RTOS Modeling: A Survey

Y. Jing, G. Lhairech, *Master M.A.R.S., Université Bretagne Sud*

Abstract – Embedded systems are currently used in daily life. Within for example MP3 player, PDA, programmable thermostats and other features. They're various applications of embedded systems in industry, such us robots or intelligent machines. These systems generaly use a Real Time Operating System (RTOS) in order to meet time requirement and manage application complexity.

In this work we present some RTOS models and classify different approaches of RTOS modeling, and then we propose a comparative table in order to guide system designers choosing the suitable model.

Index Terms – Embedded systems, Modeling, Operating system, Real time.

I. Introduction

Real time and embedded systems operate in constrained environments. They often need to provide their services within strict time deadlines. In addition, the growth of the complexity of systems increases the use of embedded software. All these factors lead to the use of a Real Time Operating System (RTOS).

RTOS is an operating system that guaranties the system working within time constraints. It provides especially five kinds of services; task management; inter-task communication and synchronization; timers; dynamic memory allocation; and device I/O supervision. Embedded system designers need to consider RTOS at each design phase that's why RTOS modeling comes in handly.

This paper is organized as follows: Section II presents the motivations for RTOS modeling; then Section III focuses two important points; the contribution of RTOS model in the design flow and how to build the model itself. This section concludes with a comparison between the main approaches discussed in recent works; and finally Section 4 presents the conclusion and future work.

II. Motivation

RTOS modeling can target several purposes. We have identified three of them; verification of functionnality at high level, co-design implementation and energy consumption and performance analysis. In the following sub-sections, we present a panel of models discussed in recent work.

A. High level modeling

Currently the design abstraction level has been raised into system level in order to answer the increasing complexity of systems and time to market pressure. From Figure 1 we can see that simulation time at system level is around seconds and it increases as we go deeper into the design flow (around days at technologic level). That's why system designers need tools to integrate OS (Operating System) features in high level design.

Transaction Level (TL) [9] is an emergent level used to describe system level design. It illustrates the communication between all the specification functionalities described in system design.

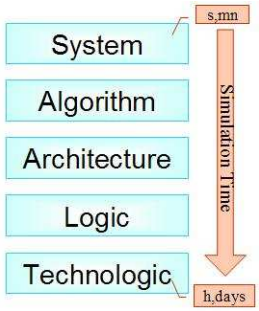

Fig.1. Impact of desing phase on simulation time

Recently several works have been focusing on abstract RTOS modeling. Hessel [1] presents an abstract model for embedded systems at Transaction Level build on SystemC language with some refinement steps for implementation at lower level. In [2] the model is similar to [1]. The main difference is that [2] model is written on top of SpecC kernel and refinement can be automatically performed. The same team has implemented another RTOS model in [4] that provides an abstraction of the key features and describes a dynamic scheduling behavior independent of any specific RTOS implementation. The [3] approach is different; it models the RTOS with a generic state machine implemented on top of SystemeC that can be configured to model different existing RTOSes.

B. Co-design implementation

To achieve the best overall solution System designers need to co-simulate hardware and software implementations. The choice of the suitable design is made by evaluating different configurations; involving RTOS implementation. Consequently we need RTOS models that offer partitioning possibility.

In [3] the RTOS model C/C++ or VHDL includes components that can be integrated into SystemeC Framework to support HW/SW co-simulation. Few models have been built in this scope. Mooney team [5] has designed a δ

Framework wich provides automatic HW/SW configuration. The framework involves the Atlanta multiprocessor RTOS kernel for Soc architectures. The user chooses Inter Process Communication (IPC) methods, Hardware and Software components he wants to add to his application. Their framework provides three kinds of hardware components SoCLC (deadlock detection); SoCDMMU (dynamic memory management); SoCLC (lock cache). After configuration, files are generated to allow co-simulation (with seamless for instance). Later the same team has designed a Real Time Unit (RTU) that moves the IPC such as semaphores and time control from software OS-kernel to hardware [6]. Results show that 36% performance gain can be obtained (with a 30 tasks application).

C. Energy consumption and performance analysis

Since many embedded systems are used in portative applications, energy consumption issue has taking more and more importance in systems design. Thus, the overhead of an RTOS must be evaluated. On the other hand, real time constraints imply hard timing requirements. System designers need to know if a target RTOS can satisfy application constraints or not. Some models or methodologies can help on performance and energy consumption analysis. The two following examples adress this issue.

[7] group has developed a simulator *SimBed* that is a high level language. However to make their simulator accurate, the binary code resulted from the simulator is the same that runs on the hardware component. Consumption analysis can be performed on their simulator thanks to the power estimation function they have added to their instruction emulator. *SimBed* gives also track of real-time jitter and response time delay, this allow performance measurements. In [8] is given a methodology for OS energy characterization in order to help designers to make a macro model of their application.

III. RTOS MODELING: COMPARISON OF DIFFERENT APPROACHES

The previous section presented several RTOS models; each one uses a particular approach. To compare them, we must first distinguish two major concepts; the granularity used to build a RTOS model and the model integration level in the design flow phases. The last point can be partlly associated with the motivation of RTOS Modeling. As follows, we explain these approaches and we'll attempt to clarify the two points mentioned above. RTOS presents an abstract layer between the application and the heterogeneous resources as shown in Figure 2. This abstraction layer can have different degree of granularity so that it involves or not the hardware features.

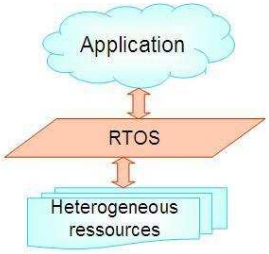

Fig.2. RTOS kernel seen as an abstraction layer

A. Approach 1

A first approach is based on TLM concepts which model a system from communication point of view. Actually, with this modeling method the RTOS is considered as a service provider entity (OS services). The model must be able to model these services and refine them into many abstraction levels (architectural one for instance). In general, OS kernel is described on top of existing languages such as SystemC and SpecC. That is the case in [1] [2] [3] [4].

In [1], a RTOS TL library was designed in SystemC and contains four models : RTOS model with primitives for initialization and preemption and resume tasks during execution; Task model, so that each task holds necessary informations to be executed and provides primitives used to allow task preemption and resume by the scheduler and other standard RTOS primitives like (notify, end, wait….); Scheduler model, wich considers all tasks are independent and have three states (Idle, Execute and Ready), it implements standard scheduling algorithms; FCFS, Round Robin, Rate-Monotonic (RM) and Earliest Deadline First (EDF); Synchronization model wich offers services to synchronize concurrent and cooperative tasks with two primitives (wait and notify).

University of Texas in collaboration with Texas Instruments [3] has developed in SystemC a RTOS model that adopts the same approach. Their model is object oriented and designs 6 components; Task Management (creation, termination, suspension, activation and priority change), Os Kernel (preemptive/non-preemptive scheduling policies with/without priority aging, semaphore creation, increment, decrement and termination), IO module, IPC, Timing and Event Handling.

University of California [2][4] has chosen SpecC as language. The approach is the same; the RTOS model provides four categories of services: OS management (Initailization), task management (creation, termination, suspension, activation, forking and joining), event handling and time modeling. With their model it is possible to refine

automatically an unscheduled application into a scheduled one. The refinement is mainly done by transforming time concerned methods into RTOS primitives.

B. Approach 2

A second approach uses instruction based techniques. This level of accuracy is suitable to model power at the processor level. In [7] for instance, the instruction model (system calls) is built thanks to measurements and extensive experiments [11]. SimBed is a model of embedded hardware system they have developed in high language and verified as cycle-accurate to within 100 cycles per million. It's used to simulate a system with a model close to the actual hardware. Princeton University [8] uses this approach for system energy macro-modeling. This method is different from the first one, it relies on a component based approach that uses features issued from an instruction level estimation tool ENSIM [12] close to previous simulator. Basically an OS is seen as a multi-entry multi-exit program (MEME-P) containing two main components SCEEP (system call entry-exit pair) and IEEP (implicit entry-exit pair) that relies on system call. The macro-modeling is performed in two phases; identifing the different components, and the generation of a test program wich allows their isolation; the second is the model adaptation. By running the simulator (an ISS) they caracterise component power consumption and build a matematical model using a regression technique.

C. Approach 3

Finally, a third approach consists of hardware implementation of the RTOS, usually done to improve performance. Here, the model is build at RTL level. To illustrate this approach we'll take the RTU model in [6]. The scheduler, message handling, semaphores and interrupt are in hardware; service calls are carried out through a set of registers and communication is carried out through a bus interfaces. Experiments discussed in the same work shows that the use of the RTU increases performance (thanks to the CPU workload reduction). In [5], was developed a framework where hardware components are designed (as we described in II.B).One of these components is System on a Chip Lock Cache (SoCLC). It contains lock variables and Pr (which stands for processor) bit location associated with each lock variable to show if the processor is waiting for the lock or not. Such solutions can help on speedup time execution (19% for a 30 tasks application).

By this classification we can see the difference between models. Parts of RTOS kernel can be performed either in software or hardware. Raising the abstraction level decreases the accuracy of the model but makes it more flexible. Whereas, models build at instruction or RTL level are near the actual RTOS but require the system designer having more details on target hardware. Anyway the main objective of modeling remains the abstraction of low level features to use at higher level for different purposes as power estimation for instance.

Table 1; comparison between RTOS models.

Models	***OS services***					***OS purposes***			
	Task managemen[3]	***Inter-task communication and synchronization***	***Timers***	***Dynamic memory allocation***	***Device I/O Supervisor***	***Behavioral analysis***	***Performance analysis***	***Energy consumption analysis***	***Co-Design ability***
[1]	Performed	events	SW model	N.A. [1]	--	Possible	--	--	--
[2]	Performed	Variables, channels(e.g. semaphores), forking and joining	SW model	N.A. [1]	--	Possible	--	--	--
[3]	Performed	Semaphores, locks, signals, memory charing, mailboxes	Simulated	Hardware	I/O module available	Possible	--	--	Simulation[2]
[4]	Performed	Events, forking and joining	SW model	N.A. [1]	--	Possible	--	--	--
[5]	Performed	Semaphores, queues, mailboxes, group of events	Available	--	--	Possible	--	--	Partitioning
[6]	Performed	Hardware semaphores	HW	--	--	Possible	--	--	Partitioning
[7]	Performed	Available	Available	--	I/O ports	Possible	System level	Instruction level	--
[8]	Performed	Available	Available	--		Possible	--	Instruction level	--

[1]: Not Available
[2]: the model offers co-simulation possibility but no hardware model is developed
[3]: see the paragraph above for details

We sum up all models features in table 1. From this table, we can see that almost all models lack on memory management. They often have the same kind of services the difference is in general. The way they implement a particular service; for example, inter-task synchronization can be carried out by semaphores, signals or events…
Models used for performance or energy consumption analysis must be quite accurate in order to mimic the real hardware; otherwise a first evaluation can be done using a model at a higher level, especially when the hardware target is not specified. RTOS models can be also used to validate a system behavior, this purpose is common for all models.

IV. Conclusion and future work

This paper prensented several RTOS models, and compared the different approaches used to build them. It also explained how and in wich design phase an RTOS model can be integrated. Finally, this paper concludes with a comparative table. Memory management modeling is missing in almost all models whereas it is an inportant criteria in energy estimation issue for instance. Our work can be helpful for choosing the appropriate RTOS model according to the designer purpose; it also gives a survey of existing models and explains them briefly.
Our future work will focus on energy consumption issue. In fact, few works have raised the problem. We project to evaluate RTOS overhead in terms of consumption and we'll study µC/OS RTOS using Xilinx enviromment. Applications are executed in *MicroBlaze* processor on wich µC/OS is instantiated. µC/OS is a portable, ROMable, scalable, preemptive, real–time, multitasking kernel that can manage up to 63 tasks. The execution time for every service provided by µC/OS (except one) is both deterministic and constant. µC/OS allows you to:

- Create and manage up to 63 tasks,
- Create and manage binary or counting semaphores,
- Delay tasks for integral number of ticks,
- Lock/Unlock the scheduler,
- Change the priority of tasks,
- Delete tasks,
- Suspend and resume tasks.

We will run several applications on this hardware and measure the current that the chip consumes during the execution. Since few work is based on real measurments, this work could give interesting data capable to build an accurate RTOS power consumption model. This work is a contribution to the ITEA European project SPICES.

References

[1] F. Hessel, V. da Rosa, I. Reis, C. Marcon, and A. Susin. "Abstract RTOS Modeling for Embedded Systems". In Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping (RSP'04), Geneva, Switzerland, June 2004.
[2] Andreas Gerstlauer, Haobo Yu, and Daniel D. Gajski. "RTOS Modeling for System Level Design". In International conference on Design, Automation and Test in Europe (DATE), pages: 10130 – 10135, Munich, Germany, March 2003.
[3] Z. He, A. Mok, and C. Peng. "Timed RTOS Modeling for Embedded System Design", In Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'05), pages 448-457, March 2005
[4] Haobo Yu, Andreas Gerstlauer and Daniel Gajski. "RTOS Scheduling in Transaction Level Models". In Proceedings of the International Symposium on System Synthesis, pages 31--36, October 2003.
[5] J.Mooney III, V. and M.Blough, D. "A Hardware-Software Real-Time Operating System Framework for SoCs". In Design & Test of Computers, IEEE, pages 44--51, Nov. 2002.
[6] J. Lee, V. J. Mooney III, K. Ingstrom, A. Daleby, T. Klevin, and L. Lindh. "A Comparison of the RTU Hardware RTOS with a Hardware/Software RTOS". In Proceedings of the ASP-DAC, pages 683--688, 2003.
[7] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang and B. Jacob. "The Performance and Energy Consumption of Embedded Real-Time Operating Systems". In IEEE Trans. Comput. Vol. 52, 11, pages 1454--1469, November 2003.
[8] T. K. Tan, A. Raghunathan and N. Jha. "Embedded Operating System Energy Analysis and Macro-modeling". In Proceedings of the International Conference on Computer Design, pages 515—522, September 2002.
[9] L. Cai, D. Gajski. "Transaction Level Modeling: An Overview. In Proceeding of CODES+ISSS, pages 19-24, October 2003.
[10] T. K. Tan, A. Raghunathan, and N. K. Jha. "EMSIM: An energy simulation framework for an embedded operating system". In Proc. Int. Symp. Circuit & Systems, pages 464–467, May 2002.
[11] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step towards Software Power Minimization," IEEE Trans. VLSI Systems, vol. 2, no. 4, pp. 1277-1284, Dec. 1994.