Performance Characterization of Real-Time Operating Systems for Systems-on-Silicon

Douglas P. B. Renaux[1,3], Rafael E. De Góes[3], Robson R. Linhares[2,3]

[1] Departamento Acadêmico de Eletrônica (DAELN)
Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Sete de Setembro 3165 – Curitiba – PR – Brasil

[2] Departamento Acadêmico de Informática (DAINF)
Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Sete de Setembro 3165 – Curitiba – PR – Brasil

[3] eSysTech – Embedded Systems Technologies
Travessa da Lapa 96, cj 73 – Curitiba – PR - Brasil

douglasrenaux@utfpr.edu.br;
rafael@esystech.com.br;
robson@dainf.ct.utfpr.edu.br

Abstract. *An RTOS is a software component that is used in the majority of the real-time embedded systems. It has a significant effect on the system's performance and reliability. This paper addresses the issue of publishing parameterized performance characteristics of an RTOS in a platform independent manner.*

Concepts of parametric timing analysis were extended to consider the performance of the processor, memory and peripherals in a parameterized way. The proposed method was applied to a commercial RTOS. Validation of the method shows results with a precision better than 10%.

Key-words*: timing analysis, WCET (Worst Case Execution Time), RTOS (Real-Time Operating System) performance characterization, COTS (Commercial Off The Shelf) software component performance.*

1. Introduction

As embedded systems complexity and diversity are constantly increasing, developers face a number of opposing needs in the design cycle. The development time and effort can be reduced with the use of both hardware and software COTS components, however, these components must be appropriately integrated and characterized in advance. High-volume production costs can be reduced with the use of the high-integration Systems-On-Silicon (SOS), however, this poses stringent demands on the design process and tools, demanding the use of hardware-software co-design and heavy use of modeling, simulation and estimation, since the actual hardware is available only near the end of the design cycle.

Current fabrication technologies allow for dies with less than 30 mm^2 to implement over 50 million transistors using below-40 nm processes. Complete systems can be implemented in a single die, including multiple processor cores, RAM, Flash, dynamic

RAM, and several types of generic and special purpose peripheral units that implement communication channels, audio and video processing, interfaces to data storage devices, among many others.

An essential COTS component used in most of current embedded systems designs is the RTOS. It provides an abstraction of the hardware and manages its resources allowing the software development team to focus on the application specific software. Although the RTOS contributes heavily to the performance and robustness of the final system, its performance is usually described very coarsely, mainly citing the context switch times and latencies. In most embedded system's designs, and particularly in those using SOS, a much more detailed performance characterization is required, so that the final system´s performance can be accurately predicted in early phases of the design. Design cycles are too lengthy and costly to allow for a second attempt to achieve the desired performance.

The aim of this paper is to propose a means of describing the performance of an RTOS. The rationale for such a description is fourfold:

1. RTOS execution times and blocking times are essential information to be used in the schedulability analysis of real-time systems;
2. when comparing different RTOSes as alternatives for a design, specific RTOS configurations can be compared from a performance point of view;
3. the application programmer can identify which RTOS services are time consuming or have execution times that are not compatible with given response time requirements;
4. when combined with the performance data of the other components of the system, the performance of the whole system can be accurately predicted and checked against the performance requirements.

It is important to notice that the performance characterization of an RTOS is severely dependant on the HW platform where it runs. The embedded world is characterized by a very large variety of hardware platforms, as opposed to the standardized hardware platforms of general computing (such as PCs and Macs). To encompass such a large variety of hardware platforms we use parameterized generic hardware models, as no RTOS provider would be able to provide performance data for every possible embedded hardware platform and their various configurations.

2. Problem Statement

A typical embedded systems development process is illustrated at a high level in Figure 1. Such a process may be used both when silicon is designed for a specific application as well as when COTS HW components are used. It is important to realize that in this process, HW and SW are designed and implemented concurrently (HW/SW co-design). Actual measurements of performance can only be done on the final HW platform in the Integration phase, however, particularly when specific SOS is designed for this application, a good estimate of system performance must be already available during the System Design phase.

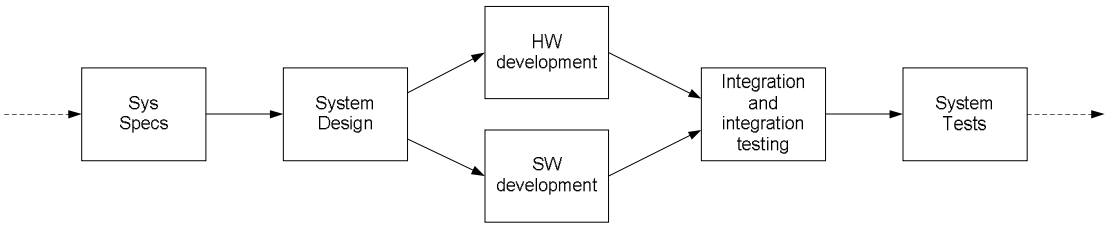

Figure 1 - Typical SOS development process

In Figure 1, the activities preceding Sys Specs and those after System Tests are not shown. In the System Design phase, the identification and characterization of COTS HW components (from libraries and catalogs of HW components), COTS SW components (including RTOS), custom HW component definition (including glue logic and application specific HW), and custom SW component definition (SW wrappers and application specific SW) is done. In this phase, the performance of the final system must be estimated based on the documented performance of its COTS components and estimated performance of the HW and SW components to be developed.

During the design and development phases, models, simulators and evaluation boards may be used to support prototyping and performance evaluations. The level of precision of the simulators and the similarities of the evaluation boards when compared to the actual hardware will define how accurate these performance estimates are.

The problem we are addressing in this research is to identify a way that an RTOS provider (or any other SW component provider) can document the performance characteristics of his product. Since RTOS are used in a wide range of HW platforms, and since the performance of the HW platform strongly affects the RTOS performance, a means is required to parameterize the performance characteristics of the RTOS. In this paper we propose such a means, and we evaluate the proposed means on a commercial RTOS.

3. Literature Review

The determination of the Worst Case Execution Time of real-time software is a subject of study for over a decade. Many conferences deal with this subject. Since 2001 a WCET Workshop is held along to the Euromicro conference. Among the vast literature available, the papers most closely related to our research are described in this section.

Colin, A., and Puaut, I. (2001) analyzed the RTEMS RTOS and identified several problems that made the analysis difficult and imprecise: unstructured code in the RTOS, use of dynamic function calls, and unknown bounds in loops. They reported an 86% overestimation on the WCET.

Sandell, D., Ermedahl, A., Gustafsson, J., and Lisper, B. (2004) reported other types of problems when analyzing the WCET of RTOS services: high dependency of execution times on the RTOS configuration; high dependency of loop bounds on the RTOS state; and high variation of execution times depending on the current mode of the RTOS.

Puschner, P. and Schoeberl, M. (2008) proposed a means of avoiding unpredictability of execution times by rethinking HW and SW implementations: (1) use of single path programming; (2) simplifying HW design; and (3) perform static scheduling of accesses

to shared memory. The achieved gain in predictability came at a high cost of always executing both the “then” and the “else” part of a decision, and discarding the results of one of the parts, as well as significantly reducing the performance of the HW.

Lv, M., Guan, N., Zhang, Y., Deng, Q., Yu, G., Zhay, J. (2009) present a survey on the five most prominent techniques for WCET analysis of RTOS. They also identified three challenges still to be resolved: (1) Parametrization of WCET; (2) Combining the WCET of the application with the WCET of the RTOS; and (3) Combining WCET analysis and schedulability analysis.

A significant step forward was achieved by Altmeyer, S., Hümbert, C., Lisper, B., and Wilhelm, R. (2008) who developed a parametric timing analysis. Instead of the tradicional way of representing the WCET of a service by a single value, they represent it by a formula that includes the service call parameters that affect the execution time. The dependency of the RTOS state was not modeled.

4. Proposed RTOS Performance Modeling

The aim is to characterize the performance of the kernel. To do that in a broad and precise manner the following information is required:

1. Execution characteristics of each service on a generic hardware platform. This is processor’s architecture specific since a change to the architecture implies in changes the machine code.
2. A model of how the service calls arguments and the RTOS state affect the service’s execution time.
3. A characterization of the memory regions that are used and their latencies, for single read, single write, burst read and burst write.
4. A model of blocking during the execution of each service. Blocking, if it occurs at all during calls to a given service, can be in forms such as: masking all interrupts, masking specific interrupts, and preventing context-switches, i.e. preventing preemption.

The RTOS performance characterization proposed here is based on a parameterized generic hardware model (Figure 2). Hence, all performance data that is provided is dependent of the parameters of the hardware, such as clock frequency, memory latencies, and peripheral latencies.

4.1. Performance Parameters Definition Process

The process to be used to determine the performance parameters of an RTOS is depicted in Figure 3. Starting with the source code of the RTOS under analysis, a static analysis is performed to determine the path that determines the worst-case execution time of each service. Then, a test case is build that exercises this path. This test case is executed and its execution is logged, at instruction level. The log, or execution trace, is then analyzed to extract information about the number of accesses to each peripheral or memory section (PRi and MRj in Figure 2). The sections of source code that cause blocking or that are dependent on arguments or state are identified during the static analysis and their execution parameters are identified as well.

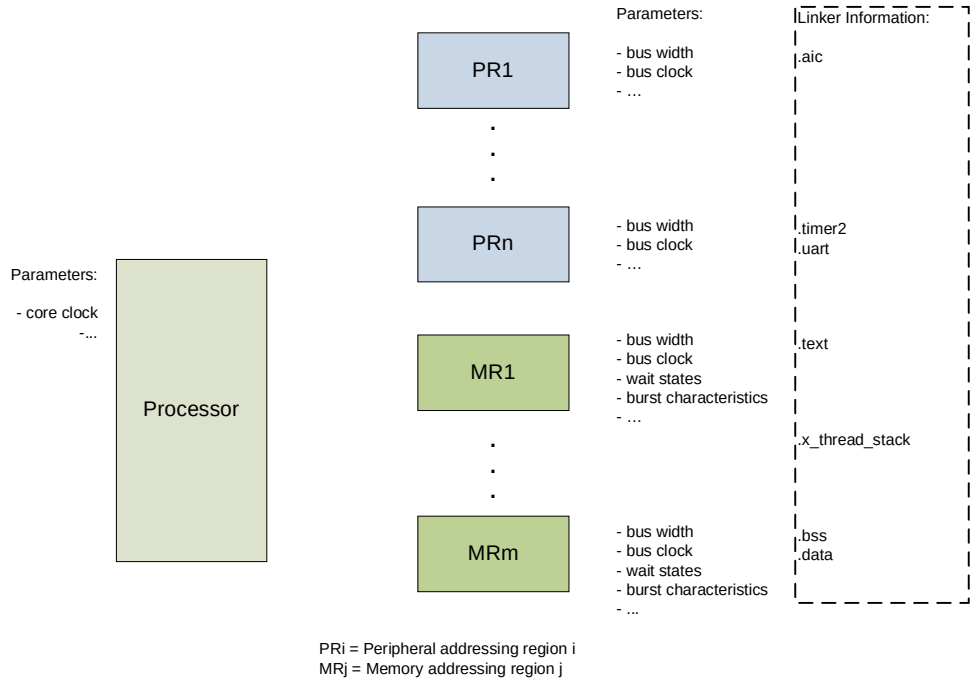

Figure 2 - Parameterized Generic Hardware Model

The process described in Figure 3 is executed by the RTOS provider (or any other SW component provider) to obtain the performance characterization of the RTOS. The use of this information is described in Section 4.2.

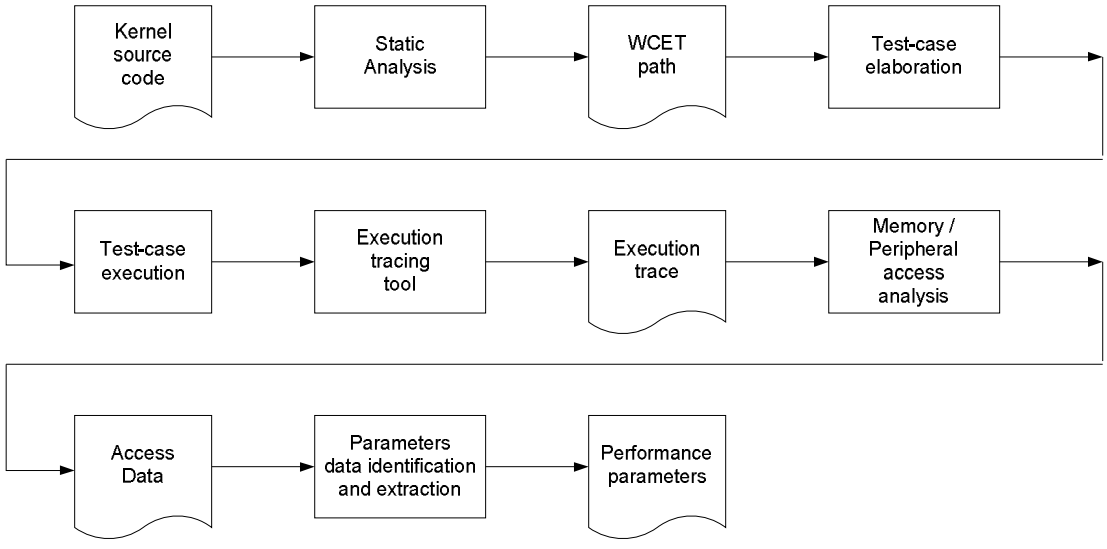

Figure 3 - Performance parameters definition process

This process combines static and dynamic techniques, aiming at obtaining the best possible results from each one. In our validation (Section 5), we used LDRA Testbed [LDRA Testbed (2010)] and PERF [Renaux, D. P. B. ; Góes, J. A. ; Linhares, R. R. (2002)] for the static analysis; here, the possible execution paths are extracted and the WCET path is identified. Then, a SW developer elaborates a test case that exercises the

WCET path. This test case is executed on real hardware, or on a instruction level simulator. The execution is monitored by a trace tool that records the execution trace. A Segger´s J-Trace unit [Segger(2010)] was used as well as the trace recording functionality of the IAR´s EWARM IDE [IAR(2010)].

At this stage of our research, we are evaluating the proposed method; hence, there are no tools yet to support the analysis of the trace. As such, the two final steps were performed manually. The execution traces of the service calls range from less than ten instructions execution to around 600 instructions. This is the case of a call to CreateThread, which is analyzed in Section 5.

4.2. System Performance Estimation

The performance of the final system can be estimated by combining information from three different sources:

1. The RTOS provider. Using the process depicted in Section 0, the RTOS provider publishes the performance parameters of all the RTOS services as well as RTOS internal activities.
2. The Hardware designer. During the design phase the hardware designer identifies the performance characteristics of the processor as well as the access characteristics to each memory/peripheral region.
3. The Software integrator. He provides the mapping between the linker sections (listed in the RTOS performance characterization) and the corresponding physical memory/peripheral regions.

Once these three sets of information are available, it is possible to estimate the WCET of each service of the RTOS on the final system, as well as the WCET of the internal activities of the kernel, such as the timer interrupt handler.

Furthermore, it should be noted that the RTOS traces change for every processor architecture, and for different configurations of the RTOS (if this is the case). Hence, the RTOS provider must perform this analysis, and publish their results, many times. For the case described here, the traces are for the X Real-Time Kernel [eSysTech(2008)] configured for ARM7TDMI processors. Hence, these traces represent the execution of this RTOS on any HW platform based on this processor core.

5. Validation

The RTOS used as testbed in this research is the X Real-Time Kernel, developed by eSysTech Embedded Systems Technologies. A technical cooperation agreement between UTFPR and eSysTech resulted in a long term collaboration between the LIT (Laboratory for Innovation and Technology in Embedded Systems) at UTFPR and eSysTech. The X Real-Time Kernel, or simply X, is representative of microkernels used in deeply embedded systems. It is basically composed of the following modules: microkernel, hardware abstraction layer (X-HAL), shell, event tracing, TCP/IP stack, USB stack, FAT 16/32 and graphics library. The structure of this kernel is depicted in Figure 4 – see eSysTech (2008).

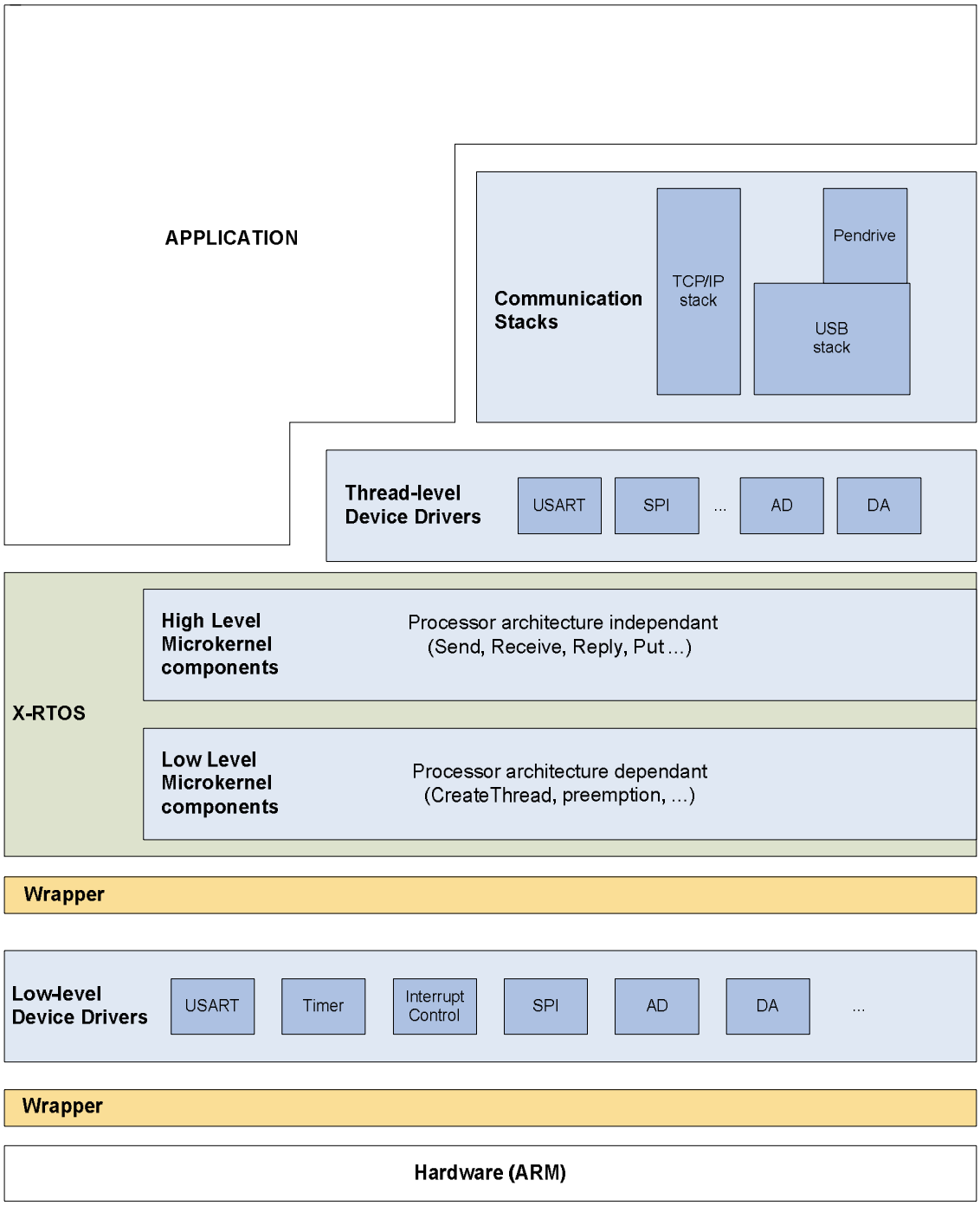

Figure 4 - X Real-Time Kernel Structure

X is being used extensively in embedded systems designed by eSysTech and by its customers. Concerning the first case (systems designed by eSysTech), the company reports [eSysTech(2010)] that, in given applications, over 95% of the embedded code is part of the kernel, hence less than 5% of the code needs to be developed to produce a new system. In such a case, where the application logic is relatively simple compared to the RTOS functionalities (scheduling, USB stack, etc) the latter plays an even more significant role in determining the performance of the system.

5.1. Performance Characterization of the CreateThread Service Call

As a result of this research, the characterization of all calls to the X's microkernel (called the microkernel´s methods in the X literature) is being performed. One of such methods was selected to be presented here: CreateThread. It concerns access to the internal data structures of the microkernel, as well as to the thread's stack. It is one of the most complex and lengthy of the microkernel's methods. It was selected as representative of the effectiveness of the proposed performance characterization.

Definition of the CreateThread Service Call:

```
TId X::CreateThread(
            void (* t_main) (uint32_t, uint32_t),
            uint32_t arg1,
            uint32_t arg2,
            const char * name,
            uint32_t stack_size,
            uint32_t put_queue_size,
            uint32_t config,
            uint32_t priority)
```

Following the process described in Section 4.1, the first activity is the static analysis. The flowgraph of CreateThread was extracted by the LDRA Testbed tool (Figure 5). The analysis of this flowgraph indicates no timing dependency to any of the parameters of the call. The only timing dependecy is to the state of the kernel's internal heap: to the current number of segments present in a linker section named x_heap. A test case is elaborated aiming at creating a given number of segments in the heap before the call to CreateThread is performed. The test case is then executed and its trace is recorded.

The kernel accesses the following linker sections, i.e. logical addressing spaces that are mapped to physical addressing spaces at link time are:

.iram_text: a code section with functions that require fast execution times;
.text: code section with most of the functions of the kernel;
.const: read only data section;
.stack: section for the stack;
.x_heap: a heap section used only for the internal data structures of the kernel.

The trace is then analyzed to identify its sections and the characterize the accesses done in each section. A trace section is a part of the trace that has the same repetition and blocking characteristics. The table below presents the result of the analysis of the trace of the execution of the test case of CreateThread. It is divided in three sections: the first is executed once without blocking; the second is executed once with blocking (interrupts are disabled), and the third is executed N times and also with blocking (again interrupts are disabled). The number of executions of the third section (N) is given by the number of segments in the x_heap.

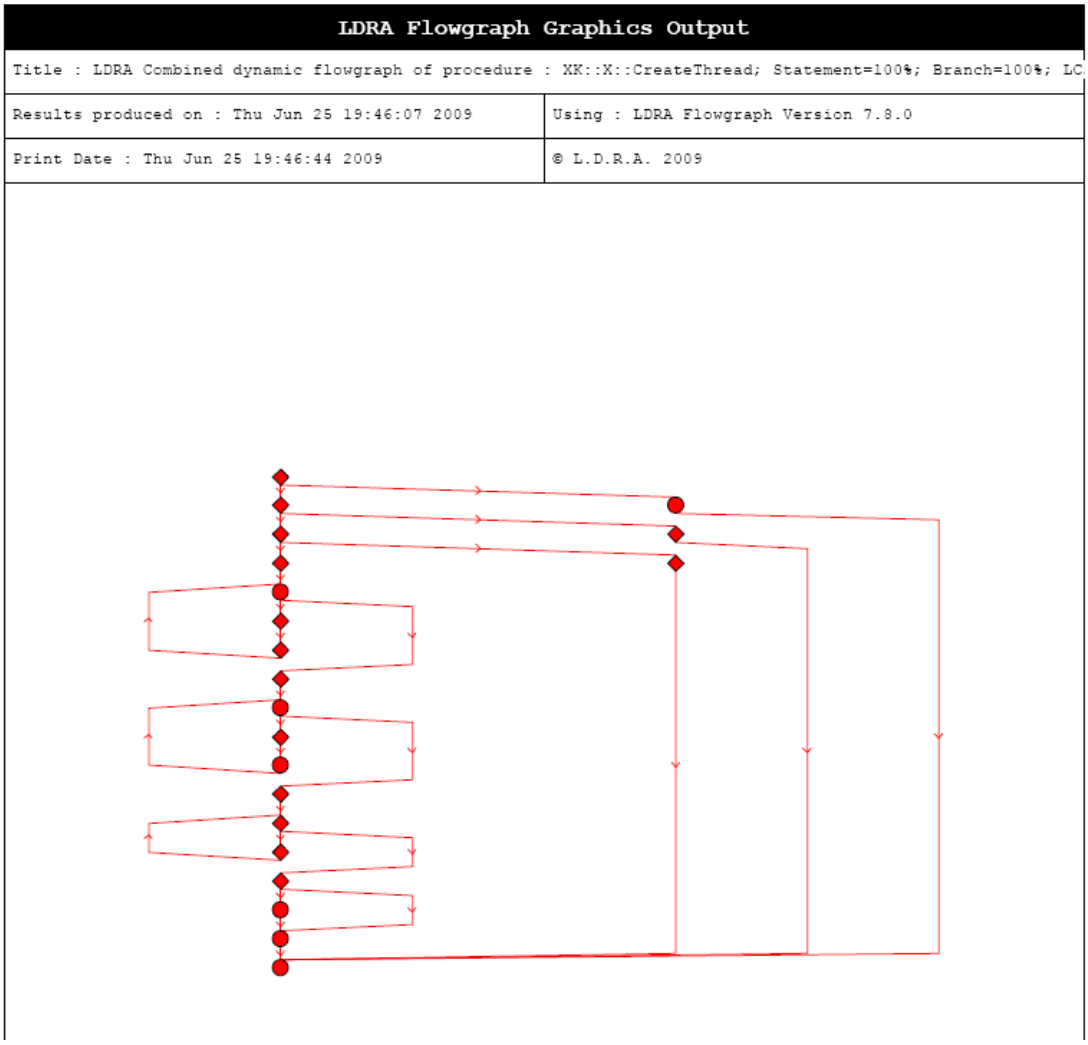

Figure 5 - LDRA Testbed Static Analysis - Flowgraph of CreateThread

The process in Section 4.1 finishes by publishing the following performance parameters for CreateThread. The values in columns “i” to “w8” are the number of accesses to each of these linker sections in each of the trace sections.

Performance characterization for the sections of CreateThread:

Sec	Rep	Block	i	t	c	s	b	r32	w32	r8	w8
1	1		3	13	1	17	4	1	0	0	0
2	1	Y	49	268	7	32	54	12	36	19	25
3	N	Y	0	18	0	0	2	2	0	0	0

Where:

Sec = code section of CreateThread; each section has different repetition and blocking charateristics.
Rep = number of times the execution of this section is repeated (N = represents the current number os segments in kernel’s internal heap).

Block = indicates if this section of code blocks preemption by disabling interrupts.

Logical sections (also known as linker sections):
i = iram_text – frequently used kernel code, usually allocated to fast memory;
t = text – code area;
c = const – constants in code area;
s = stack;
b = number of branches (jumps);
r32 = 32-bit wide read accesses to the x_heap area;
w32 = 32-bit wide write accesses to the x_heap area;
r8 = 8-bit or 16-bit wide read accesses to the x_heap area;
w8 = 8-bit or 16-bit wide write accesses to the x_heap_area.

5.2. Performance Validation

Given the performance parameters from Section 5.1, the values of WCET of a call to CreateThread were estimated for three different hardware platforms using the section mapping information provided by the software integrators of each test program (one test program per hardware platform) and the hardware access times provided by each hardware designer. The execution times were also measured on real hardware, since the three hardware platforms are available. The comparison of the estimated and measured results are shown below.

In these experiments, the call to CreateThread was performed when the kernel's heap was fragmented into 13 segments. Hence, N = 13 for the third section.

Hardware platforms description:

1. ARM7TDMI@72MHz with 16 MBytes of external 32-bit wide SDRAM and 32KBytes of internal SRAM;
2. ARM7TDMI@72MHz with 64KBytes of internal SRAM;
3. ARM7TDMI@72MHz with 1 MByte of external 16-bit wide SRAM and 32KBytes of internal SRAM.

Test on hardware platform 1:

	i	**t**	**c**	**s**	**b**	**r32**	**w32**	**r8**	**w8**	
Accesses	52	515	8	49	84	39	36	19	25	
Access time	14	14	70	70	250	200	200	200	200	ns
Total time	728	7210	560	3430	21000	7800	7200	3800	5000	ns

Total estimated execution time: 56,728 ns (addition of values in last row).

Measured execution time: 51,111 ns

Test on hardware platform 2:

	i	t	c	s	b	r32	w32	r8	w8	
Accesses	52	515	8	49	84	39	36	19	25	
Access time	14	14	14	14	45	14	14	14	14	ns
Total time	728	7210	112	686	3780	546	504	266	350	ns

Total estimated execution time: 14,182 ns (addition of values in last row).

Measured execution time: 14,137 ns

Test on hardware platform 3:

	i	t	c	s	b	r32	w32	r8	w8	
Accesses	52	515	8	49	84	39	36	19	25	
Access time	14	160	160	180	450	160	200	80	120	ns
Total time	728	82400	1280	8820	37800	6240	7200	1520	3000	ns

Total estimated execution time: 148,988 ns (addition of values in last row).

Measured execution time: 146,400 ns

An analysis of these results show that the estimated performance is both safe (estimated execution times are never lower than actual execution times) and have a low overestimation (maximum of 10%). When the memory accesses are more predictable, such as the case of internal SRAM, the overestimation was lower than 1%.

At the current stage of development, the effects of the SDRAM access buffers and cache memories are not modeled. Once this models are developed and included in our performance characterization data, we expect to achieve even better results.

6. Conclusion

This paper presents a significant contribution of considering the performance of hardware components in the characterization of the performance of an RTOS. This was achieved in a three step process: (1) the RTOS provider publishes the **parameterized performance characteristics** of his RTOS; (2) the HW designer provides the access time characterization of a specific hardware platform; and (3) the SW integrator provides the mapping of the logical sections listed by in the RTOS characteristics to the physical devices of the actual system's hardware. From the combination of the information provided by these three sources precise performance estimations can be obtained.

This research extends previous parametric timing analysis by considering the effects of the internal RTOS state and data structures and the characteristics of hardware components to obtain more accurate performance estimations.

To illustrate the method, a representative kernel service was selected: CreateThread. The same method applies to the other services of the kernel.

The process presented here was experimented on microcontrollers using an ARM7TDMI core. These cores do not use cache memories, hence, these were not considered in the model so far. Future versions of the proposed model will consider other architectures, such as the Cortex-M4 and the Cortex-A8 and will include the effects of the cache in the performance characterization. Also, we are evaluating the development of tools that would automate some of the activities (mainly Performance Parameters Identification and Extraction) that are currently performed by hand.

References

Altmeyer, S., Hümbert, C., Lisper, B., and Wilhelm, R. (2008) "Parametric timing analysis for complex architectures" In: The 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA).

Colin, A., and Puaut, I. (2001) "Worst-case execution time analysis of the RTEMS real-time operating system" 13th Euromicro.

eSysTech (2008) X Real-Time Kernel, Programmer´s Manual, 2008.

eSysTech (2010) "Code functionality assessment and performance measurements for the ISPI project", Internal Report, Feb 2010.

IAR (2010) "IAR Embedded Workbench for ARM" http://www.iar.com/website1/1.0.1.0/68/1/

LDRA (2010) "Software Development and Testing with LDRA Testbed" http://www.ldra.com/testbed.asp.

Lv, M., Guan, N., Zhang, Y., Deng, Q., Yu, G., Zhay, J. (2009) "A Survey of WCET Analysis of Real-Time Operating Systems", In: 2009 International Conference on Embedded Software and Systems. IEEE.

Puschner, P. and Schoeberl, M. (2008) "On composable system timing, task timing, and WCET analysis" In: WCET 2008.

Renaux, D. P. B. ; Góes, J. A. ; Linhares, R. R. (2002) "WCET Estimation from Object Code implemented in the PERF Environment" In: Second International Workshop on Worst-Case Execution Time Analysis, 2002, Viena. WCET'2002 - 2nd International Workshop on Worst-Case Execution Time Analysis, 2002. v. 1. p. 28-35.

Sandell, D., Ermedahl, A., Gustafsson, J., and Lisper, B. (2004) "Static timing analysis of real-time operating system code" In: 1st International Symposium on Leveraging Applications of Formal Methods.

Segger (2010) "J-Trace ARM". http://www.segger.com/cms/j-trace-arm.html