Implementation of Register Files in the Processor of Hard Real Time Systems for Context Switching

[1] **Prof. B Abdul Rahim,** [2] **Mr. S.Narayana Raju,** [3] **Mr. M M Venkateswara Rao**
[1] Head, Dept. Of ECE AITS, Rajampet, kadapa (Dist.), A.P, India
[2] M.Tech (Embedded Systems), NECT, Hyderabad, RR (dist.), A.P, India
[3] M.Tech (Embedded Systems), AITS, Rajampet, kadapa (Dist.), A.P, India

Abstract
Embedded Real Time applications use multi threading to share of real time application. The advantage of multi-threading include greater throughput, more efficient CPU use, Better system reliability improved performance on multiprocessor computer.
Real time systems like Flight control systems require very precise timing; multi threading itself becomes an overhead cost mainly due to context switching of the real-time operating system (RTOS). In this paper we propose a new approach to improve the overall performance of embedded systems by implementing register files into processor register bank itself. So that the system use multithreading by moving the context switching component of the real-time operating system (RTOS) to the processor hardware. This technique leads to savings of processor clock cycles used by context switching, By this approach the hard real time embedded systems performance can be improved

Keywords: Context Switching, Operating System, Memory, Simulation, Programming, Hard Real Time, Log Terminal

I. Introduction

In general, an operating system (OS) is responsible for managing the hardware resources of a computer and hosting applications that execute on the computer. A RTOS Is a Specialized type of operating system designed to execute applications with very precise timing and a high degree of reliability. They are intended for use with real time applications. Such applications include embedded systems (such as programmable thermostats and household appliance controllers), industrial robots, spacecrafts, industrial controllers, scientific research equipments, etc. RTOS can be divided into two categories, hard real-time and soft real-time systems. In a hard real-time or immediate real-time system, the completion of an operating after its deadline is considered useless, and this may cause a critical failure of the complete system and can lead to an accident (e.g. Engine Control Unit of a car, Computer Numeric Control Machines). Usually the kernel divides the application into logical pieces Commonly called threads and a kernel that coordinates their execution. A thread is an executing Instance of an Application and its context is the contents of the processor registers and program counter at any point of time. A scheduler, a part of the Real Time operating System's kernel, schedules threads execution based upon their priority. Context switching function can be described in slightly more detail as the Kernel performs different activities with regard to threads on the CPU as follows.

1. Suspend the progress of current running thread and store the processor's state for that thread in the memory.
2. Retrieve the context of the next thread, in the scheduler's ready list, from memory and restore it in the processor's registers.

Context Switching occur as a result of threads voluntarily relinquishing their allocated execution time or as a result of the scheduler making the context switch when a process has used ip its allocated time slice. A context switch can also occur as a result of a hardware interrupt, which is a signal from a hardware device to the kernel indicating that an event has occurred. Storing and restoring processor's registers to/from external memory (RAM) is a time consuming activity and may take 50 to 80 processors clock cycles depending upon context size and RTOS design. If the system needs to respond to an event in less than this time, the event response has to be implemented as an Interrupt Routine (ISR). On the other hand, if several events happen continuously, then the overall performance of the system may not be acceptable as most treads may not get a chance to execute. To improve responsiveness, the context switch time needs to be reduced. In general, there are two factors that effect the context switching cost. Direct cost due to moving the processor's registers to and from external memory or cache and indirect cost because of perturbation of cache, CPU, pipeline, etc. This presents difficulty in estimating the total cost of context switching cost[2], several algorithms have been developed and implemented to reduce the direct cost of context switching[3][4][5]. As discussed earlier, context registers need to be saved externamemory one at a time. A MIPS processor with 12 registers (9 temporary registers, stack pointer, global pointer and program counter), need to be saved, it requires at least 2 X 2 X 12=48 clock cycles to switch the context. Using our suggested approach, this context switch time would be reduced drastically to 4 processor's clock cycles independent of the number of context registers.

The approach presented in this paper is to save the context in newly created context register files. These register files are implemented in the processor hardware itself as part of the processor's register bank module. To test and achieve the required performance, software also needs to be modified to exploit the suggested hardware design.

II. HARDWARE IMPLEMENTATION

To prove the concept and measure the performance of our suggested approach, MIPS processor architecture was selected [6] and the suggested approach was implemented on top of it. The register bank module of the processor was modified by adding register files in the register bank to save context. The size of each register file is equal to context size. We have implemented 6 register files. The block diagram of the modified MIPS architecture as shown in figure 1.

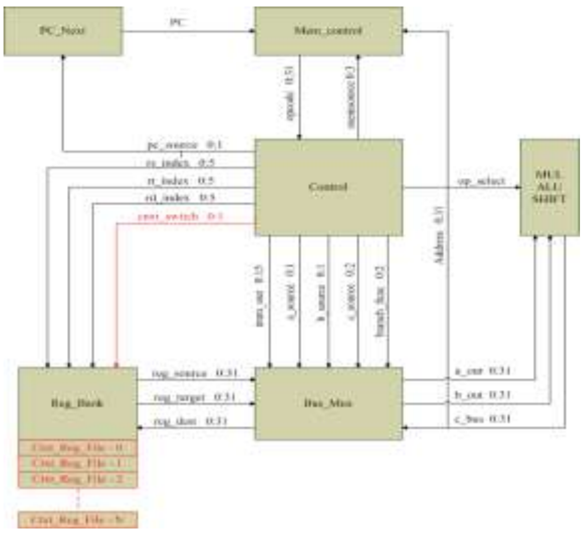


Figure 1. Modified MIPS processor architecture

To access these register files, two special context switch CPU instructions have been implemented: 'sext' and 'rext' to save and restore the processor's context respectively. The index will be stored in temporary register that to which register file the context will be stored and from which register file the context will be restore. Example register $4 contains the index of the register file.

III. SOFTWARE IMPLEMENTATION

To implement the software we have installed VMWARE for Linux operating system environment. VMWARE is an utility to enter from one operating system to another operating system.

The software implementation is divided into two parts. The first part deals with modification of GNU MIPS tool-chain by adding the 'scxt' and 'rcxt' instructions to the GNU-MIPS assembler [8].

The second part mainly deals with the implementation of a small co-operative operating system that executes the threads in round robin fashion using the newly implemented context switch instructions. The'sext' and 'rext' was developed using mips assembly language and the cooperating operating system was developed using C- language. Those two files executed in VMWARE linux environment. These object files added to the GNU MIPS assembler so that the GNU MIPS tool-chain gets modified. This operating system supports context switching using external RAM locations as well as internal register files.

Figure 2. Shows the co-operating operating system's tread structure. To achieve the fast context switch using internal register files, application has to set the 'FastCtxtSwitch' member of the task structure to 1 for that particular thread at the time of thread creation.

```
#include "plasma's"

#define CONTXT_SIZE 15
typedef void (*TaskFunc)(void)
typedef  struct task
{
Void (*Taskptr)();
…………..
…………..
……..
}

Void createTask (….)
  {
If (cnxt_type= =0)
{Threads[TaskID].State=0;
Threads[TaskID].Fastctxtswitch=1;
}else
{
Threads[TaskID].FastCtxtSwitch=0;
}
```

Figure 2. Task Structure

IV. SIMULATION RESULTS

Xilinx Spartan 3E 1600E the advanced development board was used for our test and experimentation [9]. Xilinx ISE 10.1 was used to design the hardware and the simulations were executed using the advanced simulation and debugging toolset, ModelSim [10]. Figure 3 shows the simulation wave form. Context registers are saved in the context switch register file-2 in 2 clock cycles.

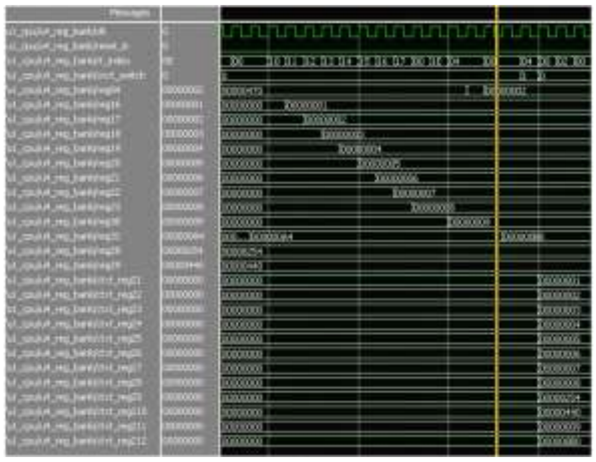


Figure 3. Simulation results for ‘scxt’ instruction

Similarly figure 4. Shows the simulation waveform of the ‘rext’ instruction. The context registers are also being restored from the context switch register file-2 in 2 clock cycles.

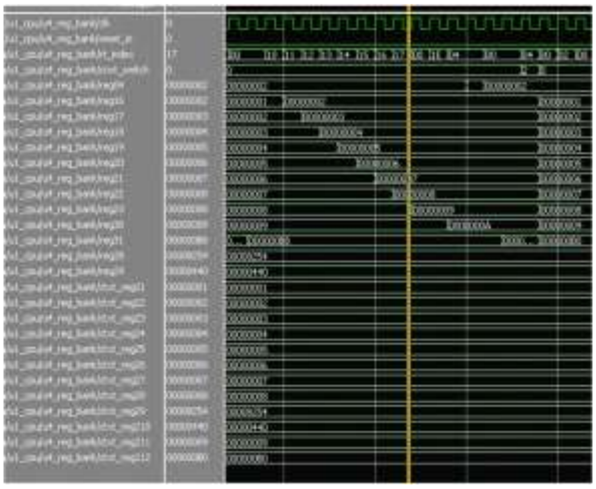


Figure 4. Simulation results for ‘rext’ instruction

V. PROGRAMMING VIA IMPACT

After successfully compiling an FPGA design using the Xilinx development software, the design can be downloaded using the iMPACT programming software and the USB cable. To begin programming, connect the USB cable to the starter kit board and apply power to the board. Then double-click configure Device (iMPACT) from within Project Navigator. To start programming the FPGA, right click the FPGA and select program. When the FPGA successfully programs, the iMPACT software indicates success, as shown in Figure 5.

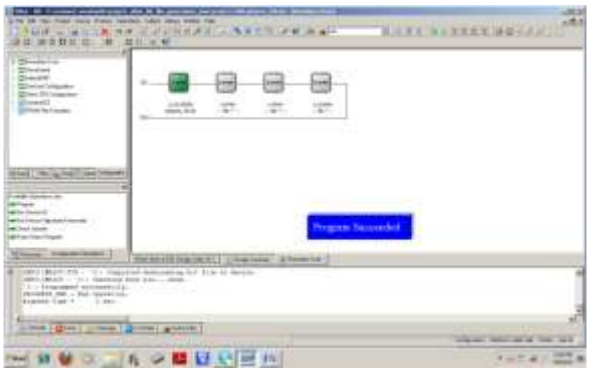


Figure 5. iMPACT Programming Succeeded

VI. TEST APPLICATION 1

This Application tests the successful operation of the proposed approach by switching four threads using internal register files. This test is used to ensure that data between threads is not corrupted and thread’s context switching is correct. The flow chart of this application is shown in figure 6.

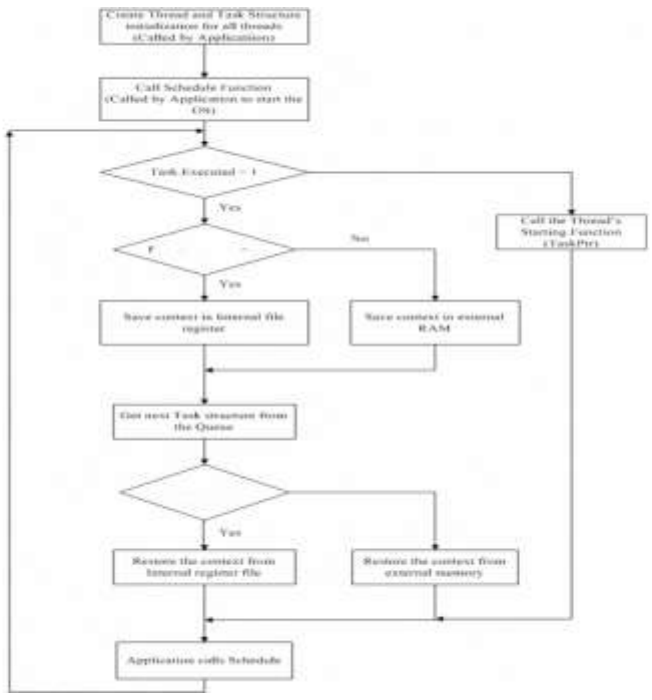
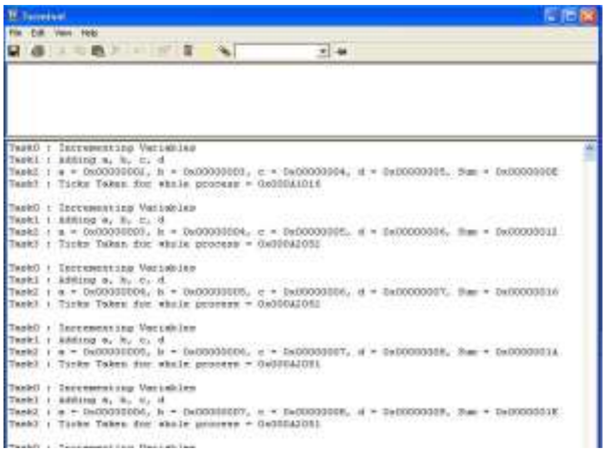

Create Thread and Task Structure initialization for all threads (Called by Application)
Call Schedule Function (Called by Application to start the OS)
Task Executed = 1
Yes
No
Yes
Call the Thread's Starting Function (TaskPtr)
Save context in internal file register
Save context in external RAM
Get next Task structure from the Queue
Yes
Restore the context from internal register file
Restore the context from external memory
Application calls Schedule

Figure 6:Flowchart for Test Application -1

There are four Tasks Executes in round robin fashion. TaskID=0, TaskID=1, TaskID=2, TaskID=3 executes one by one, finally the TaskID3 calculates and prints the Number of Clock cycles consumed to process one data sample. These Tasks will send the messages to the debug terminal port, the output log received on the debug terminal as shown in figure 7.

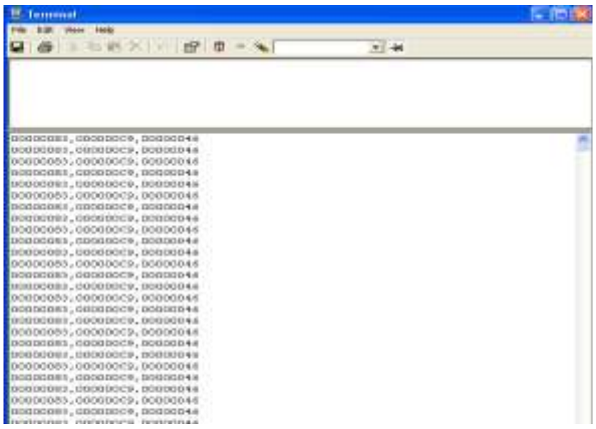


```
Task0 : Incrementing Variables
Task1 : Adding a, b, c, d
Task2 : a = 0x00000002, b = 0x00000003, c = 0x00000004, d = 0x00000005, Sum = 0x0000000E
Task3 : Ticks Taken for whole process = 0x000A1016

Task0 : Incrementing Variables
Task1 : Adding a, b, c, d
Task2 : a = 0x00000003, b = 0x00000004, c = 0x00000005, d = 0x00000006, Sum = 0x00000012
Task3 : Ticks Taken for whole process = 0x000A2052

Task0 : Incrementing Variables
Task1 : Adding a, b, c, d
Task2 : a = 0x00000004, b = 0x00000005, c = 0x00000006, d = 0x00000007, Sum = 0x00000016
Task3 : Ticks Taken for whole process = 0x000A2052

Task0 : Incrementing Variables
Task1 : Adding a, b, c, d
Task2 : a = 0x00000005, b = 0x00000006, c = 0x00000007, d = 0x00000008, Sum = 0x0000001A
Task3 : Ticks Taken for whole process = 0x000A2051

Task0 : Incrementing Variables
Task1 : Adding a, b, c, d
Task2 : a = 0x00000006, b = 0x00000007, c = 0x00000008, d = 0x00000009, Sum = 0x0000001E
Task3 : Ticks Taken for whole process = 0x000A2051
```

Figure 7: Serial Debug Log from Test Application -1

VII. Test Application-2

The Second application is designed to measure the performance improvement, in clock cycles. It creates four threads that executes in never-ending loops. First two threads does context switching using internal register files. Next two threads does context switching using external memory. Message send to the debug serial port include: Number of Clock cycles consumed by first two threads, Number of clock cycles consumed by the next two threads. Finally, the difference between these two. Figure 8 shows the output log for this test.



```
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
00000053,000000C9,00000046
```

Figure 8: Serial Debug Log from Test Appl-2

VIII. Conclusion

This paper Presents a new approach as an attempt to improve the hard RTOS system performance which helps in meeting the deadlines in an efficient Way. It boosts the system performance for such systems where lot of context switching is required in a small period of time. This approach can be easily extended to soft RTOS and regular operating systems to improve the overall system performance. In these systems, threads are created at run time and it is difficult to know the number of threads at the design time. Therefore, threads that are part of frequent context switching using internal register files using the specially designed scheduler algorithm. This paper also takes another step forward in moving the real-time operating system kernel to hardware and programming iMPACT using Xilinx software.

IX. References

[I] http://www.rtos.com/PDFs/ AnalyzingReal-TimeSystemBehavior.pdf

[2] Francis M. David, Jeffery C. Carlyle, Roy H. Campbell "Context Switch Overheads for Linux on ARM Platforms" San Diego, California Article No. : 3 Year of Publication: 2007 ISBN:978-I-59593-751-3

[3] Zhaohui Wu, Hong Li, Zhigang Gao, Jie Sun, Jiang Li An Improved Method of Task Context Switching in OSEK Operating System" Advanced Information Networking and Applications, 2006. AlNA 2006. Publication date: 18-20 April 2006 ISSN : 1550-445X

[4] Jeffrey S. Snyder, David B. Whalley, Theodore P. Baker "Fast Context Switches: Compiler and rchitectural support for Preemptive Scheduling" Microprocessors and Microsystems, pp.35-42, 1995. Vailable:citesser.ist.psu.edul33707.html

[5] Xiangrong Zhou, Peter Petrov "Rapid and low-cost context-switch through embedded processor customization for real-time and control applications" DAC 2006, July 24- 28 San Francisco, CA [6] http://www.opencores.org/projecLplasma

[7] MIPS Assembly Language Programmer's Guide, ASM - 0 I-DOC, PartNumber 02-0036-005 October, 1992

[8] http://ftp. gnu.org/gnuibinutilsl

[9] MicroBlaze Development Kit Spartan-3E 1600E Edition User Guidewww.xilinx.com

[10] http://model.com/contentlmodelsim-pe-simulation-and-debug