Comparison of Xilkernel and uc/osII on Microblaze

Ravi Jani[1] , Kunjal Mehta[2]
[1]Student, [2]Professor
Dept. EC, LJIET, Ahmedabad

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.**

Keywords— RTOS; Microblaze; Soft Processor; uclinux.

I. 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.

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 [1]. 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.

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 [1]. 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.

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

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)

Standalone versus 'with RTOS' option

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 bepossible 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.

Xilkernel vs iC/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 [1].

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 [1].

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.

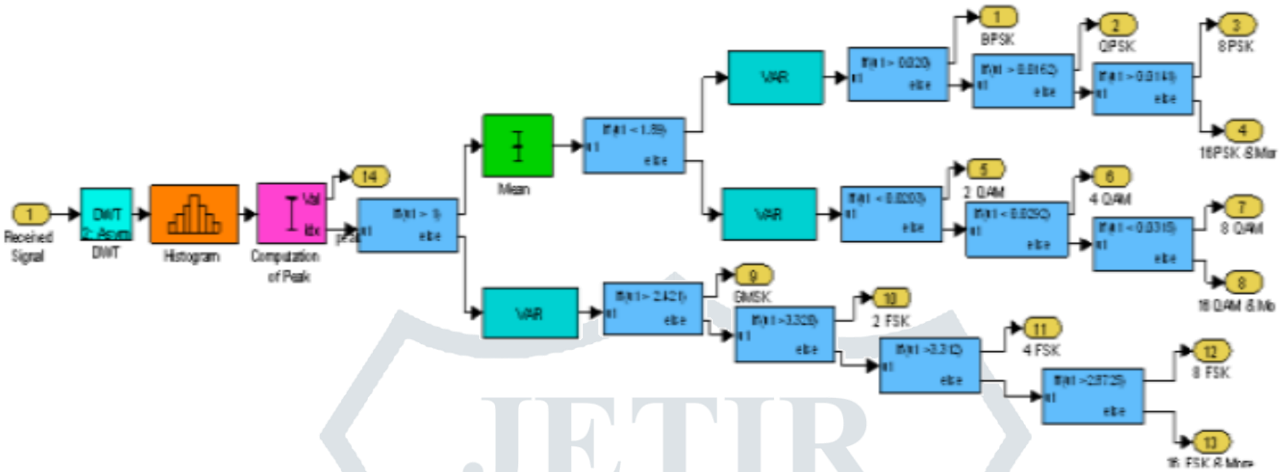

Fig.1. Simulink Model of the Proposed Algorithm

The histogram is computed for the detailed wavelet coefficients and the number of peaks is recorded. If the number of peak is 1 then the identifier identifies that the received signal is QPSK modulated signal otherwise it identifies as GMSK modulated signal.

The QPSK demodulation is done by means of coherent in-phase and quadrature phase demodulation. The output of the demodulator is passed through the decision circuits and output of the two decision circuits is multiplexed to get a original message signals. In GMSK demodulation the received signal is splitted into in-phase and quadrature phase components and filtered and it is passed through simple FM discriminators and both the components are added to get the demodulated signal.

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 3 are obtained.

Table 3. 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..

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.

II. CONCLUSION

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 3, won't fit into Xilinx Virtex V FPGA's internal RAMs. If one wants to use one of the RTOSs above on MicroBlaze running on Virtex V, an external memory should be placed on the board.

REFERENCES

[1]. Gokhan Ugurel, Dept. of Image Processing, Turkey, Cuneyt F. Bazlamacci, Dept. of Electrical and Electronics Eng., Turkey, "Context switching time and memory footprint comparison of Xilkernel and µC/OS-II on MicroBlaze" , 7th International Conference on, Electrical and Electronics Engineering (ELECO), 2011 (ISBN : 978-1-4673-0160-2).

[2]. Nan Xiang, Xiaodan Guan,Tong Zhou, Dept. of Electronic Engineering, "A design for 2-dimensional DCT transform kernel based on MicroBlaze soft core" , World Automation Congress (WAC), 2012 (ISBN : 978-1-4673-4497-5).

[3]. Poonm S. Isasare, Mahesh T. Kolte, University of Pune, Pune, India, "Fast Fourier Transform Implementation on FPGA Using Soft-Core Processor NIOS II" , IJETT, International Journal of Engineering Trends and Technology , *Volume 8 No 10* 2014 (ISSN : 2231-5381).

[4]. Emi Eto "Difference Based Partial Reconfiguration", XAPP290 (v2.0). Xilinx Inc., December 2007.

[5]. Xilinx Inc. "Processor IP Reference guide", v1.9, January 2004.

[6]. Xilinx Inc. "Vertex Series Configuration Architecture User Guide", XAPP151 (v1.7), October 2004.

[7]. Xilinx Inc. "PLB IPIF v2.02a", DS448, April 2005.

[8]. Xilinx Inc. "EDK 9.2 Microblaze Tutorial in Vertex", WT001 (v4.0), October 2007.

[9]. Xilinx Inc. "EDK Concepts, Tools, and Technique-A Hands on Guide to Effective Embedded System Design", May 2007.

[10]. Xilinx Inc. "Embedded System Tools Reference Manual", EDK 9.2i, September 2007.

[11]. Xilinx Inc. "Microblaze Processor Reference Guide Embedded Development Kit EDK 9.2i", UG081 (v9.0), January 2008.

[12]. Xilinx Inc. "ML505/ML506/ML507 Evaluation Platform User Guide", UG347 (v3.1.1), October 2009.

[13]. Xilinx Inc. "Vertex-5 Family Overview", DS100 (v5.0), February 2009.

[14]. Xilinx Inc. "Vertex 5 FPGA user guide", UG190 (v5.3), November 2009.