Real-Time Operating Systems for Embedded Computing

Yanbing Li[†], Miodrag Potkonjak[‡], and Wayne Wolf[†]
[†]Department of Electrical Engineering, Princeton University
[‡]Department of Computer Science, UCLA

Abstract

We survey the state-of-the-art in real-time operating systems (RTOSs) from the system synthesis point of view. RTOSs have a very long research history which provides important theoretical results and useful industrial implementations. Convergence of applications, technology, and market trends of embedded systems implies a strong need for new generation of RTOS. Therefore, new system synthesis problem areas, notably hardware/software co-design and synthesis for systems-on-silicon (SOS), are opening up new avenues for RTOS research and development. This paper starts with a survey of classical academic and industrial RTOS work and continues with a survey of recent results related to co-design and design systems-on-silicon. We conclude by outlining future directions for the SOS RTOS.

1 Introduction

Recently, a convergence of applications, technology, and market trends of embedded systems has resulted in drastic quantitative and qualitative change in synthesis of application-specific systems. The size of average embedded consumer electronics applications has been approximately doubling each year. At the same time the average size, in terms of number of gates, of the integrated circuits has been doubling every three years, while the clock period has been steadily decreasing at just slightly lower pace. For example, the state of the art general-purpose processor in 1971 (Intel 4004) had 2,300 transistors and had clock rate 0.1 MHz. Today the latest-generation processor, the Intel Pentium-II, has 7.5 million transistors and a clock rate of 300 MHz [8]. These trends imply that, in each new generation of technology, higher levels of hardware sharing are required, feasible, and economically desirable.

From the qualitative point of view, new application are characterized by high volume of data, cost sensitivity, and very short market windows. Many popular applications, such as wireless telephony, internet browsing, and video-conferencing, are intrinsically defined by a need for high volume data management. Recent analysis of electronic market clearly indicates crucial importance of considering customers spending limits and to act promptly to the changes in the market conditions[6].

As a result of these trends, there is a growing consensus that the only **design reuse** techniques can close the fast growing gap between the potentials of semiconductor production technologies and traditional design productivity[9]. Software is a major element of modern reuse strategies and **real-time operating systems** (**RTOSs**) are critical components in reusable embedded software systems.

Traditionally, application-specific systems have been implemented using the algorithm-to-architecture methodology. While this was a viable option for synthesis of low complexity application-specific systems, it does not address needs of more complex emerging designs. Figure 1 illustrates the new system-level synthesis flow. RTOS provides isolation and interface to application and algorithm developers and compilation tools from one side and to architecture and integrate circuits designs from the other. Our goal

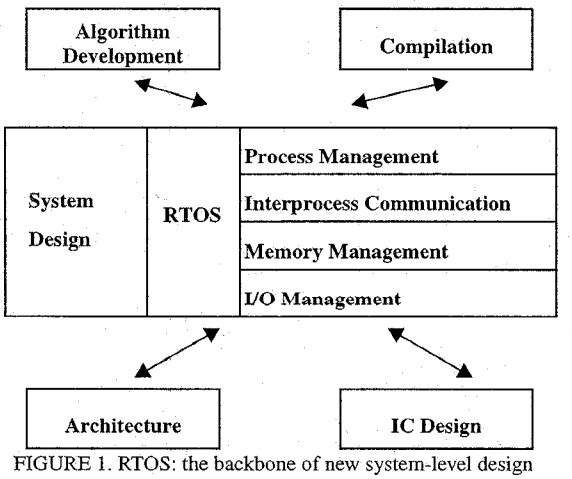

FIGURE 1. RTOS: the backbone of new system-level design process

in this paper is to analyze opportunities and challenges posed by the emerging field. RTOS have been used in many applications from car, ship, and airplane electronics to wireless and optical communication equipment, medical instrumentations, multimedia, internet, and even home appliances, factory automation, process control, financial transaction processing, and video games machines.

Even without strict space limitations of a conference paper, it would be impossible to provide a comprehensive survey of RTOS research and industrial state-of-the-art. Rather than aim for comprehensiveness, we focus attention on a selected subset of key issues and key new directions which characterize well emerging filed of RTOS for SOS.

The reminder of the paper is organized as follows. We first very briefly summarize operating system basics. After that we summarize the key results from the traditional RTOS research and discuss RTOS development efforts from the industrial point of view. Next, we present some of the first RTOS behavioral and system-level synthesis efforts. We conclude by outlining the directions for the merging field of system-on-silicon, in particular from RTOS point of view.

2 Preliminaries

In this section we briefly outline the basic facts about real-time operating systems and systems-on-silicon.

2.1 [Real Time] Operating Systems Basics

An **operating system** is a system program which provides an interface between application programs (an often a user) and the computer hardware. They have two primary functions: to make the computer system convenient to use and to organize efficient and correct use of the computer resources. There are four main tasks of an operating system: **process management**, **interprocess communication** and synchronization, **memory management**, and **input/output (I/O) management**. The process management component is responsible for process creation, process loading and execution control, the interaction of the process with signal events, process monitoring, CPU allocation and process termination. Interprocess communication covers issues such as synchronization and coordination, deadlock and livelock detection and handling, process protection, and data exchange mechanisms. Memory management includes services for file creation, deletion, reposition, and protection. I/O management handles request and release subroutines for a variety of peripherals and read, write, and reposition programs.

Real-time systems are systems where the proper functionality assumes both the correctness of the output as well as the correct timing behavior of the system.

2.2 Systems-on-Silicon

Integrated circuits with 20 millions or more transistors allow truly significant systems, such as HDTV, video camcoder, and software radio, to be placed on a single chip. Systems-on-silicon may contain multiple CPUs, special-purpose function units, and large amounts of embedded RAM. They find important uses in areas as diverse as wireless, multimedia, and mechanotronics.

Systems-on-silicon demand more careful design because design revisions are both costly and time-consuming. They also require very high-quality software for several reasons: the software will definitely be hard to observe and therefore debug due to pin limitations; the software may be hard to change after fabrication; and the hardware cannot be easily changed to accommodate software performance problems.

Design of systems-on-silicon pose many qualitatively new problems such as intellectual property protection, debugging using incomplete information about hardware and/or software, and one which we address here, a need for embedded operating system layer.

3 Traditional RTOS Research

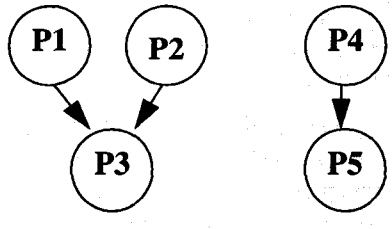

FIGURE 2. A pair of tasks.

A real-time operating system can supply many valuable functions to an embedded application, but the central purpose of an RTOS is scheduling of the CPU. The application is structured as a set of **processes**, each of which has its own program code and state consisting of register and memory values. The process is as well-known and almost universal model. Some operating systems support an additional level of structure known as the **task**. A task is a set of processes with data dependencies between them, as shown in Figure 2. This example contains two tasks—processes which have no path of data dependencies between them are in separate tasks. A process or task may be executed **periodically** or **aperiodically**.

Processes and tasks generally have some sort of temporal constraints on their behavior. The exact nature of these constraints depends on the scheduling model, but several types of temporal measurements of process/task performance are used. A **deadline** is the time at which a process must finish

execution after being initiated. The **period** of a periodic process or task is the interval between initiating successive executions.

The RTOS considers a process to be in one of three states: **waiting**, **ready**, and **executing**. In a uniprocessor system, only one process can be executing at any time. Some processes may be waiting for data or other events. Processes which are not blocked for external events but are not currently executing are considered ready. The transfer of execution from one process to another is called a **context switch**. To execute a context switch, the RTOS must save the state of the old process, determine what process will next obtain the CPU, and then set the CPU state to that process's state. Context switch overhead is non-trivial but often not a major factor in performance; scheduling policies, process partitioning, memory performance, and other factors are often more critical to obtaining good performance.

Stankovic et al. provide a good survey of real-time scheduling techniques[4]. There are several types of scheduling policies for real-time systems:

- A **cooperative** scheduler relies on the current process to give up the CPU before it can start the execution of another process. Cooperative multitasking is suitable for extremely simple systems, such as digital filters with purely periodic inputs, but is not used in sophisticated embedded systems.
- A **static priority-driven** scheduler can **preempt** the current process to start a new process. The highest-priority ready process is always the currently executing process. Priorities are set before the system begins execution.
- A **dynamic priority-driven** scheduler can redefine the process priorities at run time. The highest-priority ready process is still the currently-executing process, but since the RTOS can redefine priorities, the scheduling policy is embodied in the dynamic choice of priorities.

The best-known static priority-driven scheduling methodology is **rate-monotonic scheduling (RMS)**[3]. Liu and Layland showed that the optimal scheduling policy for one particular category of real-time systems is to assign priorities in inverse order of deadline—the shortest-deadline process receives the highest priority.

A well-known dynamic priority-driven scheme is **earliest-deadline-first (EDF)**. This policy assigns priorities based on the time left until each process reaches its deadline, with the process facing the nearest deadline receiving the highest priority. EDF results in higher CPU utilization than is possible with RMS, but since priorities in an EDF system change during execution, it is more difficult to prove that an EDF system meets all its deadlines under all conditions.

3.1 Recent Directions

Recently, there has been a number of new RTOS- and embedded system-related research results which open new rand important development directions. We mention only a subset of them: profile-based load balancing[14], lottery scheduling[18], dynamic code generation[19], embedding of security and authentication mechanisms[17], jitter hiding[20], verification[15], and CORBA-related efforts[16].

Bestvarsos[14] demonstrated significant advantages of using the profile-based balancing over traditional balancing techniques for optimization of distributed soft real-time systems. Set of promising scheduling technique which address a number of important issues has been also developed by utilizing randomization[18]. There has been also efforts to ensure satisfaction of real-time constraints in object oriented CORBA, an important first step in development of real-time large software systems.

Inferno, the real-time version of the Plan 9 operating system from AT&T Bell Labs, is the first RTOS which provides a number of cryptographic, authentication, and protection mechanism[17]. The first techniques for jitter hiding show significant advantages in handling relative small variation in throughput rate[20]. Finally, a number of recent studies shows that a great advantages in throughput, latency, and memory requirements can be achieved by applying the dynamic code generation techniques[19].

4 Commercial RTOS Offerings

Leading estimates that a few tens of millions copies of RTOS have been installed. Some of large RTOS industrial players include Integrated Systems (pSOS Systems), Wind River Systems (VxWorks), Microtec Research (VRTX), Microware Systems (OS-9), Spectron Microsystems (SPOX), QNX (QNX software), Hewlett-Packard (HP-RT), Lynx Real-Time Systems (LynxOS). Among the most notable recent entries include Lucent (Inferno) and Eonic Systems (Virtouso RTOS). Recently Microsoft entered the market with their Windows-CE embedded operating systems. While Windows-CE does not directly address requirements for real-time constraints, at least one company already announce its real-time version.

While traditional academic RTOS research is dominated by scheduling efforts, the industrial efforts have been mainly focused on optimization for three design metrics: memory requirements and two performance metrics: context switch time and interrupt latency. Table 1 shows area requirements and delay for 3 different cache configuration and five differ-

ent cache sizes obtained using the Flynn's cache modeling

Organization	Property	1KB	2KB	8KB	96KB
Direct	Area (mm^2)	2.107036	3.807370	13.962853	161.704666
Mapped	Latency (μ)	3.79491	3.97239	4.50642	6.78866
2-way	Area (mm^2)	2.385426	4.021449	13.799528	156.061787
	Latency (μ)	5.67454	5.74781	6.22891	8.53658
4-way	Area (mm^2)	3.042479	4.645923	14.245449	154.000343
	Latency (μ)	6.04178	6.23715	6.57738	8.83248

TABLE 1. Cache Trade-offs

tools[11]. Table 2 show features of some of the most popular microprocessor cores. Majority of RTOS have versions which start from a few KB and go up to several MB.

Microprocessor core	Clock (MHz)	MIPS	Technology (μm)	Area (mm^2)	Power diss. (mW) (Volt.)
StrongARM	233	266	0.35	4.3	300 (1.65)
ARM, 7	40	36	0.6	5.9	200 (5)
ARM. 7 Low-Power	27	24	0.6	3.8	45 (3.3)
LSI Logic, TR4101	81	30	0.35	2	81 (3.3)
LSI Logic, CW4001	60	53	0.5	3.5	120 (3.3)
LSI Logic, CW4011	80	120	0.5	7	280 (3.3)
DSP Group, Oak	80	80	0.6	8.4	190 (5)
NEC, R4100	40	40	0.35	5.4	120 (3.3)
Toshiba, R3900	50	50	0.6	15	400 (3.3)
Motorola, 68000	33	16	0.5	4.4	35 (3.3)
PowerPC, 403	33	41	0.5	7.5	40 (3.3)

TABLE 2. State of the art processor cores

Although, the state-of-the-art SRAM design reports state by factor of 2 smaller area per KB of fast memory[10,12,9] it is clear that in cost sensitive applications large RTOS imposes high cost penalties.

The performance criteria, context switch time and interrupt latency, are highly platform dependent. For 60 MHz computers both metrics are usually a few microseconds. The notable exception, with respect to all three metrics is the Virtuoso RTOS which claims at least an order of magnitude reduction for all of them.

The state-of-the-art RTOS usually provides about 16 or more scheduling priority levels, preemption mechanisms, several tasks scheduling algorithms (e.g. FIFO, round robin, rate monotonic, earliest deadline first), file and I/O support, and a sets of utility programs (e.g. linkers, compilers, library managers, debuggers, and increasingly more often visual interfaces).

A good starting point for obtaining additional information about commercial RTOS offerings are the following two WWW sites: http://cs-www.bu.edu/pub/ieee-rts/Home.html and http://msn.yahoo.com/msn/omputers_and_Internet/Operating_Systems/Realtime

5 RTOS and [Computer Aided] Design Research and Development

In this section, we survey the recent RTOS-related CAD efforts. The results are mainly due to an impetus provided by two new important trends: hardware/software co-design and synthesis techniques for systems-on-silicon.

Recent interest in hardware/software co-design[5] has opened up new avenues in RTOS research. The ability to simultaneously design the hardware platform and software allows for new trade-offs and poses new challenges in system architecture. The move toward systems-on-silicon, motivated by the ability to put one or more sophisticated CPUs and embedded RAM on a single chip, will make RTOSs increasingly important over the next several years. Good design and use of an RTOS can help increase the efficiency with which scarce hardware resources are used, while a bad RTOS can kill performance and reliability. And because systems-on-silicon are much harder and significantly more expensive to fix than printed circuit boards, performance and reliability are at a premium.

The RTOS CAD efforts can be classified in four groups: behavioral synthesis results, system-level synthesis results, design studies, and theoretical and simulation efforts.

Potkonjak and Wolf addressed synthesis of a system of multifunctional ASICs where individual tasks are under hard real-time constraints[21]. They also introduced a heuristic approach for the design and optimization of ASIC implementations which realize multiple computational tasks under hard real-time constraints on a single multifunctional ASIC[22]. Kim et al. [20] developed a set of techniques to minimize context switching time using the behavioral synthesis techniques.

System-level techniques can be grouped into two categories: uniprocessor and multiprocessor. One major trend in next-generation embedded systems is multiprocessors which may contain devices and special-purpose accelerators as well as a CPU; although these devices may not be programmable, they still allow parallel execution that must be taken into account by the RTOS. Furthermore, many embedded systems include several CPUs, often of different types. Coordination between the processing elements in the system is essential for both performance and correctness.

While there have been a high interest in the multiprocessor real-time system-level synthesis, only a few efforts have been dedicated. Lee et al.[1]developed a synthesis algorithm which generates an application-based real-time multiprocessor. The algorithm uses a three-step synthesis methodology of resource allocation, assignment, and scheduling. Li

and Wolf[2] developed a hierarchical algorithm for scheduling and allocating processes on a multiprocessor. The synthesis algorithm takes advantage of the task structure to efficiently schedule and allocate tasks to meet deadlines. Yen and Wolf addressed synthesis of distributed embedded systems with both hard and soft deadlines[13]. While all hard deadline must be satisfied, the optimization goal is to satisfy as many as possible soft deadlines. Recently, Kirovski et al. proposed a combination of system-level and compilation techniques for area minimization for a set of application being executed on core-based systems-on-silicon[33].

While there have been a large number of RTOS design studies and commercial products in several communities, a very limited efforts have been reported in design community. The most notable practical application of real-time scheduling approaches, and in particular rate-monotonic scheduling algorithms, include the inclusion of rate-monotonic scheduling as the scheduling policy for the IEEE POSIX real-time operating system standard and IEEE Futurebus+ standards[26,27,29], and the use of the generalized rate-monotonic scheduling techniques in several major advance-technology projects such as Space Station Program and the European Space Agency on-board operating system[30]. The strong endorsement of several research and development groups of the earliest-deadline-first and rate-monotonic scheduling as most suitable resource allocation policies for continuous media servers[24,25,28] and ATM switch scheduling[34] further emphasizes importance of this hard-real time scheduling approach.

Before concluding our survey of the related research efforts with design methodology and algorithmic issues, we refer to three recent publications in the CAD literature which addressed use of the rate-monotonic scheduling algorithms. Hu et al.[31] compared several hard real time scheduling policies during hardware-software partitioning of the controller for automotive powertrain module, but did not develop synthesis approach or conduct any synthesis optimization. Atlenbernd[32] generalized deadline monotonic scheduling policy along several lines, but also did not make connection to the synthesis process. Balarin et al. verified real-time operating system for an automatically controlled vehicle using scalable models[15].

6 RTOS for Systems-on-Silicon: Future Directions

In this Section we briefly outline our vision about future of the RTOS research and development directions. As Bohr said, it is difficult to predict, in particular future. Nevertheless, it seems that some trends can be identified.

There is a pending change of behavioral and system level synthesis efforts from the algorithm to architecture framework to compiler/RTOS based design methodology. Exploring the interaction between RTOS and system-level synthesis of systems-on-silicon as well as between RTOS and modern embedded real-life applications will be one of the key issues in synthesis of systems using cores.

Technology, pricing and applications trends, in particular emerging integration of DRAMs and processor logic and popularity of large data volume applications, will most likely alleviate the industrial emphasis on RTOS size minimization. On other hand, it is reasonable to expect that switching time, and in particular latency, will stay as an important design metrics.

The increasing complexity of both hardware and software will accentuate a strong need for both software and hardware debugging. RTOS benchmarking will receive significantly higher attention. Flexibility and scalability will be important. Applications trends indicate that both security, privacy, authentication, and intellectual property protection mechanisms as well as rich I/O features for interaction with sensors, actuators, and humans, will gain importance. Soft real-time systems will also gain popularity.

There is also a need for adopting a new computational and hardware models which will form sound theoretical foundation for the new filed. New technical ideas, such as dynamic code generation and profile-based resource balancing, and robust implementation will be the key criteria for achieving the outlined goals. Finally, we note the market trends for other large system software market, indicating that most likely a very small number of RTOSs will have a dominant share of the market.

7 Conclusions

Real-time operating systems have long been critical components of embedded computing systems; with the advent of systems-on-silicon, RTOSs are playing a major role in all sorts of integrated circuits. The ability to simultaneously configure the operating system, applications, and underlying hardware platform opens up new avenues for improved real-time operating systems. We believe that the next few years will see continued substantial advances in real-time operating systems.

Acknowledgments

Li and Wolf's work was supported by the National Science Foundation under grant MIP-9424410. Potkonjak's work was supported by MICRO grant 96-182 and a grant from the OKAWA Foundation.