DE GRUYTER OPEN

Embedded Systems Development Tools: A MODUS-oriented Market Overview

Michalis Loupis
Central Greece Institute of Technology, Department of Electrical Engineering

Abstract

Background: The embedded systems technology has perhaps been the most dominating technology in high-tech industries, in the past decade. The industry has correctly identified the potential of this technology and has put its efforts into exploring its full potential. **Objectives:** The goal of the paper is to explore the versatility of the application in the embedded system development based on one FP7-SME project. **Methods/Approach:** Embedded applications normally demand high resilience and quality, as well as conformity to quality standards and rigid performance. As a result embedded system developers have adopted software methods that yield high quality. The qualitative approach to examining embedded systems development tools has been applied in this work. **Results:** This paper presents a MODUS-oriented market analysis in the domains of Formal Verification tools, HW/SW co-simulation tools, Software Performance Optimization tools and Code Generation tools. **Conclusions:** The versatility of applications this technology serves is amazing. With all this performance potential, the technology has carried with itself a large number of issues which the industry essentially needs to resolve to be able to harness the full potential contained. The MODUS project toolset addressed four discrete domains of the ESD Software Market, in which corresponding open tools were developed.

Keywords: embedded systems, formal verification, co-simulation, performance optimization, code generation

JEL main category: Economic Development, Technological Change, and Growth
JEL classification: O31
Paper type: Research article

Received: 12, June, 2013
Accepted: 10, February, 2014

Citation: Loupis, M. (2014), "Embedded Systems Development Tools: A MODUS-oriented Market Overview", Business Systems Research, Vol. 5, No. 1, pp.6-20.
DOI: 10.2478/bsrj-2014-0001

Acknowledgments: This research activity is funded under the EU Research for SME associations FP7 project, MODUS-Methodology and supporting toolset advancing embedded systems quality (Project No.286583).

Introduction

The embedded systems technology has been perhaps the most dominating technology in the industry, in the past decade. The industry has correctly identified the potential of this technology and has directed its efforts in exploring its full potential. The versatility of the application this technology serves is amazing. With all this performance potential, the technology has carried with itself a large number of issues which the industry essentially needs to resolve to be able to harness the full potential contained. Embedded applications normally demand high resilience and quality, as well as conformity to quality standards and rigid performance. As a result embedded system developers have adopted software methods that yield high quality. The MODUS project toolset addressed 4 discrete domains of the ESD Software Market, in which corresponding open tools were developed. This paper presents a MODUS-oriented market analysis in the domains of Formal Verification tools, HW/SW co-simulation tools, Software Performance Optimization tools and Code Generation tools.

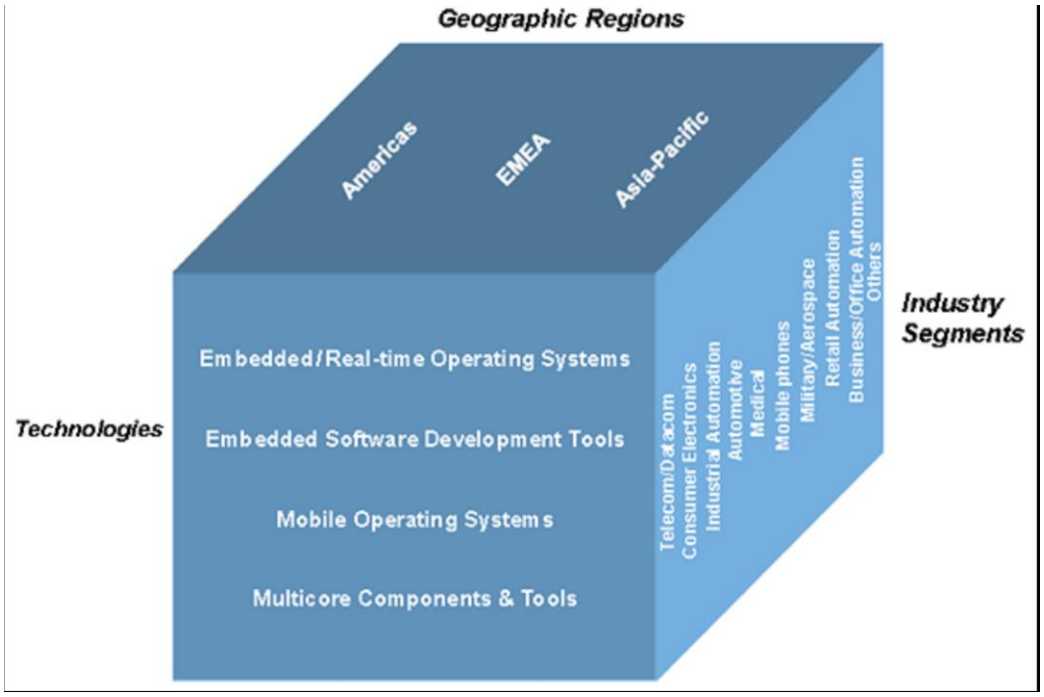
Market overview and size

Embedded systems

Embedded system technology has positively affected a wide range of application sectors including consumer electronics, medical electronics, automotive, and aerospace. Serving various aspects of the needs of these application sectors has predominantly been the challenge for developments therein. The overall market of embedded real-time operating systems can be classified as in

Figure Figure 1, which shows the technologies involved, the main industrial segments and the various geographic regions.

Figure 1
Market Definition and Segmentation for Embedded Systems



Source: IDC (2012)

The embedded systems market can be divided into sectors that concern top safety critical applications (i.e. automotive/rail, medical, aerospace) and applications of less stringent safety requirements (i.e. consumer electronics, mobile phones, industrial automation, telecom).

However, a failure of any system that could entail a very high financial loss is not to be ignored and therefore a vendor that can provide reliable applications of high availability and integrity can potentially capture a significant portion of the embedded systems market, regardless of market specific area. The emergence of new processors has also raised new potentials for these embedded applications. They can now incorporate more functionality, a feature however that inherently increases system complexity. In turn this justifies the need to allow multiple applications of different criticality to run on a single processor and share a common memory. Such a software paradigm would require on the other hand independent, protected execution time and memory space for each system application. Partitioned software architectures are the key component to share applications on the same hardware while increasing the security and robustness of the system (MODUS Project, 2013).

Embedded systems market value was almost €852 billion in 2010. The overall industry has been growing at a compound annual growth rate (CAGR) of 12% throughout the period and should reach a revenue size of €1,5 trillion by 2015. This is growing not only at a faster pace than any traditional IT sector but also more than 150% faster than the total semiconductor industry. Energy is still the fastest growing market (2010-2015 CAGR: 34%) while communications, automotive and healthcare market will sustain an annual double digit growth along the same period. The worldwide market for embedded systems development software (ESD) had a size of about €3.31 billion in 2010. IDC forecasts that the worldwide market for ESD software will expand at a 7.1% compound annual growth rate to surpass €6,5 billion in 2020 (IDC, 2012). European ESD software revenues in 2010 reached about €986 billion, or 30% of worldwide revenues, compared with 46% for North America and 19% for the Asia-Pacific region including Japan. Notably Europe's and North America's 2010-2020 CAGR are slightly below the worldwide total ESD Software revenue growth during the same period; the European one is estimated at 6.3% and the North American one at 6.8%.

This growth in most cases will be attributed to advanced, cloud-aware embedded systems of increased complexity, which will entail faster hardware, reliable connectivity and more sophisticated operating systems and analytical software. It is expected that in 2015, more than 4 billion units will be sold, to create a compound market of $2 trillion. In the same period the needs of embedded systems for microprocessor cores will exceed 14.5 billion units. Moreover embedded systems developers need to adapt to a challenging economic environment, pressing time-to-market limitations and cost reduction requirements, while simultaneously dealing with the actual technical impediments and particularities of their work. Often these impediments will directly conflict with the increasing complexity now linked with many new embedded systems. A further complication involves the growing mobility needs and intensifying requirements for inherently safe and secure, sometimes even critical, in a pervasive digital world (MODUS Project, 2013).

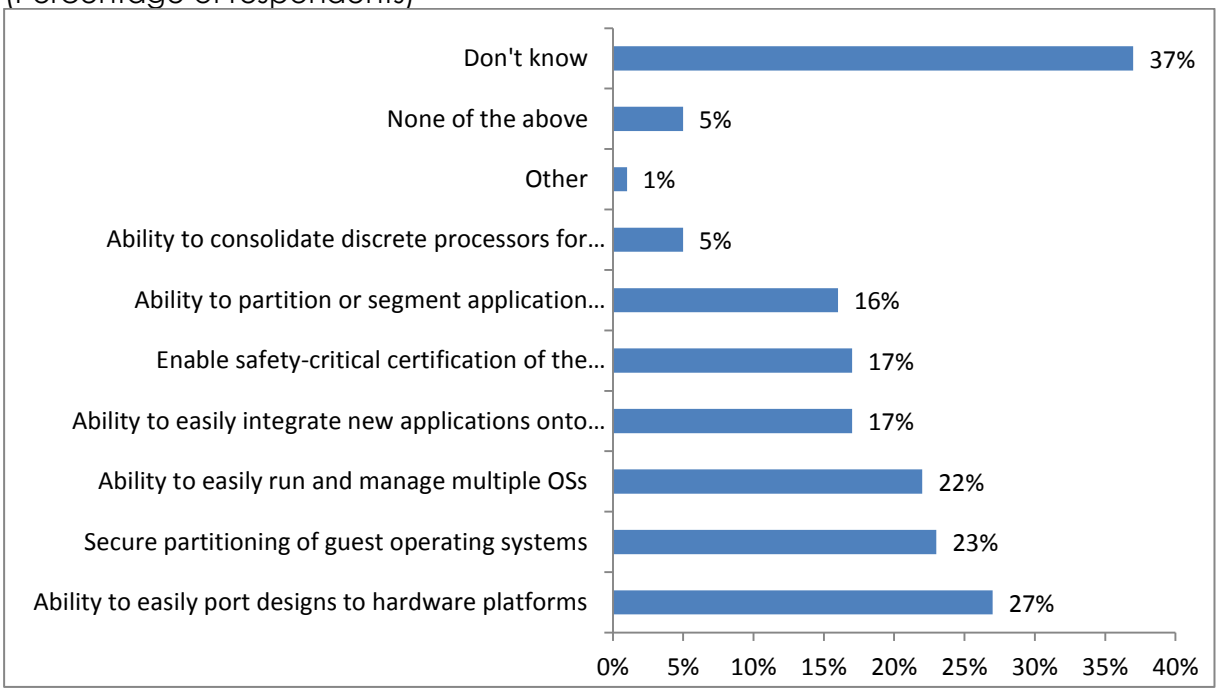
Furthermore, the lack of effective management tools for multiple cores and enhanced performance creates a barrier towards the adoption of multicore architectures. Virtualization solutions for mobile and embedded applications have emerged in recent years, as an approach addressing many of these challenges. It should be noted however that any advantages of virtualization for enterprise systems, which could range from potential overhead savings through server

consolidation to improved flexibility and data storage capacity, will not be identical to the ones encountered in mobile and embedded systems (Kebemou, Schieferdecker, 2007). Many of these differences, of course, are attributed to the specifications inherent in many embedded designs, mainly pertaining to power and memory limitations, and the normally small form nature of embedded devices. The top benefits, as shown in Figure 2, include design portability to new hardware platforms, secure partitioning of guest operating systems, and the ability to easily run and manage multiple operating systems.

The embedded systems market can be divided into sectors that concern top safety critical applications (i.e. automotive/rail, medical, aerospace) and applications of less stringent safety requirements (i.e. consumer electronics, mobile phones, industrial automation, telecom).

However, a failure of any system that could entail a very high financial loss is not to be ignored and therefore a vendor that can provide reliable applications of high availability and integrity can potentially capture a significant portion of the embedded systems market, regardless of market specific area. The emergence of new processors has also raised new potentials for these embedded applications. They can now incorporate more functionality, a feature however that inherently increases system complexity. In turn this justifies the need to allow multiple applications of different criticality to run on a single processor and share a common memory. Such a software paradigm would require on the other hand independent, protected execution time and memory space for each system application. Partitioned software architectures are the key component to share applications on the same hardware while increasing the security and robustness of the system (MODUS Project, 2013).

Figure 2
Primary Advantages from the Use of Virtualization in Mobile and Embedded Systems (Percentage of respondents)



Source: IDC (2012)

Embedded Systems SW Engineering

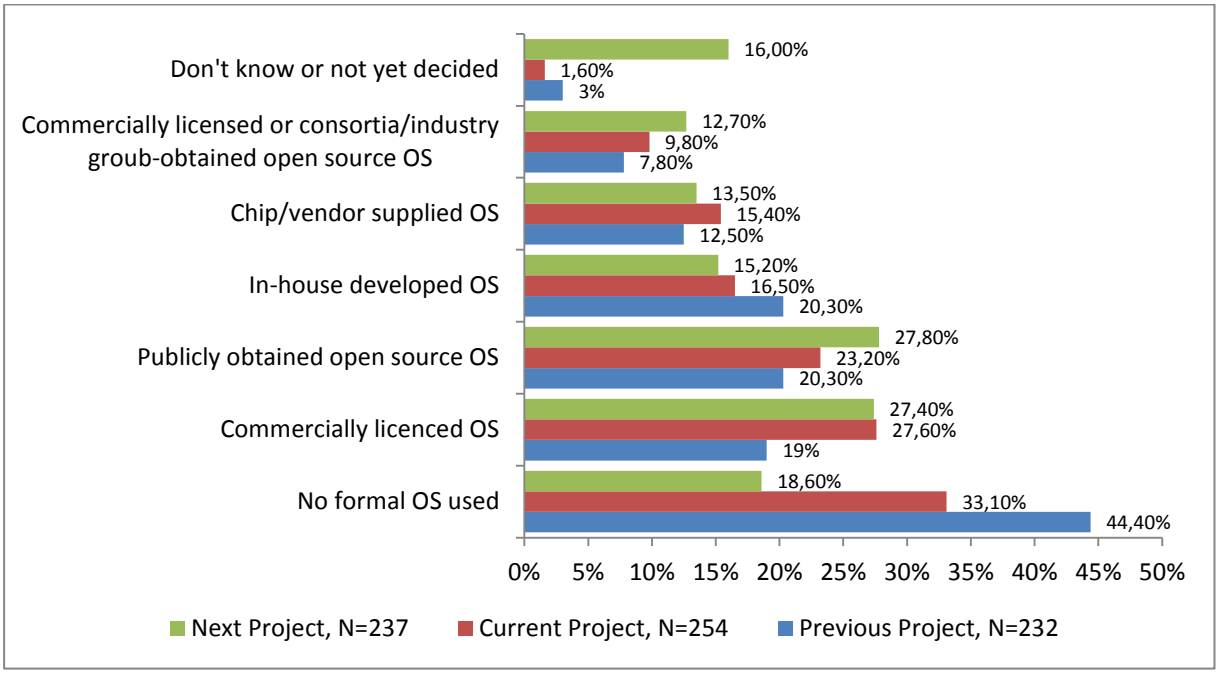
Embedded applications normally demand high resilience and quality, as well as conformity to quality standards and rigid performance. As a result embedded system developers have adopted software methods that yield high quality. Examples of the practices used in embedded software development include the following:

- Requirements prioritization and traceability of quality through Quality Function Deployment (QFD).
- Model-driven design and test.
- Mathematical modelling for reliability, power consumption, thermal, and performance analysis
- Formal design and code verification.
- Automated static code analysis for memory, performance, and security.
- Extended automatic testing.
- Design of explicitly reusable code.

In the embedded world development methods are usually more formal in its compared to other software domains. The main reason is that many embedded applications are intrinsically safety-critical (for example, medical devices, industrial automation, automotive, or aerospace). Hence developers have been led to adopt and use formal methods that concentrate on quality on a systematic base to face the safety and criticality constraints. Along these lines, several industries are forced to comply with stringent quality and schedule requirements, resulting in extensive fault criteria. For example, in the aerospace business, deadlines and quality are unconditional requirements, and the same stands as for the automotive electronics or industrial automation, where extensive external systems are depending on the timely availability of robust embedded software controllers. Nowadays, SMEs have many choices as to which target operating system to integrate in their embedded systems depending on their needs. The key parameter is that device/system functionality is driving complexity and increased software content, which in turn leads to more sophisticated system interfaces, complex graphical elements, wired and wireless capabilities, and others. Consequently, there is a notable shift regarding the operating systems used in target devices, from not formal and in-house developed, to a variety of other commercial and open source products (MODUS, 2013). In a recent survey it was recorded that the use of commercially licensed (not open source) operating systems is expected to remain fairly stable, whereas almost 50% of the participating engineers stated that their target OS is selected on a project-by-project basis.

Real-time applications should be deterministic, exhibiting timeliness and predictability, and the operating systems addressing these applications meet these constraints by paying special attention to a number of OS features such as multitasking, task synchronization, deterministic handling of interrupts and events, i/o management, inter-task communication, provision of timers and clocks and memory management. Various RTOSs (Real-Time OS) implementing those functional requirements, differ in their implementation choices and strategies. Leading solutions for the embedded domain include Real-Time Systems' RTS Hypervisor, Green Hills Software's INTEGRITY Multivisor, LynuxWorks' LynxSecure, SYSGO's PikeOS, TenAsys' eVM for Windows, and Wind River Hypervisor. For mobile embedded applications, Open Kernel Labs' OKL4 Microvisor and Red Bend Software's VLX are two of the most widely used solutions, while VMware – a leader in enterprise/IT virtualization– is expected to expand in the mobile domain (MODUS Project, 2013)

Figure 3
Survey Results on the Type of Operating System Used (% of respondents)



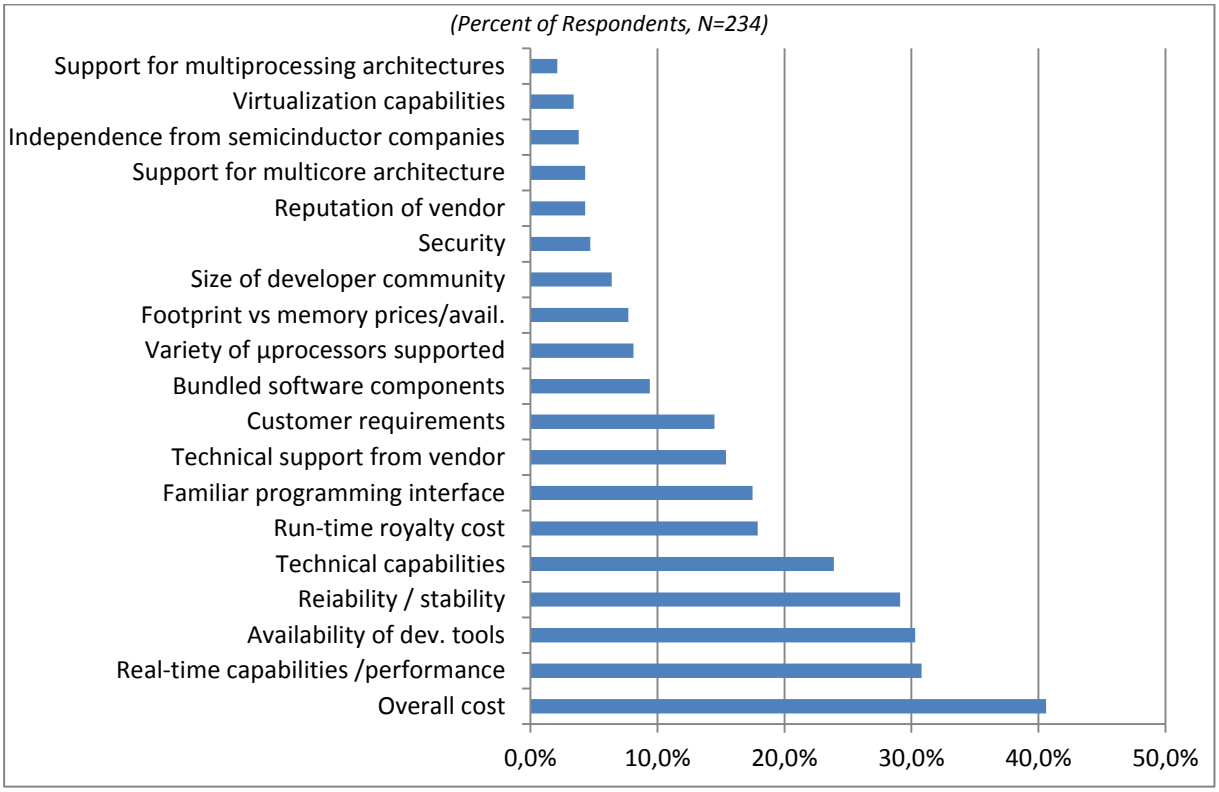
Source: IDC (2012)

The commercially available operating systems can be seen as a closely coupled system of functionality, performance, and price. They range from those offering a basic pre-emptive scheduler and a limited number of system services, which are usually inexpensive or royalty free and include modifiable source code, to those more sophisticated operating systems that typically include a lot of functionality beyond the basic scheduler and can be quite expensive. With the given variety of operating systems and corresponding features, it is usually difficult to decide which OS is the best for a given application. Many developers base their decision on performance, set of functionalities, or compatibility with their legacy or chosen compiler, debugger, and other development tools, as shown in Figure 4.

Moreover, although the actual use of multi-core and operating system virtualization technologies is currently limited to a fraction of projects under development (see Figure 1), an increasing number of embedded system developers is planning to incorporate these technologies in future projects. The validity of these plans is sustained by supplier messaging and the availability of training seminars and product support for both operating systems and tools solutions to support both multi-core and virtualization environments. Operating system virtualization specifically offers advanced capabilities to cope with such issues as obsolete silicon parts, evolving legacy software assets, for which significant investments have been made already, and providing an environment where multiple guest operating systems and applications can operate independently.

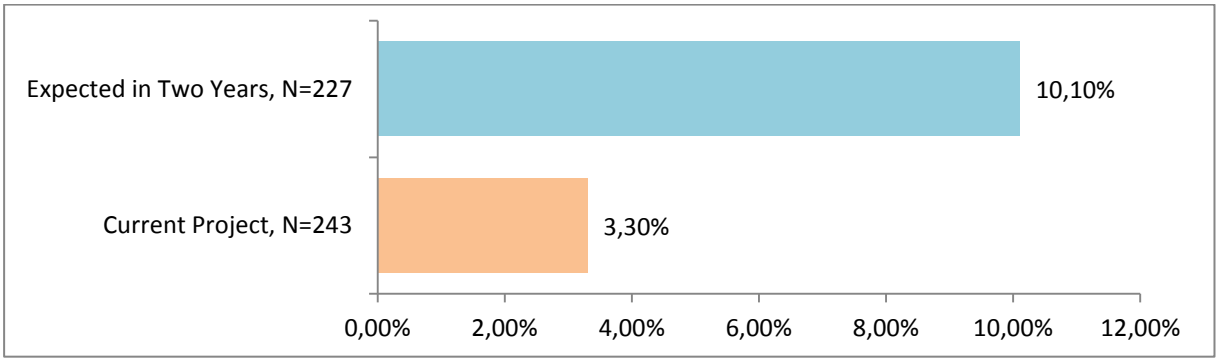
For enterprise IT applications, virtualization has emerged as a key strategy to control costs by consolidating servers, therefore reducing the related hardware, floor and in-rack space, power consumption, and cooling. In the same time it offers an increase of availability, reliability, redundancy and performance, leading to a new approach to such computing infrastructures like cloud computing, grid and clusters that can be easy implemented and managed.

Figure 4
The Most Important Characteristics for Selecting Embedded Operating System



Source: IDC (2012)

Figure 1
Hypervisor/Virtualization Layer/Microkernel Use in Current and Future Projects (Percentage of respondents)



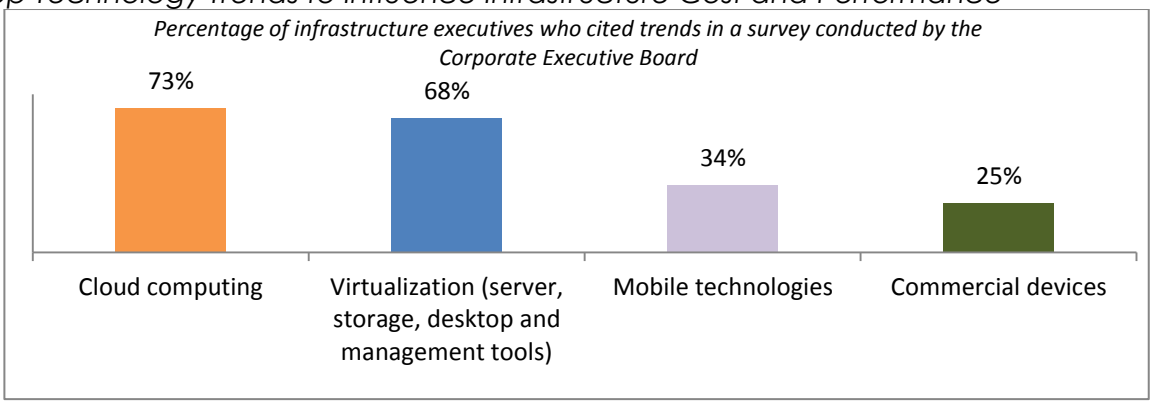
Source: IDC (2012)

In recent years, virtualization has evolved from an experimental technology used only in test and development environments to a core infrastructure technology. Now, many businesses plan for all new servers to be virtualized and the use of virtual servers has overcome the implementation of physical servers (Figure 2). Although there are several parameters to be considered, server consolidation is the primary reason that is driving the wholesale adoption of virtualization. Server consolidation allows organizations to increase the rate of hardware utilization while decreasing the power costs and management requirements at the same time. In addition, high-availability technologies such as vMotion and Live Migration have also disassociated virtual machines (VMs) from their physical hosts, creating the foundation for the

dynamic data centre where VMs can be moved between hosts automatically in response to changing workloads. Virtualization is a core mainstream technology that will definitely alter the IT landscape for the foreseeable future. IDC studies have shown that one out of every five servers is virtualized today but it seems clear that those numbers will be reversed in just a few years, and virtual servers will far outnumber physical ones (IDC, 2012).

As virtualization separates the server from the underlying hardware, virtualization technology is applied more on infrastructure such as cloud computing and mobile computing platforms. Cloud computing is indeed an emerging trend that tends to rely on virtualization, such as Amazon's EC2 and Windows Azure's Hyper-V, and many related services are built on virtual servers. The cloud disassociates the service or application from the underlying infrastructure and allows the management of multiple servers and applications as part of an extended service. The cloud's viability has now enabled emerging businesses and services to provide businesses with compelling and ready-to-use solutions (e.g., Windows Intune, Microsoft Office 365).

Figure 2
Top Technology Trends to Influence Infrastructure Cost and Performance



Source: IDC (2012)

The current leverage may be limited, but the adoption of cloud technologies will certainly grow, migrating parts of the IT infrastructure to off-premises hosting companies. Furthermore, mobile computing platforms and smart phones such as the iPhone, Samsung, Android and BlackBerry, are now beyond the point of phone production alone. The torrential growth of the mobile apps market and Internet connectivity has made smart phones a useful productivity device that could make use of the virtualization technology. Virtualization of mobile devices would allow carrying one device with multiple virtualized environments which could support different levels of confidentiality and applicability.

Market Analysis of Relative Tools

Model Verification Tools

Formal verification in general is a process of verifying the correctness of a system with respect to the formal system specification. With the help of mathematical methods and tools the correctness of the system is proved or disproved. Formal verification can be used for almost any system, but due to additional effort involved it usually is used for critical systems. These systems may include protocols, digital circuits, and software. When system needs to be formally verified, the first step of the process is the formal description of the requirements. This alone is beneficial because it is often a case that the specification is ambiguous, incomplete, and inconsistent. When it is

required to write down the specification using mathematical formulas it is easy to spot ambiguous parts of informal description. Model checking is the automated process or technique for verification of properties of the finite state machine. The idea is to ensure that the system does not reach deadlock and other forbidden states. The algorithm that is used for model checking procedure requires the specification and the model of the system to be described in precise mathematical language (Schmidt, 2006).

Formal specification languages are the languages used during system design, and analysis, describing in a formal way the system requirement. Typically specification languages are not intended for creating executable code. They are tailored for description of certain system properties often using mathematical syntax. There are many formal specification languages: VDM (++), Z, B, RSL, CASL, Petri Nets, CSP, Temporal, Logic, OCL, and JML. The specification languages differ as some of them were intended to deal with different types of services. Many of the languages were inspiration for creation of automated verification and validation methods and tools. These tools for model checking available within different frameworks are summarised in Table 1(MODUS project, 2013).

Table 1
Summary Table for Formal Verification tools

Features		RAISE	MATLAB Simulink	SPIN	UPPAAL	NuSMV	UML
Language		RSL	Graphical + Matlab syntax	Promela	Graphical	NuSMV	Graphical
Functional features		Formal specification (RSL), verification (SAL)	Specification – Stateflow, Verification Simulink	Formal verification	Formal specification, Verification	Formal specification, Verification	Formalisation of the specification
Integration in the process flow	**Previous step**	Informal specification	Informal specification	Informal specification	Informal specification	Informal specification	Informal specification
	Next Step	Verification (using SAL)	Final system design, code cogeneration using Matlab tools	Final system design, code cogeneration	Final system design, code cogeneration 3rd parties	System implementation	System verification
Interface		Text, Emacs	Graphical	Graphical or Emacs (text)	Graphical	Text	Graphical
Extensions		Translator to SML, SAL and UML	Simulink Coder or Embedded Coder, integration with xPC Target board	Eclipse plug-in, UML to SPIN (Hugo/RT)	UML translator, C translation	UML translator Eclipse extension	Extensions to verification and code generation tools
Environment		Solaris, Linux, Windows	Windows, Linux, and Mac OS	Windows, Linux, and Mac	Requires Java	Linux, Mac OS X, Microsoft Windows	Windows, Linux, and Mac
License		Open Source	Requires license	Open-source software	Requires license	LGPL	Free and commercial tools available

Source: Author

Co-Simulation Tools

The aim of this section is to make an analysis and evaluation of co-design tools available in the market (Table 2). In this extent, several criterions have been taken into account: Technical approach of the tool, Ability to be integrated into MODUS design flow, Compliance to industry standards, Extensibility, Support, Price, Licensing models, and Long-term availability. Within MODUS we have identified 5 tools, considered as representatives of the current industrial practises: CarbonStudio, Cadence Virtual Prototyping, SpaceStudio, OVP, and Mentor Graphics ModelSim.

Further analysis of the respective tools is available in (MODUS Project, 2012), where an overview of the features and capabilities of the relevant available tools in the market are presented. Among these tools we have eliminated CarbonStudio and Cadence Virtual Prototyping as cost of a license is too high (starting from 100k€).

Table 2
Summary table for HW/SW co-simulation tools

Features	Open Virtual Platform	Space Studio	ModelSim
Language	C/C++, SystemC, assembly	C/C++, SystemC	SystemC, VHDL, Verilog and SystemVerilog
Functional features	Instruction-accurate simulator LGPL'd peripherals models API for creating new components Integrates SystemC models	Design environment for embedded systems and SOCs Timed and untimed simulations In-depth non-intrusive hardware/software Integrates external SystemC models SystemC – VHDL translator	Native support of VHDL,Verilog, SystemVerilog and SystemC Full code coverage Post-simulation analysis Open TCL/tk or C API
Interface	Command line, graphical is using gdb front-ends	Graphical	Graphical
Environment	Windows, Linux	Windows	Windows
License	Requires license only for simulator part	Requires license	Requires license

Source: Author

Software Performance Optimization Tools

Software performance optimization is a critical component in achieving high performance for embedded systems. Performance optimization can be defined as the process of modifying specific software system components in order to work more efficiently or use fewer resources. Computational specialists have adopted programming strategies affecting the utilisation of hardware resources and have parameterized their software implementations to accommodate the architectural variety of modern computing platforms. While this approach has been quite successful, it is largely susceptible to errors and extremely time consuming for developers to manually program the management of hardware resources. This approach mandates a repeatedly modify-compile-execute development cycle until a sufficient performance gain is achieved (or the development deadline has been expired) (Falk et al., 2004). Towards overcoming the aforementioned problems, there has been considerable work in the domain of iterative optimisation in an effort to effectively automate this process. Those practices concentrate on choosing optimal program modifications or transformations in order to reduce the number of modify-compile-execute cycles (see Table 3).

Table 3: Summary table for Software Performance Optimization tools

Features	MENTOR EDGE System Profiler	IAR Embedded Workbench	RAPITA Verification Suite	QNX Momentics Tool Suite
Language	Embedded C, C++, assembly	C/C++	C/C++, Ada	C/C++
Functional features	Dynamic Memory Problem Analysis, System OS Analysis	Automatic checking of MISRA C rules, Power debugging	Eliminates unnecessary test activities, on-target coverage and timing measurement	Source debugger, target system information, application profiler, system profiler, memory analysis
Interface	Graphical	Graphical	Graphical	Graphical
Environment	Windows, Linux, Nucleus OS	Windows	Windows, Linux	Windows, Linux
License	Requires license	Requires license	Requires license	Requires license

Source: Author

Code Generation Tools

As already outlined, emerging technology gives rise to an increase in the complexity of applications. To deal with this code generators are a used practice to increase code quality and decrease development time. Their objective is to generate repetitive source code while maintaining a high consistency level of the generated program code. The act of code generation is based on an ontological model such as a template. Code generation tools will normally produce chunks of repetitive code, allowing programmers to concentrate on specific code. Thus generators increase productivity; generate large volumes of code, which would require a longer development time if coded manually (see Table 4). The need for consistent code quality is satisfied throughout the entire automated code generation by applying consistently coding conventions, unlike manual coding, where the quality is usually inconsistent. If errors are traced in the generated code, they can be corrected in short time through the revision of templates and rerunning the process of code generation (Vestal, 1994).

Table 4
Summary table for Code Generation tools

Features		Acceleo MTL	IBM Rational Software Architect	MagicDraw	Mia-Generation	SinelaboreRT	SunRPC
Language	**input**	metamodel compatible with EMF like UML	UML	UML, SysML	UML	UML	.x IDL
	output	C, Fortran, Java, , any Markup Language	Java, C#, C++ and other	J2EE, C#, C++, CORBA IDL and other	C++, Java, Delphi	C++, Java and C#	ANSI C
Functional features		Code generation from EMF based models	Model-to-code and code-to-model transformations	Code generation from model static structure	Code generation from model static structure	Code generation from model static structure	Automatically generates the client and server stubs for RPC calls
Interface		Graphical	Graphical	Graphical	Graphical	Graphical	Command line
Environment		Windows, Linux, Nucleus OS	Windows	Windows, Linux	Windows, Linux	Windows, Linux, Mac OS X	Unix, Linux, Windows
License		Open-source (EPL)	Requires license	Requires license	Requires license	Requires license	Sun (Free)

Source: Author

Virtualisation Tools

Embedded hypervisors differ from their conventional counterparts in that they implement a specific type of abstraction with different constraints than other platforms. Efficiency is the objective in all cases, but embedded hypervisors must deal with further constraints, beyond the conventional virtualization environments. Besides processor sharing, memory tends to be one of the main performance limitations in embedded applications. To that extent, embedded hypervisors need to be small and extremely efficient as to their use of memory. Normally smaller code sizes are easier to validate and verify. In fact, several embedded hypervisor vendors offer a formally verified hypervisor and guarantee their bug-free operation.

Furthermore a smaller hypervisor results in a more secure and reliable development platform, effectively because the hypervisor is typically the only portion of the system to run in a privileged mode, implementing what is known as the Trusted Computing Base (TCB) and constituting a secure platform. Embedded hypervisors are normally built to share a hardware platform with multiple guests and applications but also extend communication methods to allow them to interact. This communication means is both efficient and secure, permitting privileged and non-privileged applications to coexist. In addition to providing containment for security and reliability, the embedded hypervisor provides benefits in terms of license segregation.

Table 5
Virtualization Techniques Overview

Technique	Advantages	Disadvantages	Products
Operating System Level Virtualization (separation kernel)	Adequate performance.	No strong isolation between virtualized environments. Simultaneous execution of multiple OSs not supported. Only Linux distributions supported. Related OSs provides no real-time characteristics.	Linux VServer, Solaris Zones & Containers, FreeVPS, openVZ
Full Virtualization	No modification of the guest OS required.	Multiple levels of abstraction, lead to low performance. Related OSs provides no real-time characteristics.	VMWare Server, Virtual Box, Kernel-based Virtual Machine (KVM)
Paravirtualization (Hypervisor virtualization)	Good performance. Strong isolation of virtual environments. Safety critical applications can coexist with non-critical ones. Applications of different levels of security can securely coexist. (separation kernels) Small footprint kernel easier to validate. Ability to monitor the guest OSs.	Guest OSs has to be modified.	Xen VMWare ESX Server PikeOS RTS Hypervisor OKL4 Virtual Logix Wind-River VxWorks Integrity-178B LynxSecure QNX XtratuM

Source: Author

Embedded hypervisors also offer a communication mechanism which permits proprietary software and open source software to coexist in isolated environments. Because of embedded devices becoming more open, the need to mix proprietary software with third-party and open source software is a key issue. Finally, the embedded hypervisor must support real-time scheduling. In the case of handsets, the hypervisor can share the platform with core communication capabilities and third-party applications. Real-time scheduling allows the critical functions to coexist

with applications that operate on a best-effort basis.

Two main approaches to build a partitioned system through virtualization can be found; separation kernel, also known as operating system level virtualization and platform virtualization that can be divided as well in full virtualization and para-virtualization. Most of the several virtualization solutions fall under one of these virtualization mentioned techniques that are in common use today: operating system level virtualization, full virtualization and para-virtualization. Table 5 provides a summary of advantages and disadvantages of these virtualization techniques and the most representative products that use them.

Table 6
Virtualization Solutions Using the Hypervisor Technique

Product	OS support	Licensing scheme	Areas of use
Xen	FreeBSD, NetBSD, Linux, Solaris, Windows XP & 2003 Server (needs vers. 3.0 and an Intel VT-x (Vanderpool) or AMD-V (Pacifica)-capable CPU), Plan 9	GPL	Server/Desktop Consolidation, Development/Testing
VMWare ESX Server	Windows, Linux, Solaris, FreeBSD, Osx86 (as FreeBSD), Virtual appliances, Netware, OS/2, SCO, BeOS, Darwin, others: runs Arbitrary OS	Proprietary	Enterprise Server Consolidation, Business Continuity, Development/Testing
PikeOS	PikeOS, Linux, RTEMS, OSEK, ARINC 653 APEX, ITRON	Proprietary	Safety and security critical embedded systems.
RTS Hypervisor	Windows XP, XP- Embedded, Linux, VxWorks, Windows CE, Android (OS), OS-9, RTOS-32, QNX, proprietary Oss	Proprietary	x86 based devices for robotics, industrial automation, medical, telecom, testing and measurement, real-time applications.
OKL4	Linux	GPL / proprietary	Embedded, mobile telecommunications, soft-real time applications
Virtual Logix	Linux, Windows XP, C5, VxWorks, Nucleus, DSP/BIOS, proprietary Oss	Proprietary	Embedded, mobile telecommunications
Wind-River VxWorks	VxWorks, bare metal virtual board	Proprietary	Embedded, safety critical, secure (defence, aerospace etc).
Integrity-178B	integrity	Proprietary	Embedded, safety critical, secure (defence, aerospace etc).
LynxSecure	LynxOS, Linux, Windows	Proprietary	Embedded, safety critical, secure (defence, aerospace etc).
QNX	QNX	Proprietary	Embedded, safety critical, secure (defence, aerospace etc).
XtratuM	UPVLC/FENTISS	open source	Embedded, safety critical, secure (aerospace, etc).

Source: Author

The para-virtualization is currently the fastest virtualization technique because of the provided virtual machine is near the native machine. With para-virtualization, it is possible to provide a higher-level interface to the hardware customized to be efficiently used by the partition. Thus, the para-virtualization is the technique more suited to the requirements of embedded systems, namely faster, simpler, and smaller code. The customization (para-virtualization) of the guest operating system is also not

a problem because the source code is available. Additionally, this technique does not call for special processor functionality that may increase the cost of the end product. In the use of para-virtualization techniques, the most common way of implementing virtualization is the use of hypervisors. Nevertheless, it is possible to find systems using the called microkernel. It is not fully clear and no consensus exist about which is the better solution for real-time embedded multicore systems, as we are talking about a very recent implemented technology. It is possible nowadays to find several solutions for embedded systems using para-virtualization, some based on microkernel and some in hypervisors: Xen, LynxSecure, PikeOS. Nevertheless, one of the drawbacks of the µkernels is the overhead introduced by the different layers of software. A most representative set of products that can be found is summarized in Table 6 (MODUS Project, 2012).

Conclusions

MODUS provides a software solution that complements existing CASE tools, by allowing the effective interfacing with formal model verification engines and SystemC-based HW/SW co-simulation platforms, as well as the effective design performance-tuning and customisable source-code generation, respecting coding standards. As shown in this market overview, existing CASE tools present limitations in terms of supporting quality strategies as model verification, HW/SW co-simulation, performance optimisation and customisable source-code generation for embedded software development. On the other hand, there are indeed different stand-alone and specialised tools (e.g. model verification engines, HW/SW co-simulation platforms, etc.) that the industry has not adopted mainly due to the fact that these cannot interface to popular CASE tools (i.e. they do not support common formalisms).

The aforementioned problems are magnified when considering the needs and capabilities of SMEs active in the embedded systems sector, which have to cope with increasing market pressure and rapid changes in demand. Indeed, big embedded-system development companies may have the means to apply expensive SW quality approaches (TQM) and iterative quality/design/development cycles or even having the associated tools customised to their development practises and different customer requirements. Therefore, software quality practises are presenting a competitive challenge to small and medium sized organisations with limited budgets and resources as compared to large industrial players.

References

1. Falk, H. Marwedel P. (2004). Source Code Optimization Techniques for Data Flow Dominated Embedded Software, Dordrecht: Kluwer Academic Publishers.
2. IDC (2012), "Final Study Report: Design of Future Embedded Systems (SMART 2009/0063)", available at: http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/documents/idc-study-final-report.pdf (1 April 2012).
3. Kebemou, A., Schieferdecker I. (2007), „Evaluating modeling solutions on their ability to support the partitioning of automotive embedded systems", in Mieso K. D. et al. (Eds.), Proceedings of the 2007 conference on Emerging direction in embedded and ubiquitous computing (EUC'07), Springer-Verlag, Berlin, Heidelberg, pp. 674-685.
4. MODUS Project (2012), State-of-the-art review and identification of technological requirements', Deliverable D2.1, Internal documenation

5. MODUS Project (2013), Title: Interim Market Assessment and Business Plan, Deliverable D7.7, Internal documenation
6. Schmidt, D. C. (2006), "Model-Driven Engineering", IEEE Computer, Vol. 39, No. 2, pp. 25-31.
7. Vestal, S. (1994), "Assuring the Correctness of Automatically Generated Software", AIAA/IEEE Digital Avionics Systems Conference, Vol. 13, pp. 111–118.

About the author

Michael I. Loupis was born in Athens in 1962. He earned a Dipl.-Ing. in Electrical Engineering, an M.Sc. in Microprocessor Engineering, a Dr.-Ing. in Information Technology and an M.Sc. in Quality Assurance. He is currently an Assistant Professor with the Central Greece Institute of Technology, Department of Electrical Engineering, Greece. His current research interests include software and modelling tools for embedded systems and design tools for renewable energy systems and energy management. Professor Loupis is a Senior Member of IEEE, a Member of the Technical Chamber of Greece, the Greek Union of Electrical and Mechanical Engineers and the Greek Association of Computer Engineers. Author can be contacted at **mloupis@teiste.gr**