Energy Characterization of a RTOS Hardware Accelerator for SoCs

Raimo Haukilahti
Department of Microelectronics and Information Technology
Royal Institute of Technology (KTH), Sweden
&
Mälardalen Real-Time Research Centre
Mälardalen University, Västeras
rhi@imit.kth.se

Abstract

During the last decade several Real-Time Operating System (RTOS) hardware accelerators have been proposed. The accelerators perform operating system functionality traditionally implemented in software. While their effects on predictability and speedups have been studied, their impact on system energy consumption is still unknown. As a first step towards comparison of energy efficiency between HW-RTOSs and SW-RTOSs, this paper presents an energy characterization of the different operating system calls offered for an in-house developed RTOS hardware accelerator called Real Time Unit (RTU). The obtained results show that a RTU consumes about 0.16 mW/MHz for a 1.8V 0.18-micron process and that the power consumtion is independent of the function it performs. The power consumtion variances are within 4 percent of the average value. Even during the idle periods huge amount of power is wasted due to unwanted activity triggered by the clock. We believe that if a HW-based RTOS is to beat a SW-RTOS based system in terms of energy consumtion, power optimization techniques such as gated clocking needs to be used for the hardware accelerator.

1 Introduction

RTOS are gaining popularity when building todays complex embedded systems. They provide a hardware abstraction inorder to simplify the user interface and they act as a resource manager for applications. RTOSs provide helpful facilities such as scheduling, support for periodic and aperiodic tasks, semaphores, and inter-process communication.

Several specialized hardware units have been proposed [8, 2, 7, 10] to boost operating system performance and to increase predictability in the system. By moving OS functionality traditionally performed in software to a specialized hardware unit system call speedups up to 5 times has been reported [9] compared to conventional software OS.

During the last years the interest for lowering the power and energy consumtion of digital systems has increased. Energy efficient designs can be achieved by applying power optimization techniques at each level of abstraction and for every component in the system, including the operating system. Studies show that the operating system account for a significant fraction of an embedded systems energy consumption [5, 4]. Work by Acquaviva et al. also indicates that the knowledge of the energy behaviour of the RTOS is important for the effectivness of power management policies [1].

As a first step to investigate possible energy savings when adding a HW-RTOS kernel to a SoC, an energy characterization of the different system calls performed by an inhouse developed RTOS hardware accelerator called RTU is performed. Until now there has been no reports describing the power consumtion of RTOS hardware accelerators.

There are several reasons to believe that a RTOS utilizing a specialized hardware accelerator for the operating system could beat a conventional SW-RTOS. A RTOS hardware accelerator replaces parts of the SW in a RTOS with specialized hardware and therefore the SW part of the RTOS decreases significantly. Specialized hardware generally are more efficient in terms of power and performance than general purpose designs. Also since the SW-part of the RTOS decreases when utilizing an accelerator, the number of instructions executed on CPU and the number of instruction fetched from memory decreases. Moreover, cache misses will also slightly be reduced since there is less competition of the cache lines when the amount of software is scaled down. In a conventional SW-RTOS the application is interrupted every OS clock tick by the kernel to traverse through task queues to check if a task switch needs to be done. In a system with the RTU running in parallel with the rest of the system, the application will only be interrupted when a task switch needs to be performed.

This paper is organised as follows. Section 2 describes the real-time accelerator and its functionality. Section 3 presents the methodology to characterize the power and energy consumtion for the different services provided by the RTU. Section 4 presents the results obtained from simulations and analyses the results. The paper is summarised in section 5 with some concluding remarks and section 6 points out directions on future work.

2 RTU - Real-Time Kernel in Hardware

Hardware support to increase performance and predictability in real-time operating systems have been proposed in [8, 2, 7, 10]. The Real-Time Unit, RTU by Lindh et. al. [2, 7], is a co-processor with support for real-time kernel services such as process scheduling and management (create, terminate, etc), inter-process communication (IPC, message send/receive), synchronisation (semaphores), and I/O interrupt handling. The RTU runs in parallel with the target system's CPU. Although the functionality is implemented in hardware a thin software layer abstracts the hardware from the application programmer. The CPU interface with the RTU by memory-mapping to its CPU-independant register interface. Via this interface, system-calls are placed by writing to dedicated system-call registers. The handshaking scheme guarantees the functionality independent of CPU, bus, and RTU clock frequency. To gain the best speedup, the accelerator should be placed as close as possible to the CPU to minimize the register access times. Hence, a SoC solution with a RTU on-chip attached to the local CPU bus is an attractive solution, compared with having it off chip.

Figure 1 shows the basic building blocks of the RTU. The core part is the scheduler which schedules processes on-line (pre-emptive priority scheme) and dispatches process execution. Connected in between the scheduler and the programming/bus interface, a set of functional modules implements the various services in the RTU, such as management of the scheduler, IPC, semaphores, clock and timer management. Process context-switching is notified to CPUs using interrupts causing handlers in software to perform the actual context-switching.

The RTU used in this paper supports 16 tasks, 8 priorities and 4 external interrupts. All timers are handled with a resolution of 1 uS.

3 Methodology

The RTU was synthesized and power optimized according to flow shown in figure 2 using the *Synopsys Design Compiler* tool set using a 1.8V UMC 0.18 micron process [12]. After the synthesis process the resulting gate level netlist is simulated using a RTL-level simulator called *Modelsim*. Throughout the whole simulation the testbench stimulates the RTU with system calls while the switching activity is recorded. A power optimization tool called *PowerCompiler* then optimizes the design with respect to the power consumtion using the swithing activity file generated from the gate-level simulation.

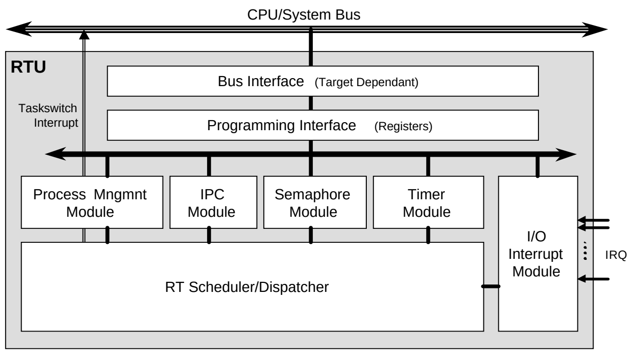

Figure 1. Basic building blocks of the RTU

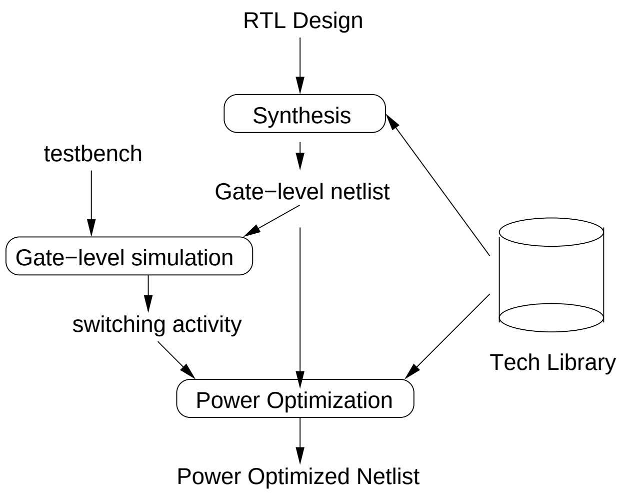

Figure 2. Synthesis and Power Optimization flow

To characterize the power consumtion for the different calls new simulation of the power optimized gate-level netlist is done with a testbench that generates one switching activity file for each system call performed by the RTU. The execution times are also recorded to be able to calculate the energy for each call. For the power analysis a tool called *Synopsys DesignPower* is used. The tool performs power estimation at gate-level for each one of the swithing activity files corresponding to a system call. DesignPower can predict the average power within 10 to 25 percent of power analysis results, using a transistor-level simulator such as SPICE. Figure 3 shows the power analysis flow.

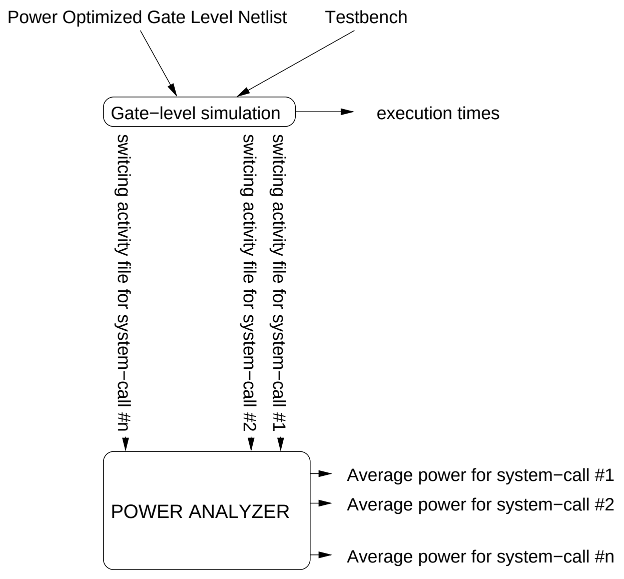
Power Optimized Gate Level Netlist
Testbench
Gate−level simulation
execution times
switcing activity file for system−call #n
switcing activity file for system−call #2
switcing activity file for system−call #1
POWER ANALYZER
Average power for system−call #1
Average power for system−call #2
Average power for system−call #n

Figure 3. Power analysis flow

Since the RTU works in parallel with the CPU it is important that during the idle period (when no system call requests arrive to RTU) the unit should consume as little power as possible. Hence, it is also important to analyse the power consumption during idle periods.

4 Results

By using DesignPower for automated power optimization for the RTU, the average power consumtione decreased with about 40 percent to about 2.5 mW when running the RTU at 16 MHz. If translated to mW/MHz the result is 0.16mW/MHz which is approximately the half of a ARM7TDMI core [11]. Switching power comprises 96 percent of the total power consumtion, hence leakage power is ignored in this analysis. Since there is not much functional activity inside the RTU while not servicing any requests from the CPU one could excpect that the power consumption during idle periods would be lower than when performing a system call. But the results from the power estimations shown in table 4 states the opposite: the power consumption is independent of what function the RTU performs. The variations from the average value is less than 4 percent. The energy consumtion of the different calls range from 4.9 nJ to 11.8 nJ due to differences in execution times of the system calls.

Even though there is not much functional activity within the RTU it consumes a considerable amount of power. This is caused by all the unnecessary switching activity. Recall that even if input signals of the circuit are constant there could be a lot of switching activity for transistors caused by the clock. The power consumption for a conventional flip-flop without activity on the input is 42 percent of the consumption when switching the input every cycle [6]. This is one explanation of the obtained results. Also, the clock network consumes 0.55 mW of power constantly which is about one fifth of the total power consumtion. Another reason is that the OS tick is 1us in this simulation which means that the timer queues inside the RTU is traversed continiously.

System Call	***Power***	***Duration***	***Energy***
thread_create	2.53 mW	41 cycles	6.43 nJ
tsw_on	2.52 mW	51 cycles	7.96 nJ
thread_block	2.50 mW	68 cycles	10.54 nJ
thread_yield	2.47 mW	77 cycles	11.79 nJ
thread_delete	2.48 mW	86 cycles	2.11 nJ
thread_start	2.53 mW	41 cycles	6.43 nJ
thread_delay	2.45 mW	68 cycles	10.33 nJ
init_period_time	2.52 mW	41 cycles	6.41 nJ
start_period	2.62 mW	41 cycles	6.66 nJ
wait_for_next_period	2.46 mW	68 cycles	9.46 nJ
semaphore_create	2.48 mW	41 cycles	6.30 nJ
semaphore_pend	2.56 mW	56 cycles	8.89 nJ
semaphore_release	2.55 mW	41 cycles	6.48 nJ
irq_init	2.45 mW	32 cycles	4.86 nJ
irq_wait	2.44 mW	68 cycles	10.29 nJ
task_switch	2.50 mW	51 cycles	7.90 nJ
idle	2.46 mW	N/A	N/A

Table 4: Power and energy characterization of the system calls

5 Conclusions

This paper presents an energy characterization of a RTOS hardware accelerator called RTU. Simulations show that the power consumption of the RTU is almost independent of what action it performs. The unit consumes about 2.5 mW for a 0.18 micron process at a clock frequency of 16 MHz. Also, the results show that power consumption during idle periods are approximately the same as during system calls. The variation from the average power consumtion is less than 4 percent. Since the RTU works in parallel with the rest of the system it will waste power during the idle periods. Hence, we believe that the RTU needs to be further power optimised to be able to compete with conventional SW-RTOSs. Since many of the flip-flops in the RTU are non-active most of the time and waisting power for every clock cycle, clocked gating looks like a promising technique to decrease the power consumtion during idle periods.

6 Future work

To decrease the power consumption significantly a low power version of the RTU will be designed by using techniques such as gated clocking. Both automatic insertion of gate clocks with tools and manual insertion will be considered. After characterization of the power-optimized RTU, the power and energy consumtion of the remaining SW-layer of the RTOS needs to be modeled. A conventional RTOS energy characterization with and without a RTU will be compared. To model the energy consumtion of the SW-RTOS system calls we plan to extended the architectural simulator SimpleScalar [3] to be able to simulate a whole application together with a RTOS. For applications that heavily use the RTOS functions we believe there is great potential for energy reduction by using a power optimized RTOS hardware accelerator.

7 Acknowledgements

This work is beeing supported by the Swedish Foundation for Strategic Research (SSF) through the Intelect program and is supervised by associate professors Axel Jantsch and Lennart Lindh.

References

[1] A. Acquaviva, L. Benini, and B. Ricco. Energy characterization of embedded real-time operating systems. In *Workshop on Compilers and Operating Systems for Low Power*, 2001.

[2] J. Adomat, J. Furunas, L. Lindh, and J. Starner. Real-time kernel in hardware rtu: A step towards deterministic and high performance real-time systems. In *8th Euromicro Workshop on Real-Time Systems*, LAquila, Italy, June 1996. EUROMICRO.

[3] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. pages 59–67, February.

[4] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob. The performance and energy consumption of three embedded real-time operating systems. In *CASES'01*, November 2001.

[5] R. Dick, G. Lakshminarayana, A. Raghunathan, and N. Jha. Power analysis of embedded operating systems. In *Proceedings of Design Automation Conference*, 2000.

[6] M. Hamada, T. Terazara, T. Higashi, S. Ktabayashi, S. Mita, Y. Watanabe, M. Ashino, H. Hara, and T. Kuroda. Flip-flop selection technique for power-delay trade-off. In *Digest of Technical Papers, ISSCC'99*, pages 270–271, February 1999.

[7] L. Lindh, J. Starner, J. Furunas, J. Adomat, and M. E. Shobaki. Hardware accelerator for single and multiprocessor real-time operating systems. In *Seventh Swedish Workshop on Computer Systems Architecture*, Goteborg, Sweden, June 1998.

[8] L. D. Moleksy, K. Ramamritham, C. Shen, J. A. Stankovic, and G. Zlokapa. Implementing a predictable real-time multiprocessor kernel - the spring kernel. In *IEEE Workshop on Real-Time Operating Systems and Software*, May 1990.

[9] T. Nakano, Y. Komatsudaira, A. Shiomi, and M. Imai. Performance evaluation of stron: A hardware implementation of a real-time os. E82:2375–2382, November 1999.

[10] T. Nakano, A. Utama, M. Itabashi, A. Shiomi, and M. Imai. Hardware implementation of a real-time operating system. In *Proceedings of TRON'95*, pages 34–42, 1995.

[11] Arm processor selection guide, 2001. http://www.embedded.com/directories/embedded/arm2001/.

[12] esi-route/11tm high performance standard cells. Virtual Silicon Technology Inc Sunnyvale, CA 94089-1116, 2001. http://www.virtual-silicon.com.