ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 12, December 2014

Application with MUCOS RTOS on embedded systems

Elumalai.R[1], A.R Purushotham Reddy [2], Pushpa.M.K[1], Jyothirmayi.M.[1], M.D.Nandeesh[1]

Department of Electronics & Instrumentation Engineering, MS Ramaiah of Institute of Technology, Bangalore[1]

FICE, Bangalore[2]

Abstract: Real-time operating system (RTOS) is a very useful tool for developing the application on embedded boards with least software development effort. Though number of RTOS products are available in the market, µC/OS-II is a freeware with minimum facility and more popular among the hobbyist, researchers and small embedded system developers. The µC/OS-II supports preemptive scheduling which is not efficient with respect to processor utilization. As a result, this may lead to missing deadline of the task assigned and hence may cause system failure. In this paper, a Rate Monotonic scheduling (RM), which is a better scheduling method when compared to preemptive technique, is implemented on µC/OS-II and its operation in terms of task execution and processor utilization are discussed. This paper presents the *RM Analysis* (RMA) on µC/OS-II with two different types of hardware: 1) Low end microcontroller i.e., 8051 based system and 2) High end system based on ARM9 is used. For the demonstration of *RM Analysis*, two tasks, namely, 1) Advanced encryption standard (AES) and 2) Text message display and graphic display are implemented. The software tools Keil IDE, SDCC compiler and Phillips Flash Magic are used for implementation of tasks on 8051 embedded development board. The ARM developers Suite v1.2 and DNW are used for implementation of tasks on ARM9 development board. In addition to the above said tasks, additional tasks like Real time clock interface, graphical LCD interface and UART interface for communication with computer are also implemented. The scaled-version of µC/OS-II with multiple tasks uses 4 kB of flash and 512 bytes of RAM in 8051 board. The entire MUCOS RTOS in ARM9 with multiple tasks uses 29.23 kB of flash and 596.36 kB of RAM. The results obtained indicate optimum utilization of processor with RMA scheduler for realizing low cost software for developing the application on embedded boards with least software development effort.

Keywords: RTOS, Keil IDE, ARM9, AES, µC/OS-II and Microcontroller.

I. INTRODUCTION

Real-time operating system (RTOS) is useful for developing the application on embedded boards with least software development effort. The human effort needed for implementing µC/OS-II is less compared to other RTOS's. Among the available RTOS's µC/OS-II is suitable for various controllers and processors, since it is low cost and easily available. The µC/OS-II supports preemptive scheduling and not efficient with respect to processor utilization. Processor utilization is a measure of CPU loading. Improving the CPU loading is difficult since embedded processor needs to complete the tasks within the deadlines.
The Rate Monotonic Analysis (RMA) helps to achieve optimum CPU loading for running real-time tasks with time deadlines. In RMA, most frequently used task gets highest priority. Optimum utilization of resources by meeting the deadlines is discussed in this paper.
A micro-kernel operating system (µC/OS-II) written by John J Lebrosse is a freeware for peaceful research purpose. The source code is written in C language which is compliant to ANSI C format. It has about 10,000 lines of source code with well-documented comments in the source code. It is a well-tested source code and has been ported to thousands of devices. It has a built-in preemptive scheduler. But, it does not support a rate-monotonic scheduler. In this paper, it is proposed to implement a rate-monotonic scheduler in µC/OS-II and will be discussed in this paper.
The microkernel approach is based on the idea of only placing essential core real-time operating system functions in the kernel, and other functionality is designed in modules that communicate through the kernel via minimal well defined interfaces. The microkernel approach results in easy reconfigurable systems without the need to rebuild the kernel. The µC/OS-II kernel is implemented completely in software and is a well-used real-time operating system for embedded systems. It is written as a monolithic kernel in the language of ANSI C with a minor part in assembler for context switching.
The paper is organized as follows. Section II describes Configurable parameters for µC/OS-II. Section III describes the implementation of scheduler on low-end and high-end hardware board. Section IV describes the analysis of results. Section V describes the conclusion of this work.
This paper presents implementation of RMA in µC/OS-II. For this, µC/OS-II is ported into Keil IDE [2] and RMA code is appended to it. ROM image file is ported into 8051 based microcontroller flash and tested for scheduling with RMA. MUCOS RTOS code is also ported in ARM 9 board and the performance parameters are discussed.

II. CONFIGURABLE PARAMETERS IN µC/OS-II

It is necessary to configure the parameters so that only the required features are built into the target system. The code size is reduced from 10,000 lines to 5,500 lines by suitable configuration of parameters. This is needed to reduce the memory requirements and to achieve optimum utilization of resources for the required functions in 8051 hardware board.

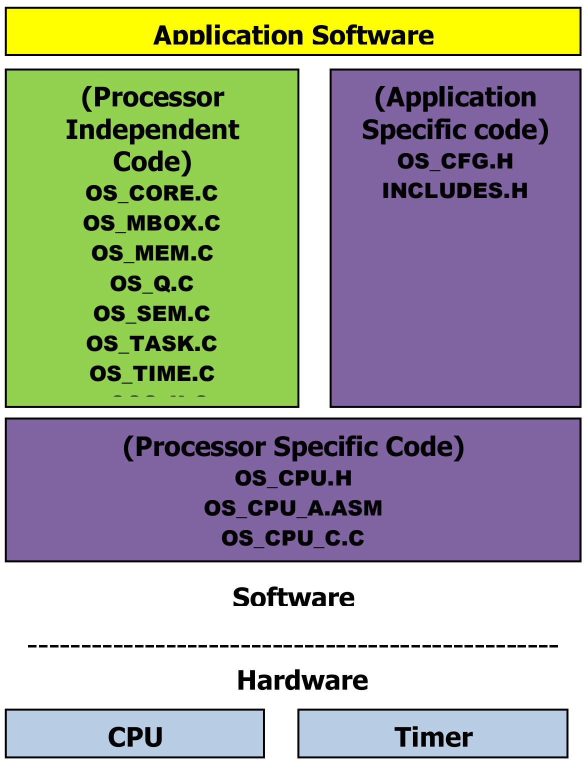

Fig. 1 µC/OS-II architecture

Figure 1 shows the architecture of µC/OS-II. The files of µC/OS-II is shown in figure 1. The application software is the code written by the user in the main function to perform several tasks. Processor independent codes are the files which can be used in any hardware without the need to configure it. Processor-specific and Application-specific code can be modified to suit the application requirements. The parameters that are configured in µC/OS-II are in file os_cfg_r and os_cpu_asm. The parameters that are configured in os_cfg_r with its explanation are given below

```
#define OS_TICK_STEP_EN            1
/* Set to 1/0 to Enable/Disable tick stepping feature
respectively for uC/OS-View */
#define OS_MAX_TASKS               3     /* Max.
number of tasks in your application, MUST be >= 2  */
#define OS_LOWEST_PRIO             5   /* Defines
the lowest priority that can be assigned ...   */
#define OS_TMR_CFG_TICKS_PER_SEC   10       /*
Rate at which timer management task runs (Hz)  */
```

The other features like semaphores, timers, queues and memory management are not needed. Hence, they are set to ‘0’ to disable these features.

The size of the OSStack and timer delay is configured in file os_cpu_asm. The routines are written in assembly language. The pseudo source code of main() function is given below.

```
void main(void)
{
  OSInit();/*Initialize OSStack and memory blocks */
TargetInit(); /*  Initialize the target hardware */
OSTaskCreate(Task0,(void)0,&
Task0Stack[MaxStkSize-1],0);/*Create Task0 with
priority 0*/
OSTaskCreate(Task1,(void
*)0,&Task1Stack[MaxStkSize-1],1);/*Create Task1 with
priority1*/
OSTaskCreate(Task2,(void
*)0,&Task1Stack[MaxStkSize-1],2);/*Create Task2 with
priority2*/
OSStart();// start muti-tasking
}/*end of main*/
```

The above code consists of OS initialization, target initialization, Creation of tasks and start multi-tasking functions. The priorities are assigned to tasks so that a task with a lowest priority is executed most frequently. This is the scheduling approach which is known as rate-monotonic scheduler approach.

III. IMPLEMENTATION OF RATE-MONOTONIC SCHEDULER ON A HARDWARE BOARD

A rate-monotonic scheduler is designed and implemented with a low-end and an high-end system. The microcontroller 8051and ARM 9 controller used in this work will be discussed in this paper.

The hardware implementation has been developed in Microcontroller (8051) Embedded Development Kit which consists of a 4 x 4 keypad, 2 x 16 LCD, 8 LEDs and an RS-232 port for serial communication. An external crystal of clock frequency 11.0592 MHz is used in the board. The board details are shown in the figure 2.

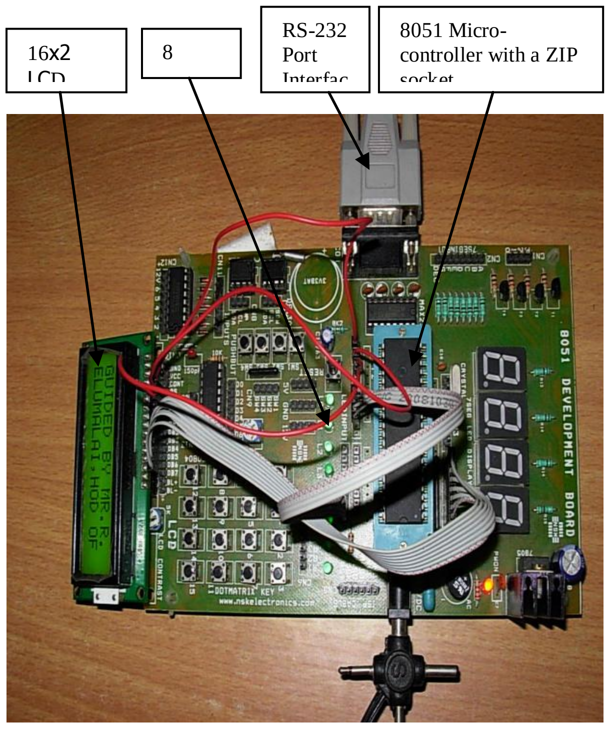

Fig. 2 8051 hardware board used for testing rate-monotonic schedule

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 12, December 2014

The MUCOS RTOS is also implemented in ARM 9 hardware board which consists of ARM 9 controller (S3C2440), 3.5" TFT LCD, USB port, RS-232 port, Ethernet port, JTAG port, 4 user-defined interrupt switches and NAND/NOR flash interface. The system clock frequency used is 400 MHz. The board details are shown in figure 3.
The software implementation has been developed in an IDE (Integrated Development Environment) with keil compiler and Phillips Flash Magic tool [3] for 8051 hardware board. SDCC compiler is used to implement the encryption/decryption task in 8051 board. The software implementation for ARM 9 hardware board is carried out using ARM Developer Suite (ADS) v1.2, DNW and HyperTerminal tool. The software tools like Keil IDE, SDCC, ADS are used to compile and build the project. The other tools like Phillips Flash Magic, DNW and Hyper Terminal are used to interact with the target hardware board.
The source code for the tasks described in TABLE I is written in C language and compiled with Keil compiler. The hex file generated by the compiler is used for programming the micro-controller (P89V51RD2) with Phillips Flash Magic tool. The user-defined application is written in file main.c. It consists of user-defined tasks to be performed by assigning the priorities in an infinite loop. The memory requirements projected after compilation of code in 8051 and ARM 9 are discussed in Table III and Table V respectively.

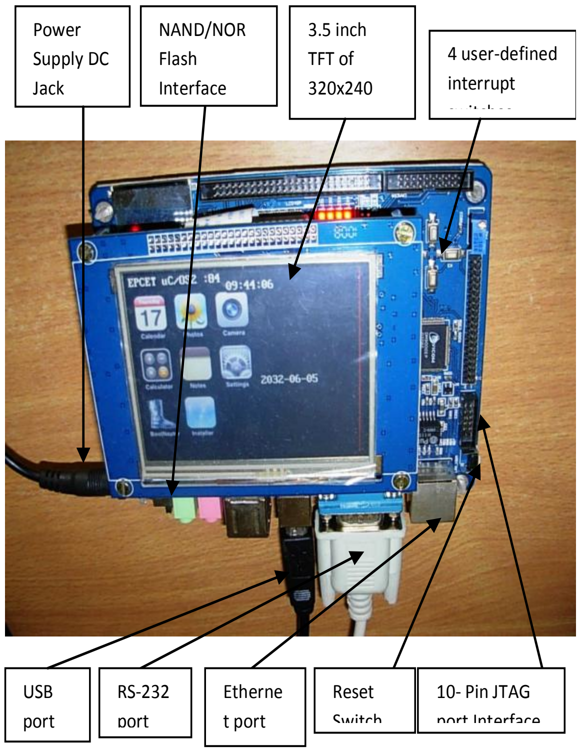

Fig. 3: ARM 9 hardware board used for testing rate-monotonic scheduler

The CPU utilization or time-loading factor (U) is a measure of the percentage of non idle processing. A system is said to be time-overloaded if U > 100%. Systems that are too highly utilized are undesirable because changes or additions cannot be made to the system without risk of time-overloading. The target for U as 69% in rate-monotonic systems gives a very useful result in the theory of real-time systems.

The theoretical results observed are close to that of experimental results. The results are computed for four different tasks and are shown in Table I. The value of U obtained is 70.18% for the four tasks shown in Table I.
Theoretical value of CPU utilization = **75.68%**
Obtained value of CPU utilization = **70.18%**

TABLE I:
EXECUTION TIME FOR VARIOUS TASKS IN 8051

Task number	**Task name**	**ei (execution time of a task)**	**pi (execution period of a task)**
1	LCD_display0	1 s	8 s
2	LED_display1	2.13 s	8s
3	LED_display2	1.42 s	8 s
4	LED_display3	1.065 s	8 s

TABLE II:
AES RESULTS IN 8051 WITH SDCC TOOL

Task name	**Memory (Bytes)**	**Time (milliseconds)**
Encryption with 10 rounds	358	85
Encryption with 14 rounds	395	100
Decryption with 10 rounds	476	105

LCD_display: The data is displayed in the LCD by initializing the values in the command register. The function is executed within 8s.
LED_display1: LED is turned ON/OFF to indicate completion of a task. To perform this task, data has to be written to o/p port of microcontroller. The function is executed within 8 s.
LED_display2 and LED_display3: It is similar to the LED_display1 task.

AES results are discussed in Table II and they are implemented independently with SDCC tool without any usage of RTOS. The memory requirement of the above tasks in 8051 controller is shown in Table III below. The results obtained in ARM 9 for multiple tasks are discussed in Table IV.

TABLE III:
MEMORY REQUIREMENT OF 8051CONTROLLER

Serial no:-	**Types of Memory**	**Memory size (Bytes)**
1	Code memory	4897 bytes
2	Data memory	410 bytes

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 12, December 2014

TABLE IV:
EXECUTION TIME FOR VARIOUS TASKS IN ARM 9

Serial no:-	**Types of Memory**	**Memory size (kB)**
1	Code memory	29.23 kB
2	Data memory	596.36 kB

Theoretical value of CPU utilization = **75.68%**
Obtained value of CPU utilization = **0.17775 %**
The Main Task in turn creates other tasks as Task 0, Task 1 and Task 2. It is used to initialize Graphical LCD and UART interface. Main Task and Task 0 is used to display messages in the hyper terminal of monitor. Task 1 is used to display a 2-digit decimal counter in the LCD. Task2 is used to display the date and time of a Real-time clock (RTC).

TABLE V:
MEMORY REQUIREMENT OF ARM 9 CONTROLLER

Task number	**Task name**	**ei (execution time of a task)**	**pi (execution period of a task)**
1	Main Task	16.66 ms	30s
2	Task 0	16.66 ms	30s
3	Task 1	3.33 ms	30s
4	Task 2	16.66 ms	30s

Comparison of results
The results discussed above are compared with respect to CPU utilization and are discussed in Table VI below.

TABLE VI:
COMPARISION OF RESULTS WITH RESPECT TO OPTIMUM CPU UTILIZATION

Number of Tasks	**Theoretical value**	**Practical value in 8051 controller**	**Practical value in ARM 9 controller**
4	75.68%	70.18%	0.17775%

The results indicate that optimum CPU utilization is not only dependent on the scheduling technique employed in an embedded system. But, it is also dependent on the processing power of controller and the complexity of the application code. A qualitative analysis of the results obtained with respect to 8051 controller and ARM 9 controller indicate that CPU utilization in ARM 9 controller is 395 times less than that of 8051 controller. It is due to the following major factors discussed below.

- 7-stage pipelining
- RISC architecture
- 32-bit controller
- Higher clock speed
- Cache memory

IV.ANALYSIS OF RESULTS

The execution time for various tasks is shown in Table I and table IV. The system can be operated with 69% CPU utilization but in this work we could operate up to 70.18% in 8051 hardware board. This is due to the fact that the interfaces in the board are limited. Also, there are other constraints like limited ON-chip memory, limited number of ports and lower processing power. If the tasks like displaying the data on LCD, indicating the status of task by turning ON/OFF LEDs, scanning and detecting the key pressed in keypad has to be done continuously for an infinite duration of time. This causes a significant delay in the RTOS. Most of these constraints are reduced to a larger extent in a high-end system like ARM9. The CPU utilization in a high-end system is less as compared to a low-end system and it is discussed in Table VI.

IV. CONCLUSIONS

The µC/OS-II is ported onto Keil IDE and RMA analysis result is discussed in this paper. ROM image file is ported onto 8051 based microcontroller flash as well as ARM9 controller and tested for scheduling with RMA. The entire µC/OS-II with multiple tasks uses 4 kB of flash and 512 bytes of RAM. The entire MUCOS RTOS in ARM9 with multiple tasks uses 29.23 kB of flash and 596.36 kB of RAM. The results have indicated optimum utilization of processor with RMA scheduler for realizing low cost software and hardware for developing embedded system. The overall CPU utilization was 70.18% with RMA approach in 8051 whereas in ARM9 controller it is 0.17775%. The result obtained indicates optimum utilization of processor with RMA scheduler for realizing low cost software for developing the application on embedded boards with least software development effort.

REFERENCES

[1] F. Engel, G. Heiser, I. KuZ, S. M. Petters and S. Ruocco, "Operating Systems on SoCs: A Good Idea?," in ERTSI in conjunction with 25th IEEE RTSS04, Lisbon, Portugal, December 2004.
[2] Furunas, J. "Benchmarking of a Real-Time System that utilises a booster." In International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA200), June, 2000.
[3] H. Hartig, M. Hohmuth, J. Liedkte, S. Schonberg and J. Wolter, "The performance of µ-Kernel-Based Systems", in proceedings of the 16th ACM symposium on Operating Systems Principles, p 66-77, Saint Malo, France, 1997.
[4] L. Johansson and T. Samuelsson, "Integration of an Ultra-fast Real-Time Accelerator in the Real-Time Operating System µC/OS-II", Master Thesis report, Malardalen University, Vasteras, Sweden, October 2004.
[5] P. Kohout, B. Ganesh and B. Jacob, "Hardware Support for Real-Time Operating Systems", in Conference on Hardware/Software codesign and system synthesis of contents, p.45-51, Newport Beach, USA, 2003.
[6] P. Kuacharoen, M. A. Shalan and V. J. Mooney III, "A Configurable Hardware Scheduler for Real-Time Systems," in Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA´03), p 96-101, Las Vegas, USA, June 2003.
[7] J. Liedkte, "Toward Real Microkernels," in Communications of the ACM, vol. 39, No 9, September 1996.
[8] L. Lindh, and F. Stanischewski, "FASTCHART – A Fast Time Deterministic CPU and Hardware Based Real-Time-Kernel." In IEEE, Euromicro workshop on Real-Time Systems, June 1991.
[9] L. Lindh, T. Klevin, and J. Furunas, "Scalable Architecture for Real- Time Applications – SARA". Swedish National Real-Time Conference SNART99 Linkoping, Sweden, August, 1999.
[10] T. Nakano, Y. Komatsudaira, A. Shiomi and M. Imai. VLSI implementation of a Real-time Operating System. Proc. of ASPDAC '97, pp. 679-680, January, 1997.