Context Switching Time and Memory Footprint Comparison of Xilkernel and µC/OS-II on MicroBlaze

Gökhan Uğurel[1] and Cüneyt F. Bazlamaçcı[2]

[1]Dept. of Image Processing, MGEO Division, Aselsan Inc. Ankara, Turkey
gugurel@mgeo.aselsan.com.tr
[2]Dept. of Electrical and Electronics Eng., Middle East Technical Univ., Ankara, Turkey
cuneytb@metu.edu.tr

Abstract

Using soft processors is an increasingly encountered trend in real-time embedded system design. If a system uses a field programmable gate array (FPGA) platform, one can save area, power, money and more by embedding a soft processor onto this FPGA platform. Another trend is using a real time operating system (RTOS) for microprocessors or microcontrollers in real-time embedded systems. RTOSs help software people in meeting the critical deadlines of the real-time environment with their deterministic and predictable behavior. In this paper, we first discuss the advantages and disadvantages of using a soft processor and give a brief description of Xilinx's soft processor MicroBlaze. We then make a simple comparison of standalone (having no RTOS) systems with systems running an RTOS and give a brief introduction of two existing RTOSs, namely µC/OS-II and Xilkernel and the benchmark criteria for comparing these. We finally compare µC/OS-II and Xilkernel over the MicroBlaze platform in terms of their context switching times and memory footprints.

1. Introduction

Nowadays, more and more embedded systems are using field programmable gate arrays (FPGAs) to control and process data by making use of inherent parallelism and flexibility concepts of FPGAs. Designers using FPGAs can choose and implement the exact amount and type of peripherals that are needed for the requirements of their application, having also the freedom of changing them while the design process is continuing. Suppose that a system designer prepares the requirements of an incoming project before the design phase of the product as usual. There is always a high possibility that these requirements are changed by the customer after the design phase starts. If the system designers decide to use for example a microprocessor of a particular type at the beginning of the project, software engineers may experience difficulties to fulfill the incoming requirements later because of the inflexible hardware architecture of this particular microprocessor. By using FPGAs on the other hand, software of the product may be protected and may be implemented in a processor independent way and the designers may not suffer from processor obsolescence.

If the decision is to use FPGAs in a project, designers can have more advantages by opting for soft processors embedded in FPGAs. Today's embedded systems must be power-efficient, sufficiently small and above all, cheap, to be commercially viable. If an embedded design uses a microprocessor, one needs to have an extra flash memory and RAM for booting the software when the system is powered up. However, FPGAs have built-in flash memory and RAM, so designers can save area, power and money by not using such extra peripherals. As a matter of fact, if a standard processor is sufficient to fulfill the requirements of an embedded project, it may be wise to use it. But if an FPGA is already employed for some other purposes then it may be cheaper and more area-efficient to use an embedded processor in the design [1].

Using an embedded soft processor on the other hand has some disadvantages. Because of the integration of the hardware and software platform design, the design tools are more complex and relatively immature compared to standard processor design tools.

Using a real time operating system (RTOS) on processors is another trend that designers increasingly follow due to RTOSs' deterministic behavior and efficient resource management characteristics. Ability to create tasks to handle and distribute huge sized codes, existence of scheduling algorithms to manage the tasks and efficient interrupt handling and faster memory allocation are some of the many advantages of using an RTOS.

RTOS comparison according to different benchmark criteria is useful for those who want to use RTOSs on their systems but who are not sure which one to use. Designers choose an appropriate RTOS for their design by considering their requirements of course. For example, if memory is critical, it is wise to choose an RTOS with the lowest memory footprint specification. There are various benchmark criteria in the literature for RTOSs defined for various purposes but we believe that there is not enough research done on porting these criteria to a specific processor to evaluate the performance of RTOSs according to these ported criteria. In [2], 16 RTOSs are evaluated according to RTOS datasheets and websites, four of which are then shortlisted according to the benchmark criteria defined by the author. This shortlist of RTOSs is then finally compared on a small scale microprocessor.

If you are a soft processor user and decide to use an RTOS on your processor, your chance of finding a survey including a detailed comparison of RTOS products on soft processors is even lower. In this paper, two RTOS candidates that can run on MicroBlaze are compared according to their context switching times and memory footprint data.

The outline of the paper is as follows. In Section 2, MicroBlaze is introduced briefly. Comparison of standalone (having no RTOS) systems with systems running an RTOS is given in Section 3. In Section 4, µC/OS-II and Xilkernel are introduced, and benchmark criteria for comparing these RTOSs are discussed. Section 5 gives the comparison results of the context switching times and memory footprint data of µC/OS-II and Xilkernel and section 6 finally presents the conclusions.

2. MicroBlaze

MicroBlaze is Xilinx's soft CPU core implemented using FPGA logic cells. Having a rich instruction set optimized for embedded applications and 70 configuration options, designers can create either a very small footprint embedded microcontroller or a high performance embedded computer running various RTOSs [3]. There are various options that affect the performance of MicroBlaze. 'Area-optimized' MicroBlaze occupies less logic cells in FPGA, leaving more cells to other peripherals. 'Performance-optimized' MicroBlaze is designed for applications that demand a faster processor. In Table 1, performance data of MicroBlaze v8.10a is presented, which may be used to compare a 'soft' processor's performance with a 'hard' one.

Table 1. MicroBlaze Processor v8.10 Performance Data [3]

Microcontroller Configuration	
MicroBlaze with local memory and debugger, UART, timer	
Performance-Optimized MicroBlaze	Area-Optimized
Virtex-6 FPGA (-3)	
307 MHz	241 MHz
(5788 LUTs)	(5118 LUTs)
Spartan-6 FPGA (-3)	
154 MHz	131 MHz
(3157 LUTs)	(2447 LUTs)

3. Standalone versus 'with RTOS' option

Real-time embedded systems must process information and give a response to outside world within a critical specified interval. Handling interrupts and switching from task to task should be as fast as possible to meet the hard requirements of real time constraints. If a real-time system with very 'loose' requirements is under consideration, software engineers tend to use 'endless loops' in their architecture [4]. For clarifying 'loose' requirements, one can say that such a system has a limited interaction with the outside world and does not have strict timing requirements. However, when a software designer use 'endless loops' in the solution due to 'loose requirements', he/she should be aware that interrupts can only be polled at each execution of the loop leading to a slower response to the outside world.

If the timing requirements are much tighter, use of 'endless loops' strategy might not be possible. Faster response times are needed then for outside world interaction, which may not be possible by using 'endless loops'. Also thousands or maybe more lines of code make it hard to control the software. Bugs appear more frequently and they are hard to detect as code size grows. This is the point where software engineers need something else to distribute the duty of the overall software to smaller but specialized units and 'control' time in a more reliable and efficient manner.

A standalone system possesses a processor, which has no operating system running on it. By running an RTOS on such a processor, the resources of the embedded system might be managed more efficiently. Using an RTOS, 'tasks' can be created to perform the duty. Priorities can be assigned to these tasks, i.e., software engineers may decide which functions of their software are more important than others. Another feature of RTOSs named as semaphores increase the predictability of the software by helping in switching the tasks safely. Each RTOS company provides a different set of application programmers interfaces (APIs), but in summary, nearly all of these provide fast memory allocation, preemptive scheduling and deterministic latency. The more software engineers have precise information about what's going on in their systems, the more their software becomes reliable.

4. Xilkernel vs µC/OS-II

There are many third-party companies giving RTOS support for Xilinx soft processor MicroBlaze. In Table 2, a list of some third-party companies that supports MicroBlaze and their RTOS products are given [3].

Table 2. Third-Party RTOS Companies Supporting MicroBlaze

Company	**Product**
eSOL Co., Ltd	PrKernel (µITRON4.0)
Express Logic	ThreadX®
Mentor Graphics ESD	Nucleus Plus
Micriµm	µC/OS-II
MiSPO	NORTi/ulTRON
PetaLogix	uClinux and Petalinux 2.6

Apart from these, Xilinx has its own RTOS, named Xilkernel, for MicroBlaze. The most important advantage of using Xilkernel is that it is shipped with Xilinx and is therefore highly integrated into the design tools of Xilinx, making it possible to configure and build an embedded system using Xilkernel in minutes [5].

µC/OS-II is also one of the popular RTOSs among companies that are supporting MicroBlaze. It is easily portable to MicroBlaze and the board support package of µC/OS-II can be ported to any MicroBlaze project with little modification. µC/OS-II is widely used in industry and also used for research purposes on RTOSs.

There are various benchmark criteria for comparing RTOSs. In [2], these criteria are summarized as; language support, tool compatibility, system service APIs, memory footprint (ROM and RAM usage), performance, device drivers, OS-awareness debugging tools, technical support, source/object code

distribution, licensing scheme and company reputation. However, we believe that MicroBlaze needs its own specific benchmark criteria in order to be able to compare RTOSs on MicroBlaze. In this paper, we compare only the context-switching and memory footprint performance of Xilkernel and µC/OS-II as a starting point. *Context-switching time* is the time between the last instruction of one task and the first instruction of the next task. It can be interpreted as the overhead that an RTOS cause for switching between tasks. *Memory footprint data* is the size of the RAM and ROM spaces needed for RTOSs to use for running accurately. The more APIs that RTOSs support, the bigger the memory footprint data is. To form a rational comparison in terms of memory footprint, both RTOSs must have the exact features enabled, thus size of codes of the same features can be compared. As future work, a detailed benchmarking list for MicroBlaze needs to be created and all RTOSs supporting MicroBlaze should be compared using these criteria.

5. Results

In the following, our experimental setup (Fig. 1) is briefly described.

Fig. 1. Experimental setup

Our board consists of an FPGA (Xilinx Spartan 3AN) and an additional 1Mbit SRAM to be used for the total memory need of the application project on MicroBlaze. FPGA code is downloaded to the board via the JTAG connector of the board. FPGA code does not change if there is no change on the hardware design of MicroBlaze project, so debugging the software is done by simply downloading the software code to the SRAM every time the code is updated.

On MicroBlaze platform, context-switching performance of Xilkernel and µC/OS-II is compared using the measurement method in Table 3.

We ran this measurement algorithm for 30 seconds for both RTOSs on MicroBlaze and determined the number of context switching operations occurred during this time for both RTOSs separately. This measurement technique also contains the time consumed by those functions that help to switch control between tasks. However switching task functions can spend varying times on Xilkernel and µC/OS-II, which is the drawback of our measurement method that needs to be noted for future works. On µC/OS-II, "OSTaskSuspend" and "OSTaskResume" system calls are used for the purpose of switching tasks.

Table 3. Measurement method of context-switching

Task1 (Higher Priority)	Task2 (Lower Priority)
Running	Sleeping
Switch control to Task2	Sleeping
Sleeping	Running
Sleeping	When Task2 starts to run, switch control back to Task1
Running	Sleeping

On Xilkernel, which uses POSIX thread architecture, there is no system call to suspend and resume a task. Semaphores, mutexes and conditional variables can be used for scheduling tasks. We chose semaphores in our case to switch the tasks.

Another important note is about running frequencies of the RTOSs and the MicroBlaze. Both RTOSs are configured to run on 1 KHz while task stack sizes are the same. Xilkernel's maximum operating frequency is 1 KHz, which is the main reason of choosing the common running frequency of RTOSs as 1KHz. MicroBlaze operating frequency is 80 MHz and both RTOSs run on a MicroBlaze which is generated with the same synthesis options on ISE (Xilinx FPGA design tool) such as optimization goal, optimization effort and performing timing-driven packing.

After running the benchmark code for both RTOSs, the results that are presented in Table 4 are obtained.

Table 4. Context Switching Count of Xilkernel and µC/OS-II

Xilkernel	µC/OS-II
2.190.150	360.596

We then conclude that Xilkernel performs the switching task functionality six times faster than µC/OS-II. It is hard to give exact time values in context switching comparisons since timing depends on processor speed, architecture of the processor (especially on soft processors like MicroBlaze) and the lack of support of RTOS timers for accurate results. Both RTOSs support timers that measure time in terms of "OS Ticks" which then leaves us with a bad resolution while working on frequencies such as 1 KHz.

It may also be proposed that both RTOSs should use semaphores when switching tasks for a more reliable comparison. If semaphores are used on µC/OS-II however, performance figures do not get any better, which leads us to the results in Table 5.

Table 5. Context Switching Count of Xilkernel and µC/OS-II when µC/OS-II uses semaphore control

Xilkernel	µC/OS-II
2.190.150	290.220

In terms of memory footprint data, both RTOSs have no problem of fitting into memory since codes of RTOSs reside in the additional 1Mbit SRAM on our board. In order to compare memory footprint data, compiled sizes of a sample test code will be given for both RTOSs. Given compiled code size data includes the user code for testing purposes and the static code of

RTOSs' features. Sample test code is a simple endless loop and negligible in size, approximately the same for both test codes, so one may think that the data presented shows only the size of the static codes of RTOSs. For a meaningful comparison, all the unnecessary features that are not used in the test code (like mailboxes, message queues, etc.) are disabled.

Two different cases are generated and measured. First test code is a sample code in which RTOSs have the semaphore feature enabled. Second test code is the same sample code in which RTOSs don't have the semaphore feature enabled. Namely, both RTOSs are compared according to their memory footprint with semaphore feature enabled and then disabled. Results are given in Table 6.

Table 6. Memory Footprint Data Results

Semaphore Feature Enabled					
	text	**data**	**bss**	**dec**	**hex**
Xilkernel	23918	468	28562	52948	ced4
µC/OS-II	16938	352	31838	49128	bfe8
Semaphore Feature Disabled					
	text	**data**	**bss**	**dec**	**hex**
Xilkernel	20010	468	27022	47500	b98c
µC/OS-II	15978	352	31838	48168	bc28

We conclude that, two RTOSs don't have much of a difference in terms of memory footprint. Interpreting the results, we may state that Xilkernel's semaphore feature code size is nearly six times bigger than µC/OS-II's semaphore feature code size. We may also state that all the code sizes mentioned in Table 6, won't fit into Xilinx Spartan 3AN FPGA's internal RAMs. If one wants to use one of the RTOSs above on MicroBlaze running on Spartan 3AN, an external memory should be placed on the board.

6. Conclusion

Designers of embedded systems are considering use of a soft processor option more frequently since they feel, of course depending on the application, free to add or remove peripherals according to the requirements of the project. Also processor performance can be tuned according to the needs and components on the board can be reduced. Processor obsolescence is also not a problem anymore. If the requirements force the designer to manage hard timing constraints on running tasks and to have more precise latencies in terms of context switching and interrupts, designers are choosing RTOS solution increasingly. Selecting an RTOS that most suits the requirements of a project should not be difficult for a designer, which means, no time should be wasted by the designer for researching all of the available RTOSs. A comparison of available RTOSs using the benchmark criteria that fulfill designers' need will accelerate their work and provoke RTOS companies to strengthen their products on areas where they are weak. In this paper, two important RTOS products on MicroBlaze are compared according to critical benchmark criteria, namely the context switching time and memory footprint. Xilkernel is turned out to be faster than µC/OS-II on context switching times as a result. But there is no clear winner on memory footprint comparison. For future work, the benchmark criteria should be enriched by combining the literature with the requirements that soft processor usage imposes. Also, all of the RTOS products that support MicroBlaze should be added to the comparison.

7. References

[1] B. Fletcher, "FPGA Embedded Processors, Revealing True System Performance", in *Embedded Systems Conference*, San Francisco, USA, ETP-367, March 2005. (available at: http://www.xilinx.com/products/design_resources/proc_central/resource/ETP-367paper.pdf)

[2] T. Anh and S. Tan, "Real-Time Operating Systems For Small Microcontrollers", *IEEE Micro,* vol. *29,* no.*5,* pp. *30-45*, Sept.-Oct, 2009.

[3] Xilinx, "MicroBlaze Soft Processor Core", http://www.xilinx.com/tools/microblaze.htm.

[4] D. Kalinsky, "Context Switch", *Embedded Systems Programming*, pp. *94-105,* February, 2001.

[5] A. Rönnholm, "Evaluation of Real-Time Operating Systems for Xilinx MicroBlaze CPU", M.S. Thesis, Department of Computer Science and Electronics, M¨alardalens University, Vasteras, Sweden, 2006.