Are there good Reasons for Protecting Mobile Phones with Hypervisors?*

Christian Gehrmann, Heradon Douglas and Dennis Kengo Nilsson
Swedish Institute of Computer Science (SICS)
Isafjordsgatan 22, SE-164 29 Kista, Sweden

***Abstract*—Security threats on consumer devices such as mobile phones are increasing as the software platforms become more open and complex. Therefore, hypervisors, which bring potential new secure services to embedded systems, are becoming increasingly important. In this paper, we look into how to design a hypervisor-based security architecture for an advanced mobile phone. Key security components of the architecture have been verified through a hypervisor implemented on an emulated ARM platform. We compare the hypervisor security architecture with TrustZone and summarize the major benefits and limitations of the hypervisor approach. In short, hypervisors exhibit several advantages such as support of multiple secure execution domains and monitoring of non-trusted domains; however, this comes at the cost of larger legacy system porting efforts.**

I. INTRODUCTION

Security has started to become a major differentiator when it comes to the design of consumer electronics such as mobile or smart phones, media players, residential gateways and various networked sensors. While security was previously an issue mainly for personal computers and network infrastructure, it is now equally important in order for end-users to be confident in using any kind of consumer electronics.

Security concerns for consumer embedded equipment range from reliability (high uptime, robust execution, network access etc.) to protection from software attacks such as viruses and trojans. In personal equipment like mobile phones, users also rely on their devices to handle sensitive information, such as online bank credentials or access keys.

What we currently see is an explosion in the embedded software domain with respect to the number of applications and usage of open software. Open software platforms and operating systems also give more freedom and power to attackers, as source code documentation and common hacking tools are available.

Hence, one would expect an increased threat to consumer electronic and embedded systems in general. This is indeed happening, and we see a boost in mobile viruses and network attacks particularly targeting mobile devices or sensitive infrastructure embedded devices [1].

Moreover, large open software systems subject to frequent updates are increasingly being expected to run on mobile devices. In order to protect such a system, there is a strong need for *partitioning* in order to isolate security-critical functions from non-security-critical functions as well as *monitoring* the security properties of the system.

A hypervisor or a Virtual Machine Monitor (VMM) runs at the most privileged execution level on a consumer device and, with the help of the basic hardware protection mechanisms available on most platforms, is a powerful approach to provide both secure *isolation* and *monitoring* services.

The major contributions of this paper are the following.

- We have analyzed the mobile phone scenario and based on the analysis we state a set of requirements for a hypervisor-based security architecture.
- We have designed a hypervisor-based solution, simulated it in OVP and show that multiple isolated secure services can be offered on a mobile platform with low overhead.
- We provide a comparison between TrustZone [2] and hypervisor architecture on ARM and show that the hypervisor solution exhibits several advantages.

The paper is organized as follows. First we give an overview of related work and the role of hypervisors in consumer electronic devices. Next, in Section III, we discuss a hypervisor-based security architecture for a mobile platform, focusing on the motivations behind the design. In Section IV, we compare the suggested hypervisor-based protection approach with a system built upon the ARM TrustZone architecture. Finally, we conclude in Section V.

II. BACKGROUND AND RELATED WORK

A. Virtualization

Virtualization through the usage of hypervisors[1] is an old technology with origin from IBM in the sixties [4]. While the technology almost was abandoned during 1980s and 1990s, the situation was drastically changed about ten years ago when VMware [5] introduced virtualization by binary translation (for the x86 architecture). Currently, we see a formidable virtualization usage boost in the computing industry. The motivations for introducing virtualization are many:

- Improved system utilization - multiprogramming
- Simplified system migration

* The research leading to these results has been conducted in the SVaMP project and received funding from Vinnova under grant DNr: 2009-01753.

[1] The virtualization technology we discuss here is the approach when a complete software system (including OS) runs on top of a hypervisor. This gives the illusion to the guest system of actually running directly upon the real hardware and it is often also referred to as *system virtualization* [3].

- Reduced downtime at system upgrade
- Running heterogeneous systems (multiple OSes) on a single machine
- Running legacy applications on new hardware
- Providing secure execution environments or secure monitoring

Among these, simplified system migration and reduced downtime do not really fit into the context of most embedded systems. Furthermore, as pointed out in [6], with the current trend towards usage of multicore chips, heterogeneous operating system support becomes a less viable argument for virtualization as each core or a dedicated set of cores in such a system can be configured to run a particular OS (as long as the memory can be physically partitioned). The system utilization, hardware abstraction and not least the security arguments for hypervisors are still viable even on multicore chips.

Virtualization can be achieved using a hypervisor with different approaches such as binary translation, paravirtualization [2], or hardware-assisted virtualization, supported by Intel and AMD processors on the x86 architecture. Currently, advanced hardware support for virtualization is still lacking in most embedded architectures, making paravirtualization or binary translations the most viable approaches. However, hypervisors are not the only way to achieve virtualization; an alternative approach is through using a microkernel such as the OKL4 [7].

B. Hypervisors for security

Multiple systems using hypervisors as enablers for security services (although not targeting embedded systems) have been suggested and designed in recent years. Kernel integrity protection of a guest system (the system running on top of the hypervisor) was implemented in [8], SecVisor, and in [9], $UCON_{KI}$. The single-guest SecVisor system prevents execution of any non-approved code in kernel mode, whereas $UCON_{KI}$ presents a more flexible and complex access control model governing system subjects (processes and kernel modules) and kernel resources (including memory and registers) based on rights, attributes, and events with dynamic continuity. The Overshadow system [10] and the "Software-Privacy Preserving Platform" [11] protect applications from their OS and each other via memory multi-shadowing – the hypervisor exposes an encrypted view of application process memory to all other processes, even kernel processes. Another hypervisor-based security service was suggested in [12], the Patagonix system. In that system the hypervisor is used to check all pages to be executed by comparing them to known "good binaries".

C. Virtualization for consumer devices

Virtualization as an enabler for security in mobile devices was discussed in [13] and the authors also showed how to run multiple protected OSes on single and multicore systems, using the Symbian EKA-2 Nanokernel and the L4 Microkernel respectively. Even if the authors recognize the security potential of virtualization, their approach is a common virtualization solution and they do not really make any analysis of *how* hypervisors should be designed or used on mobile devices when they are introduced to *serve security needs*.

In [14], a comparison with focus on consumer electronics is made between the microkernel- and hypervisor-based approaches to virtualization. The main topic of that study was however not security (even if the authors recognize the potential security benefits with the isolation provided by hypervisors). In particular, they make performance comparisons between a Virtual Logix hypervisor [15] and OKL4 "pistachio" running on an ARM11 platform with Linux as the guest operating system. The overhead and latency benchmark figures speak in favor for the hypervisor-based approach. However, the results have been criticized for not giving a fair comparison as a non-optimized version of OKL4 was used [16]. Regardless, using a microkernel to achieve virtualization typically requires more adaptations in the guest system compared to a hypervisor approach, since a hypervisor more often attempts to emulate traditional interfaces. From a security perspective, L4 has an advantage in that the kernel has undergone formal verification [17].

One of the most widely used virtualization solutions in data centers is the open source Xen project [18], which originates from Cambridge Computer Laboratory, supported by many major computer manufacturers. In [19], a port of Xen to an ARM926 mobile platform was demonstrated. Although that work shows that it is indeed feasible to run Xen on a rather constrained platform, it does not say much about what you achieve quantitively from a security perspective. Indeed, a direct port inherits vulnerabilities present in the large Xen code base [20].

A completely different way to isolate execution environments is through pure hardware-based separation. This approach is taken in the ARM TrustZone architecture [2]. ARM TrustZone technology, for ARM11 and ARM Cortex embedded processors, offers support for creating two securely isolated virtual cores (or "worlds", as they are termed) on a single real core. One world is Secure and the other is Normal. TrustZone manages transitions between them through hardware interrupts and the so called "monitor" mode, preventing state or data from leaking from the Secure world to the Normal world. System hardware, including memory and peripherals, can be allotted to each world.

III. HYPERVISOR-BASED SECURITY ARCHITECTURE

Next, we look into the potential security services a hypervisor could offer to a mobile platform and the major requirements for such services. We also look into a sample system with two CPU cores.

[2] In paravirtualization, the actual guest code is modified to use a different interface that is either safer or easier to virtualize, improves performance, or both.

A. *Security services and requirements*

Considering current security needs, clearly the main "security service" one can expect from a hypervisor layer is isolation between execution environments. However, what is *even more useful* on a mobile platform is the possibility to *combine* an open codebase rich-application domain and a real-time domain with a trusted execution domain on a shared CPU, which is exactly the main idea behind the ARM TrustZone architecture. This capacity, together with a simple and well-defined secure interface allowing applications in the open domain to use secure services in the trusted domain, is top priority. However, taking this one step further, a single entity providing secure services is not what one typically would expect in a mobile phone. Instead one would like to allow *multiple stakeholders* (operator, banks, enterprise IT administration, media provider etc.) to run their secure services *independently* from each other on the very same device [21]. Different from an ordinary virtualization scenario, these secure services would not necessarily need to run in parallel, actually making the isolation-providing hypervisor layer *simpler* than a typical hypervisor design – such as one based on the requirements coming from a consolidating server system.

However, the possibility to isolate a set of secure services is not enough; it must also be possible to provide this isolation *without substantially modifying* the existing code used in the open and closed parts of the system. If the design cannot meet this requirement, the hypervisor will most likely not be introduced into the system, independent of its security merits.

In all, we would typically require the following from a hypervisor-based mobile platform system architecture (not an exhaustive list):

- It shall be possible to provide the most security-critical functions in the system in multiple isolated, well-protected trusted execution domains.
- Small Trusted Computing Base (TCB) and evidence for that the TCB is sound.
- Communication from untrusted domains in the system towards trusted domains shall only be allowed through well-defined secure interfaces.
- It shall be possible to provide evidence that no information except the information provided through the secure interfaces can flow through the system (including all kinds of potential leakage through peripheral units).
- It shall not be possible for any application or process running in an untrusted or different trusted domain to interfere with the process currently running in a particular trusted domain.
- Execution in trusted domains should utilize on-chip memory and only rely on encrypted and integrity-protected data on external memories.
- It shall be possible to port a legacy software system with minimal efforts (few software changes).

As a second priority, a secure monitoring/enforcement service for an open "semi-trusted" domain is an attractive security service. The kernel integrity protection service offered through the previously mentioned SecVisor [8] is a good example. One can also think of other more advanced monitoring services such as security policy enforcement. However, moving too far in this direction could make the hypervisor more complex and radically increase the TCB of the system.

The functionality needed to actually provide isolation will to a very large extent depend on the available hardware characteristics and protection mechanisms like MMU, CPU modes and registers, page tables, interrupt controller functions, DMA functions and so on in the embedded system. Actually, as embedded systems do not have standard hardware architecture, the hypervisor design must be closely adapted to the actual platform it is supposed to run on. Also, when designed for security purposes, most functionality you could expect in a hypervisor architecture for a desktop or server would *not* be needed. We illustrate this point and our design strategy by looking into a sample architecture on a System on Chip (SoC) with two CPU cores in the next subsection. We have verified key parts of this architecture on an ARM926 through an OVP [22] emulation platform.

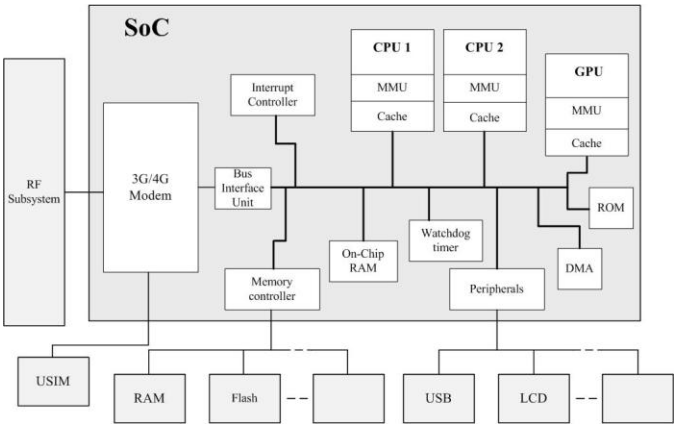

Figure 1. Schematic view of a single chip mobile platform with two CPUs, modem and a Graphical Processing Unit (GPU).

B. *Mobile platform hypervisor security architecture*

Consider the SoC architecture in Figure 1 which reflects a typical architecture one could expect in a modern smart phone [23]. The two CPUs run an application OS such as the Linux-based Android. In this system, with respect to security, we need to protect security-critical resources (on-chip memory etc.) from illegal accesses from potential hostile resources. Hence accesses from the modem and GPU must be restricted. We assume that the MMU in the GPU can be *statically* configured to protect vital parts at boot time and also that we have physical partitioning or hardware protection restricting access to security sensitive resources on the platform from potential hostile units such as the modem.

Taking the requirements in the previous section into account, if we choose to protect the execution on this platform

through the introduction of a hypervisor layer, we would like to achieve the following:

- Allow multiple trusted execution environments to run on one of the two available cores with secure interfaces exposed to untrusted domains.
- Protect the trusted execution environments and their data from all processes running on the same and other cores in the system as well as physical attacks on external memory.

In addition, we would like to offer integrity protection of the application OS kernel running on the cores in the system. Combining these together results in the architecture depicted in Figure 2. A prerequisite for a high level of security is that the system boots into a secure state. The two hypervisor layers on the two CPUs are running in the most privileged mode on the system and get control at start-up. Once all security initializations are complete, the hypervisors hand over execution to their respective "guest systems". The security functionality needed from the two hypervisors in the system is not symmetric. The main purpose of Hypervisor 2 is to protect security-critical resources. Hence, it is only needed to provide secure control of the MMU, interrupt controller, watchdog timer and DMA in the system. All other functions (including device drivers etc.) can be kept untouched, which implies that the performance impact stemming from the hypervisor is very limited. In addition, Hypervisor 2 should also provide low-overhead kernel protection (similar to SecVisor) for its Android guest.

Hypervisor 1 is more advanced as it needs to support multiple execution environments. It differs from a "classical" virtualization case though as the various services (the Secure Services in Figure 2) do not necessarily need to run in parallel, making the hypervisor overhead much smaller. Similar to Hypervisor 2, Hypervisor 1 should provide integrity protection of the Android kernel. The secure services running in the trusted domains are made available to the applications through dedicated hypercalls implemented in the hypervisor layer.

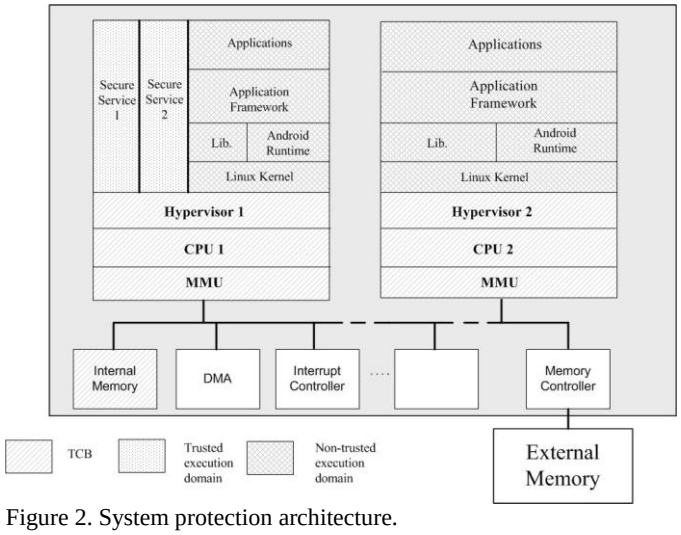

Figure 2. System protection architecture.

Preferably, the services running in the trusted execution environments should be loaded by Hypervisor 1 or trusted boot code into the on-chip RAM in the system. The secure services should then make sure to only read and write encrypted and integrity-protected data from and to the external memories. This gives a basic level of protection against physical attacks on the system.

In order to verify this architecture we have implemented a simple hypervisor layer running above an OVP-simulated ARM926ejs core. As the main purpose was to verify the hypervisor layer and its security properties, our prototype system runs only a simple OS, FreeRTOS, and *not* a rich OS such as Android. The hypervisor shields critical hardware, and supports protected memory regions and fast switching between virtual guest modes via the ARMv5 MMU's page table and domain features. This architecture is sufficient to support multiple secure isolated services running alongside a FreeRTOS real-time scheduling kernel and its tasks.

Our test programs, designed to stress the hypervisor's key performance burdens, including guest mode switching, hypercalls, and interrupt handling, experienced approximately a mere 2 to 7% overhead in terms of instruction counts[3] on our simulated platform. Additionally, porting the platform-dependent portions of the FreeRTOS kernel necessitated only an acceptable level of paravirtualization effort, making the platform as a whole feasible. Finally, we anticipate that moving to ARMv6, whose MMU includes a non-executable bit, will enable also low-overhead kernel integrity protection.

We eventually intend to fully implement the proposed architecture, which will include additional features such as providing efficient hypervisor support for virtual address spaces in the primary guest OS, and allowing a broad range of interrupt support.

IV. COMPARISON WITH ARM TRUSTZONE

What do we gain by making a design as the one described in Section III compared to doing the very same with a TrustZone architecture? The answer depends on the particular security goals. However, TrustZone is only available on a limited set of the ARM family CPUs. Moreover, it requires that the SoC design is made according to the TrustZone principles from the very beginning in order to work, as sensitive units such as DMA, memory interfaces, and interrupt controllers must be designed to support TrustZone. Hence, if we in principle fulfill the same security requirements using only the MMU, standard peripheral units and standard processor protection mechanisms, why not use such a design instead as long as the performance impacts are reasonable? Furthermore, TrustZone does not allow the Secure world to interpose on and monitor all the important events in the Normal world, as a hypervisor can. Table I below summarizes the main differences between

[3] Instruction counts are our best metric for now, as actual timing benchmarks will not be meaningful on our OVP simulation platform.

the two approaches with respect to the expected security characteristics we identified in Section III.

Almost as important as the pure security aspects are the software porting impacts. TrustZone leaves the legacy system *in principle untouched* (only trusted channel adaptations) while the hypervisor-based approach typically requires substantially larger porting (or paravirtualization) efforts due to the requirement of running the hypervisor in privileged mode, and virtualizing the MMU and security-critical peripherals.

TABLE I
COMPARISON BETWEEN TRUSTZONE AND HYPERVISOR ARCHITECTURE ON ARM

Security aspect	TrustZone	Hypervisor
Trusted execution	Single trusted execution domain. Multiple domains can be implemented *inside* the "secure world" through a secure OS.	Multiple trusted execution environments provided through the hypervisor layers, i.e., not specific requirements on the OS or OSes in trusted domains.
TCB	Monitor, CPU, Interrupt controller, on chip RAM, Boot ROM, security critical peripherals (DMA, Watchdog etc.)	Hypervisor, CPU, MMU, on chip RAM, Boot ROM
Secure interface	ARM secure channel API	Hypercall or shared memory interface
Non-trusted domain protection	Secure monitoring of the non-trusted domain is not possible	Kernel integrity or more advanced secure services provided by the hypervisor

V. CONCLUSION

As noted in many previous works, isolation through virtualization is very attractive from a security point of view. Due to the heterogeneity in mobile platform architectures, we need carefully adapted designs that suit the needs for the particular target system. Such designs, *combined* with thorough analysis as well as formal verification of key security properties of the design, have good potential for securing future systems. In particular, as we have shown, a hypervisor can provide secure on-chip execution with very limited performance and reasonable software porting costs.

Compared to the alternative architecture built around ARM TrustZone, the major advantages of a hypervisor design lie in a smaller hardware TCB, support of multiple secure execution domains and the possibilities to run on many embedded architectures *without* any need for *hardware changes/adaptations*. Furthermore, hypervisors enable secure interposition and monitoring of non-trusted domains which is *not* possible in a TrustZone architecture. This flexibility comes at the price of larger legacy system porting efforts, slightly lesser performance and larger software TCB.

In order for hypervisor-based security to really succeed, there is a need for industry standardization regarding how to write and interface to secure services in a trusted domain. Without such in place, there is a risk that a large number of different solutions will appear on the market, preventing broad adoption of the technology.

REFERENCES

[1] K. Stammberger, "Current trends in cyber attacks on mobile and embedded systems," *Embedded Computing Design*, September 2009. Available:http://embedded-computing.com/current-trends-cyber-attacks-mobile-embedded-systems

[2] ARM TrustZone Technology. Available: http://www.arm.com/products/processors/technologies/trustzone.php

[3] J. E. Smith, and R. Nair, *Virtual machines: versatile platforms for systems and processes*. Morgan Kaufmann Publishers, USA 2005.

[4] CP 67. Available: http://en.wikipedia.org/wiki/IBM_System/360_Model_67

[5] VMware. Available: http://www.vmware.com/

[6] G. Heiser, "The Role of Virtualization in Embedded Systems", *First Workshop on Isolation and Integration in Embedded systems (IIES ´08)*, April 2008.

[7] OKL4 Microvisor. Available: http://www.ok-labs.com/products/okl4-microvisor

[8] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes", *Proceedings of the 21st Symposium on Operating System Principles(SOSP 2007)*, October 2007.

[9] M. Xu, X. Jiang, R. Sandhu, and X. Zhang, "Towards a VMM-based Usage Control Framework for OS Kernel Integrity Protection", *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT 2007)*, June 2007.

[10] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. Ports, "Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems", *Proceedings of the 13th Annual International ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.

[11] J. Yang and K. G. Shin, "Using Hypervisor to Provide Data Secrecy for User Applications on a Per-Page Basis", *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '08)*, pp. 71–80, March 2008.

[12] L. Litty, H. A. Lagar-Cavilla, and D. Lie, "Hypervisor Support for Identifying Covertly Executing Binaries", *Proceedings of the 17th USENIX Security Symposium*, pp. *243–258*, July 2008.

[13] J. Brakensiek, A. Dröge, M. Botteck, H. Härtig, and A. Lackorzynski, "Virtualization as an Enabler for Security in Mobile Devices", First Workshop on Isolation and Integration in Embedded Systems (IIES '08) , April 2008.

[14] F. Armand and M. Gien, "A Practical Look at Micro-Kernels and Virtual Machine Monitors", *Proceedings of the 6th Consumer Communications and Networking Conference (IEEE CCNC '09)*, January 2009.

[15] VirtualLogix Available: http://www.virtuallogix.com/

[16] G. Heiser, "Benchmarks: How not to do them", accessed May 2010. Available: http://www.ok-labs.com/blog/entry/benchmarks-how-not-to-do-them/

[17] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal Verification of an OS Kernel", *Proceedings of the 22nd ACM Symposium on OS Principles (SOSP '09)* , October 2009.

[18] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization", *Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP)*, vol. 37, 5, ACM Press, pp. 164–177, October 2003.

[19] J-Y. Hwang, S-B. Suh, S-K. Heo, C-J. Park, J-M. Ryu, S-Y. Park, and C-R. Kim, "Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones", *5th IEEE Consumer Communications and Networking Conference (CCNC 2008)*, January 2008.

[20] T. Ormandy, ”An empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments”, CanSecWest, April 2007. Available: http://taviso.decsystem.org/virtsec.pdf
[21] C. Gehrmann and P. Ståhl, “Mobile Platform Security”, *Ericsson Review*, No. 02, 2006. Available: http://www.ericsson.com/ericsson/corpinfo/publications/review/2006_02/files/mobile_platform_security.pdf
[22] Open Virtual Platforms. Available: http://www-ovpworld.org/
[23] ST-Ericsson, U8500 smart phone platform. Available: http://www.stericsson.com/platforms/U8500.jsp