrupt should be forwarded.

These additions to the Advanced Interrupt Controller give the VMM fine grained control over interrupt handling, allowing for highly efficient interrupt virtualization. ARM VE only allows global interrupt configuration, i.e., either all trap to Hypervisor, or all trap to guest (a very unlikely case [11]). The presented approach has the potential to offer greater performance benefits thanks to the flexible control. This implementation does not conflict with scheduling theory or temporal partitioning, since interrupt control is determined by the Hypervisor and specified at design time. Thus, it is not possible for an application from the General Purpose world to "claim" to be real-time and preempt the Real-Time world. Several ARM Virtualization Extension features, such as the System MMU for I/O, have not yet been contemplated in this work. The advantages these features provide are not ignored, but will be incorporated in future work, exploiting the possibilities for improvement.

4. PERFORMANCE RESULTS

The implemented virtualization technology was tested on a Xilinx ML505 board. Performance results were obtained through system simulation on Xilinx ISE development suite (ISIM simulator) and validated on-chip using Xilinx ChipScope. Table 1 displays synthesis results for area and clock frequency with and without the Virtualization Technology, as obtained from Xilinx ISE.

Performed tests compare software execution with and without the virtualization technology in terms of clock cycles. For both cases, tests were performed with caches disabled, thus represent worst-case execution in terms of memory access. As the Hypervisor software scales down, it will be interesting to explore dedicated local memory for its code in future work. When caches are enabled, Hypervisor software must first invalidate caches, since these are virtually indexed - virtually tagged. Clock cycles are measured since the first instruction is fetched until the last instruction finishes execution through the pipeline. At this point, results are only relevant to the execution of specific tasks, e.g., interrupt forwarding. No benchmarking has yet been performed, so it is not possible to determine how much the performance gain in each task will contribute to overall system performance. Performance results are displayed on Table 2.

The first task (Memory access address translation) assumes the VMM uses segmentation to separate guests address spaces. Without virtualization technology, the VMM must implement shadow page tables and manage the address translation through them. Results display Hypervisor updating of page tables after a page fault was caused by the Guest access (which happens only without Virtualization Technology). The second task (PIT partition context restore) measures the execution of adjusting the PIT timer value when the guest state is restored. Without virtualization technology, the VMM must read the current PIT value, add it to the guest PIT image in memory, update the PIT timer, and set the corresponding interrupt in case of PIT overflow.

The four interrupt cases refer to combinations of guest AIC_IMR and HyperIMR (case 00 refers to AIC_IMR clear and HyperIMR clear, respectively). Results display the time since the AIC hardware started handling the interrupt, until:

Table 2: Virtualization Technology Performance Results

Tested task	Without Virtualization Technology		With Virtualization Technology	
	Number of VMM instructions	Clock Cycles	Number of VMM instructions	Clock Cycles
Memory access address translation (segmented)	27	756	0	0
PIT partition context restore	15	424	2	56
Interrupt case 00	0	0	0	0
Interrupt case 01	239	6692	0	12
Interrupt case 10	NA	NA	0	12
Interrupt case 11	0	12	0	12

the guest resumes execution (case 00); the Hypervisor enters its scheduler to determine the guest to which the interrupt should be forwarded (case 01); the current guest enters its ISR (case 10) and; the Hypervisor enters the VMM Partition Interrupt vector to decide on which guest should receive the interrupt (case 11). For the cases without virtualization technology, cases refer to guest (virtual) AIC_IMR and real AIC_IMR.

5. CONCLUSIONS AND FUTURE WORK

This paper presented work in progress towards the development of an embedded co-designed Hypervisor, biased towards hardware. Specifically, the implementation of virtualization support on an AT91SAM9XE ARM v5TE was described, specifying additions to the processor core, Periodic Interval Timer, Advanced Interrupt Controller and memory system. The rationale behind the research was presented, explaining why certain design decisions were taken, in the context of safety-critical embedded systems.

The implemented virtualization technology allows decreasing the required VMM software and, as demonstrated by preliminary results, results in performance increase. Results have only shown the performance gains for specific VMM functionalities, as no system-level testing has been performed at this point. As previously mentioned, micro-architectural differences will account for result variations in other processor implementations, but it seems highly unlikely it will cause erroneous results. Some flaws in the ARM VE were identified, namely on the interrupt virtualization, and an approach to offer greater flexibility and higher performance was presented.

Work in the near future will focus on performing system level testing: specifically, a Linux image will be run in parallel with a RTOS (supported by an OKL4 Hypervisor) and all VMM execution will be characterized in order to iden-

tify possible performance improvement points. ARM VE's support to these points will be analyzed, in an attempt to identify further limitations.

Research will continue towards the migration of Hypervisor code to hardware, leaving enough flexibility in the software side to tackle application variability. Taking advantage of FPGAs and, knowing system requirements at design time, application-specific virtualization technology, encompassing full and para-virtualization mechanisms for processor, memory, interrupts and peripherals will be developed. The ultimate goal is to allow fully-virtualized guests to run at near native performance with minimum guest modification by taking advantage of the architectural support; ideally, defining novel architectural and micro-architectural features for virtualization support in future SoC architectures. These will most likely include hardware-based partition scheduling, Hypervisor-level hardware interrupt handling (in parallel with guest software execution) and peripheral specific virtualization support, such as described for the PIT and AIC in the presented test architecture. Fault Tolerance mechanisms will be explored, measuring the tradeoffs between software and hardware FT at Hypervisor level.

6. ACKNOWLEDGMENTS

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the Project Scope: PEst-OE/EEI/UI0319/2014.

7. REFERENCES

[1] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden, "Programming models for hybrid fpga-cpu computational components: a missing link," *Micro, IEEE*, vol. 24, no. 4, pp. 42–53, July 2004.

[2] F. Reichenbach and A. Wold, "Multi-core technology – next evolution step in safety critical systems for industrial applications?" in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, Sept 2010, pp. 339–346.

[3] G. Heiser, "The role of virtualization in embedded systems," in *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*, ser. IIES '08. New York, NY, USA: ACM, 2008, pp. 11–16. [Online]. Available: http://doi.acm.org/10.1145/1435458.1435461

[4] C. Jeffery and R. Figueiredo, "A flexible approach to improving system reliability with virtual lockstep," *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 1, pp. 2–15, Jan 2012.

[5] T. Nakajima, Y. Kinebuchi, A. Courbot, H. Shimada, T.-H. Lin, and H. Mitake, "Composition kernel: A multi-core processor virtualization layer for highly functional embedded systems," in *Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on*, Dec 2010, pp. 223–224.

[6] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, Jul. 1974. [Online]. Available: http://doi.acm.org/10.1145/361011.361073

[7] H. Dong and Q. Hao, "Extension to the model of a virtualizable computer and analysis on the efficiency of a virtual machine," in *Computer Modeling and Simulation, 2010. ICCMS '10. Second International Conference on*, vol. 2, Jan 2010, pp. 503–507.

[8] Y. Guo, X. Wang, W. Dong, G. Shi, and Y. Li, "A cooperative model virtual-machine monitor based on multi-core platform," in *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, vol. 1, May 2010, pp. V1–802–V1–807.

[9] Z. Li, L. Xiao, and L. Ruan, "A novel hardware-assisted virtualization approach for network interface card," in *Research Challenges in Computer Science, 2009. ICRCCS '09. International Conference on*, Dec 2009, pp. 225–228.

[10] B. Zhang, X. Wang, R. Lai, L. Yang, Y. Luo, X. Li, and Z. Wang, "A survey on i/o virtualization and optimization," in *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*, July 2010, pp. 117–123.

[11] P. Varanasi and G. Heiser, "Hardware-supported virtualization on arm," in *Proceedings of the Second Asia-Pacific Workshop on Systems*, ser. APSys '11. New York, NY, USA: ACM, 2011, pp. 11:1–11:5. [Online]. Available: http://doi.acm.org/10.1145/2103799.2103813

[12] A. Suzuki and S. Oikawa, "Implementing a simple trap and emulate vmm for the arm architecture," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, vol. 1, Aug 2011, pp. 371–379.

[13] H. Guzman-Miranda, L. Sterpone, M. Violante, M. Aguirre, and M. Gutierrez-Rizo, "Coping with the obsolescence of safety- or mission-critical embedded systems using fpgas," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 3, pp. 814–821, March 2011.

[14] D. Champagne and R. Lee, "Scalable architectural support for trusted software," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, Jan 2010, pp. 1–12.

[15] "High assurance virtualization engine (haven)," Polytechnic Institute of NYU, Final Technical Report, May 2009.

[16] M. J. Z. F. Bazargan, C. Y. Yeun, "State-of-the-art of virtualization, its security threats and deployment models," *International Journal for Information Security Research (IJISR)*, vol. 2, September/December 2012.

[17] F. Armand and M. Gien, "A practical look at micro-kernels and virtual machine monitors," in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, Jan 2009, pp. 1–7.

[18] A. Aguiar and F. Hessel, "Embedded systems' virtualization: The next challenge?" in *Rapid System Prototyping (RSP), 2010 21st IEEE International Symposium on*, June 2010, pp. 1–7.

[19] G. Heiser, "Virtualizing embedded systems - why bother?" in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, June 2011, pp. 901–905.

[20] G. "Heiser, "Hypervisors for consumer electronics," in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, Jan 2009,

pp. 1–5.

[21] A. Acharya, J. Buford, and V. Krishnaswamy, “Phone virtualization using a microkernel hypervisor,” in *Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference on*, Dec 2009, pp. 1–6.

[22] T. Gaska, B. Werner, and D. Flagg, “Applying virtualization to avionics systems 2014; the integration challenges,” in *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, Oct 2010, pp. 5.E.1–1–5.E.1–19.

[23] S. Edwards and E. Lee, “The case for the precision timed (pret) machine,” in *Design Automation Conference, 2007. DAC ’07. 44th ACM/IEEE*, June 2007, pp. 264–265.

[24] A. Hansson, K. Goossens, and M. Bekooij, “Compsoc: A template for composable and predictable multi-processor system on chips,” in *Transactions on Design Automation of Electronic Systems*, p. 2009.

[25] J.-C. Yeh, K.-M. Ji, S.-W. Tung, and S.-Y. Tseng, “Heterogeneous multi-core soc implementation with system-level design methodology,” in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, Sept 2011, pp. 851–856.

[26] S. Jena and M. B. Srinivas, “On the suitability of multi-core processing for embedded automotive systems,” in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on*, Oct 2012, pp. 315–322.

[27] K. Lakshmanan, R. Rajkumar, and J. Lehoczky, “Partitioned fixed-priority preemptive scheduling for multi-core processors,” in *Real-Time Systems, 2009. ECRTS ’09. 21st Euromicro Conference on*, July 2009, pp. 239–248.

[28] LynuxWorks, “"low-level & boot-level rootkits revisited"," "White Paper - http://www.slideshare.net/aziv69/whitepaper-lynx-secure-rootkit-detection-protection-by-means-of-secure-virtualization, Tech. Rep.

[29] S. Xi, J. Wilson, C. Lu, and C. Gill, “Rt-xen: Towards real-time hypervisor scheduling in xen,” in *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, Oct 2011, pp. 39–48.

[30] J. Yang, H. Kim, S. Park, C. Hong, and I. Shin, “Implementation of compositional scheduling framework on virtualization,” *SIGBED Rev.*, vol. 8, no. 1, pp. 30–37, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1967021.1967025

[31] S. Yoder, “Kvm on embedded power architecture platforms,” *KVM Forum*, Vancouver, Canada - August 2011.