International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869, Volume-1, Issue-2, April 2013

Design and Implementation of µc/Os II Based Embedded System Using ARM Controller

Indersain, Neetika Sharma, Dushyant Singh

***Abstract*— Paper describes an embedded System based on the µC / OS II operating system using ARM7. It deals with the porting of Micro C/OS-II kernel in ARM powered microcontroller for the implementation of multitasking and time scheduling. Here the real time kernel is the software that manage the time of a micro controller to ensure that all time critical events are processed as efficiently as possible. Different type of interface modules of ARM7 microcontroller like UART, ADC, DAC, KAYPAD, LCD, USB are used and data acquired from these interfaces is tested using µC/OS-II based real time operating system. It mainly emphasizes on the porting o f µC/OS-II.**

***Index Terms*— Embedded System Design, ARM7, Real Time OS, µC/OS II..**

I. INTRODUCTION

In high end applications, sometimes devices may malfunction or totally fail due to long duration of usage or any technical problem which give fatal result. An embedded monitoring system is always necessary for continuously collecting data from onsite and later analyzing that and eventually taking proper measures to solve the problem. The system which is used today use non real time operating systems based on mono-task mechanism that hardly satisfies the current requirements. This paper is focused on porting of µC/OS II in ARM7 controller that performs multitasking and time scheduling. The µC / OS II features porting to ARM7 are discussed. Finally it provides an overview for design of embedded monitoring system using µC/OS II as application software that helps in building the total application.

II. ARM'S CORE FAMILIES

Advanced RISC Machine. First RISC microprocessor for commercial use and a market-leader for low-power and cost-sensitive embedded applications. It has the features like architectural simplicity which allows very small implementations result in very low power consumption. The ARM7TDMI has a core based on the fourth version of the ARM architecture. This implementation can use a three stage pipeline a standard fetch-decode-execute organization. The core is very successful mainly because of the extremely small but high performance processor and having more than 70000 transistors with low power consumption.

ARM cores use a 32-bit, Load-Store RISC architecture. It means that the core cannot directly manipulate the memory of system. All data manipulation must be done by loading registers with information located in memory, performing the data operation and then storing the value back to memory. There are total 37 registers in the processor. However, this number is split among 7 different processor modes. These seven processor modes are used to run user tasks, an operating system, and to efficiently handle exceptions such as interrupts. Some of registers in the mode are reserved for specific use by the processor core, while most are available for general uses. The reserved registers that are used by the processor core for specific functions are r13 is commonly used for stack pointer (SP), r14 for link register (LR), r15 for program counter (PC), the current program status register (CPSR), and the saved program status register (SPSR).The SPSR and the CPSR contain the status and control bits specific to the properties the processor core is operating under. These properties demarcate the operating mode, ALU status flags, interrupt disable or enable flags and whether the core is operating in 32-bit ARM or 16-bit Thumb state.

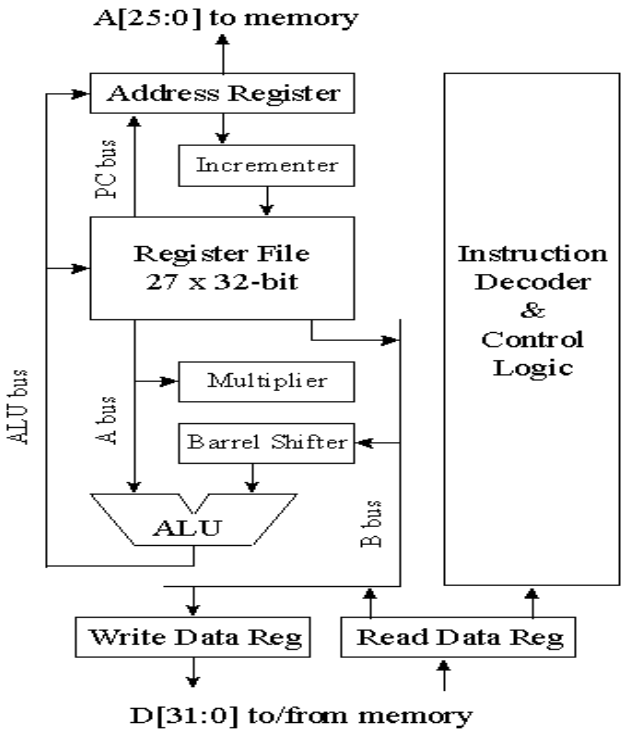

Figure 1: ARM Architecture

II µC/OS II

µC/OS II (pronounced "Micro C O S 2") stands for

Manuscript received April 11, 2013.
Indersain, Asst. Prof., Dept. of E&C, JNIT/ Jaipur/ Raj./ India, mobile no. +91-9001848536.
Neetika Sharma, M-tech. Scholar, KIET/ Jaipur/ Rajasthan/ India.
Dushyant Singh, M-tech. Scholar, SKIT/ Jaipur/ Rajasthan/ India.

Microcontroller operating system version 2 and can be termed as µC/OS-II or uC/OS-II). This is a very small real-time kernel with memory footprint is about 20KB for a kernel with full functions and source code is about 5400 lines, mostly in ANSI C. Its source is open but not free for commercial usages. µC/OS-II is upward compatible but provides many improvements than previous version, such as the addition of a fixed-sized memory manager, user definable callouts on task creation, task deletion, task switch, and system tic, TCB extensions support, stack checking, and much more.

A. µC/OS II using ARM

This real time kernel is a highly portable, ROMable, scalable, pre-emptive real-time, multitasking kernel (RTOS) for microprocessors and microcontrollers. This can manage up to 250 application tasks. This RTOS runs on a large number of processor architectures. The large number of ports shows that µC/OS-II is truly very portable and thus will most likely be ported to new processors as they become available.

B. Choosing µC/OS-II

These are the features of µC/OS-II make it convenient to port.

1) ROMable

It is designed for embedded application. This means that if you have the proper tool chain (i.e. C compiler, assembler and linker/locater), you can embed Micro C/OS-II as part of a product.

2) Portable

µC/OS-II is written in highly portable ANSI C, with target specific code written in assembly language. Assembly language is kept to a minimum to take µC/OS-II easy to port to other processors. µC/OS can be ported to a large number of micro processors as long as the microprocessors provides a stack pointer and the CPU register can be pushed onto and popped from the stack. Also, the C compiler should provided either in-line assembly language extension that allows you to enable and disable interrupt from C. µC / OS-II can be run on most 8, 16, 32 bit or even 64 bit microcontrollers or microprocessors and DSPs.

3) Scalable

µC/OS-II is designed such a way so that only the services needed in the application can used. This means that a product can be use just a few µC / OS-II services. Another product may require the full set of features. This allows to reduce the quantum of memory (both RAM and ROM) needed by µC/OS-II on a per product basis. Scalability is accomplished with the use of conditional complication.

4) Pre-emptive

µC/OS-II always runs the highest priority task that is ready because this is a fully pre-emptive real time kernel.

5) Multitasking

Multitasking is the process of scheduling and switching the CPU between several tasks. µC / OS-II can manage up to 64 tasks. Each task has a unique priority assigned to it, which mean that µC / OS- II cannot do round robin. There are thus 64 priority levels.

6) Deterministic

Execution time of all µC / OS-II functions and services are deterministic. This means that one can always know how much time µC / OS-II will take to execute a function or a service. Furthermore except for one service, execution time all C / OS-II services do not depend on the number of tasks running in the application.

C. µC/OS-II Starting

In any application µC / OS-II is started as shown in the figure2. Initially the hardware and software are initialized. The hardware is the ARM core and software is the µC / OS-II. The resources are allocated for the tasks defined in the application the scheduler is started then it schedules the tasks in pre-emptive manner. All these are carried out using specified functions defined in µC / OS-II.

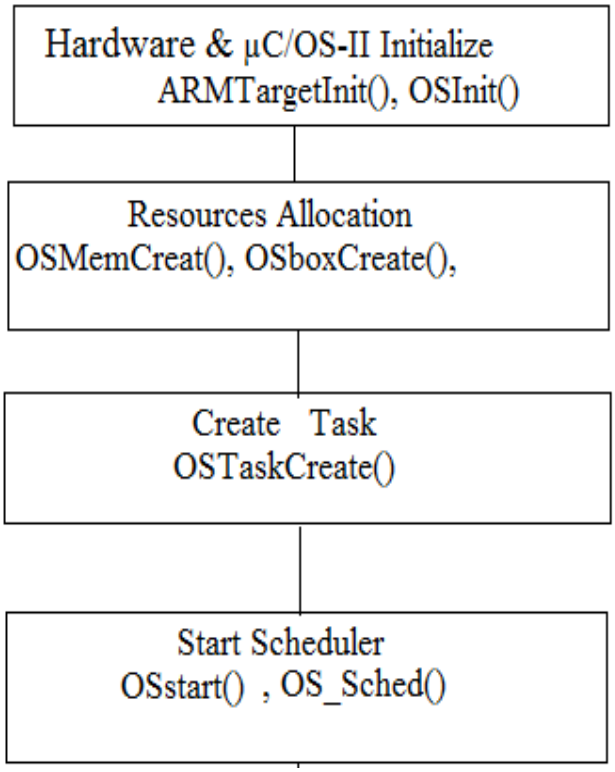

Figure 2: Starting µC/OS-II

D. Initializing µC/OS-II

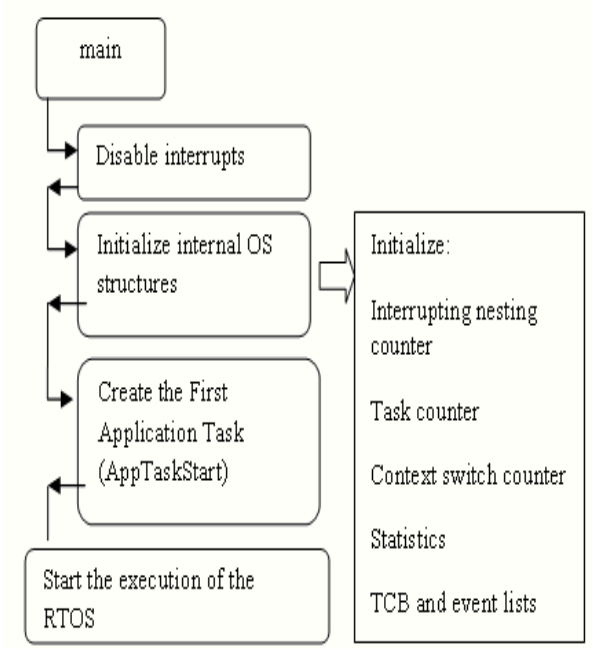

Figure 3: Initializing µC/OS-II

µC / OS-II can be initialized as shown in the figure 3. The detailed steps are shown in the figure. Below shows the sample program for the steps shown in the figure.

```
Void main (void)
  {
  /* User initialization*/
        OSInit();     /*   kernel initialization  */

  /* Start OS

  OSStart(); /*  start multitasking    */
  }
```

A. Task Creation

To make it ready for multitasking, the kernel needs to have information about the task its starting address**,** top-of-stack (TOS)**,** priority**,** arguments passed to the task**,** some other information about the task.
You can create a task by calling a service provider by µC/OS-II.

```
        OStaskCreate (void (*task) (void *parg),

        Void *parg;  // Address of Task
        OS_STK *pstk; // Pointer to task's Top
                                  //Of Task
INT8U prio);  // Priority of task (0--64)
```

You create task before you start multitasking (at initialization time) or during run time.

A. Implementation through µC/OS-II Task Creation

In embedded systems, a board support package (BSP) is implementations specific support code for a given (device motherboard) board. It is commonly built with a boot loader that contains the minimal device support to load the operating system and device drivers for all the devices on the board. It can provide a root file system, a tool chain for making programs to run on the embedded system (which would be part of the architecture support package), and configurations for the devices (while running).

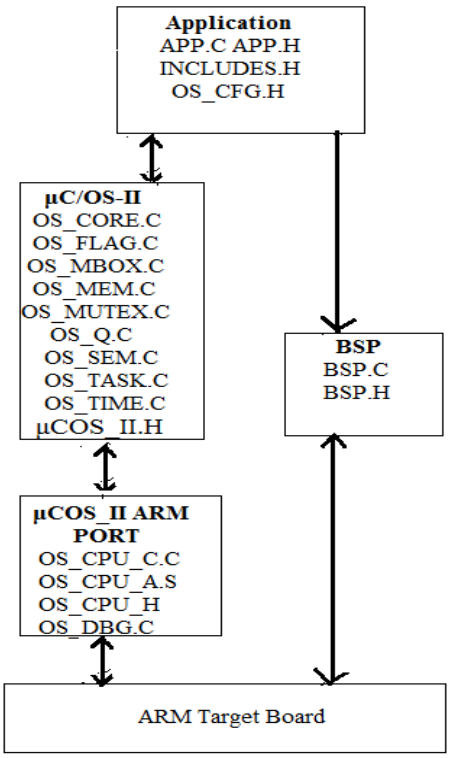

Figure 3: The architecture of hardware and software when using µC/OS II

III. SYSTEM ARCHITECTURE

The heart of the system is a real-time kernel that uses pre-emptive scheduling to achieve multitasking on hardware platform. In This research paper it deals with the implementation of hardware and software.

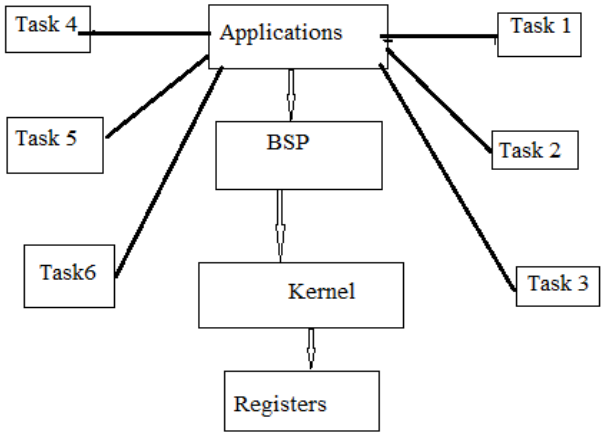

Figure 4: Block diagram

In Micro C / OS-II maximum number of tasks is 64. In the above shown figure4 the application have six tasks. Depending on the required application the number of tasks may vary. To perform a sample experiment to understand the

porting of µC / OS-II we can perform simple tasks like Temperature sensor (i.e. ADC), Graphical LCD (i.e. degree to graphical Fahrenheit), UART (i.e. digital data displaying), LED toggle (i.e. 8-bit data flow control) buzzer (i.e. alarm device). The ARM runs the Real time operating system to collect information from the external world. Here RTOS is used to achieve real time data acquisitions. µC / OS-II kernel is ported in ARM powered microcontroller for the implementation of multitasking and time scheduling. Keil IDE is used for implementation.

Keil IDE is a windows operating system software program that runs on a PC to develop applications for ARM microcontroller. It is also called Integrated Development Environment or IDE because it provides a single integrated environment to develop code for embedded microcontroller.

IV. CONCLUSION

In this paper the porting of µC / OS-II in ARM 7 is presented. It mainly focuses on designing an embedded monitoring system using ARM 7 and µC / OS-II RTOS. The steps involved in porting the RTOS and final implementation details are provided. This paper provides a detailed overview for developing an embedded monitoring system using ARM and µC / OS-II.

REFERENCES

[1] Liu Zhongyuan, Cui Lili, Ding Hong, "Design of Monitors Based on ARM7 and Micro C/OS-II", College of Computer and Information, Shanghai Second Polytechnic University, Shanghai, China, IEEE 2010.

[2] Tianmiao Wang The Design And Development of Embedded System Based on ARM Micro System and IlC/OS-II Real-Time Operating System Tsinghua University Press.

[3] Jean J Labrosse, MicroC/OS-II the Real-Time Kernel, Second Edition Beijing University of Aeronautics and Astronautics Press

[4] Design of µC/ Os IIRTOS Based Scalable Cost Effective Monitoring System Using Arm Powered Microcontroller, M. Venkateswara Rao, Dept. of ECM, K L University, A.P, India