# Design of µC/ Os II RTOS Based Scalable Cost Effective Monitoring System Using Arm Powered Microcontroller

**Ch. S. L. Prasanna, M. Venkateswara Rao**

Dept. of ECM
K L University, A.P, India

***Abstract-*** This paper describes an embedded monitoring system based on 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 a real time kernel is the software that manages the time of a micro controller to ensure that all time critical events are processed as efficiently as possible. Different interface modules of ARM7 microcontroller like UART, ADC, LCD 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 of µC/OS-II.

***Index Terms***- embedded system, µC/OS-II, arm 7, RTOS

## 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 results. An embedded monitoring system is necessary for continuously collecting data from onsite and later analyzing that and eventually taking proper measures to solve the problem. The systems that are in use today use non real time operating systems based on mono-task mechanism that hardly satisfies the current requirements. This paper will focus on porting of µC/OS II in ARM7 controller that performs multitasking and time scheduling. The µC/OS II features and its 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. MICRO C/OS II

µC/OS II (pronounced "Micro C O S 2") stands for Micro-Controller Operating System Version 2 and can be termed as µC/OS-II or uC/OS-II),. It is a very small real-time kernel with memory footprint is about 20KB for a fully functional kernel and source code is about 5,500 lines, mostly in ANSI C. It's source is open but not free for commercial usages. µC/OS-II is upward compatible with µC/OS V1.11 but provides many improvements, such as the addition of a fixed-sized memory manager; user-definable callouts on task creation, task deletion, task switch, and system tick; TCB extensions support; stack checking; and much more.

### *A. µC/OS II using ARM*

µC/OS-II, The Real-Time Kernel is a highly portable, ROMable, scalable, preemptive real-time, multitasking kernel (RTOS) for microprocessors and microcontrollers. µC/OS-II can manage up to 250 application tasks. µC/OS-II runs on a large number of processor architectures and ports. The vast number of ports should convince that µC/OS-II is truly very portable and thus will most likely be ported to new processors as they become available. µC/OS-II can be scaled to only contain the features you need for your application and thus provide a small footprint. Depending on the processor, on an ARM (Thumb mode) µC/OS-II can be reduced to as little as 6K bytes of code space and 500 bytes of data space (excluding stacks). The execution time for most of the services provided by µC/OS-II is both constant and deterministic. This means that the execution times do not depend on the number of tasks running in the application.

### *B. Choosing µC/OS II*

µC/OS II is chosen for the following features.

1. Portable

Most of µC/OS-II is written in highly portable ANSI C, with target microprocessor specific code written in assembly language. Assembly language is kept to a minimum to take µC/OS-II easy to port to other processors. Like Micro C/OS, Micro C/OS-II can be ported to a large number of microprocessors 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 provide either in-line assembly or language extension that allows you to enable and disable interrupt from C. µC/OS-II can run on most 8-, 16-,32 or even 64- bit microprocessors or microcontrollers and DSPs.

2. ROMable

µC/OS-II was 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**.**

3. Scalable

µC/OS-II is designed such a way so that only the services needed in the application can be used. This means that a product can use just a few µC/OS-II services. Another product may require the full set of features. This allows to reduce the amount 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. Preemptive

µC/OS-II is a fully preemptive real time kernel. This means that Micro C/OS-II always runs the highest priority task that is ready.

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.

7. Robust and Reliable

µC/OS-II is based on µC/OS which has been used in hundreds of commercial applications. µC/OS-II uses the same core and most of the same functions as µC/OS yet offers more features.

*C. Starting µC/OS-II*

In any application µC/OS-II is started as shown in the figure 1. 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 preemptive manner. All these are carried out using specified functions defined in µC/OS-II.

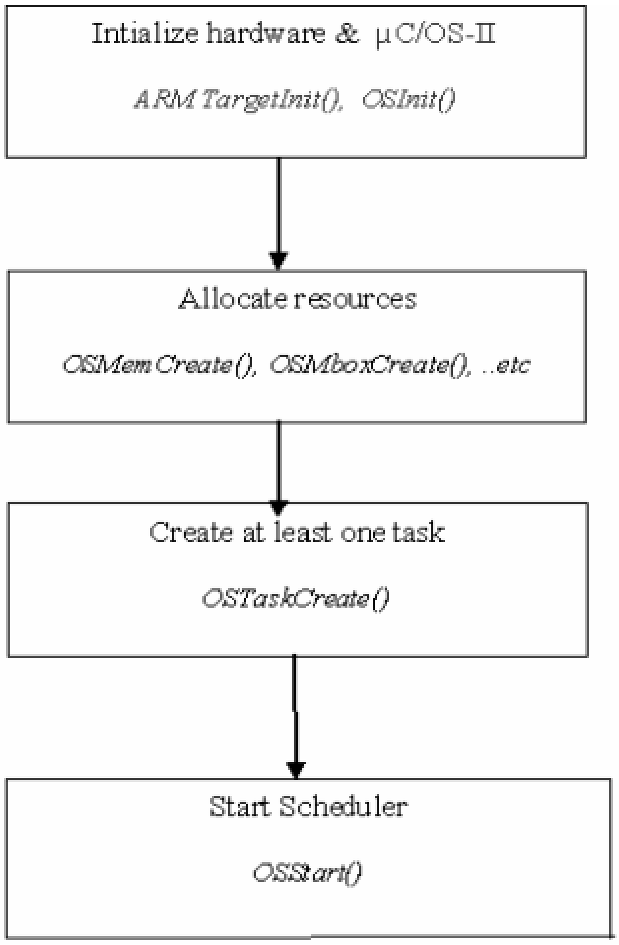
Intialize hardware & µC/OS-II
ARM TargetInit(), OSInit()
Allocate resources
OSMemCreate(), OSMboxCreate(), ..etc
Create at least one task
OSTaskCreate()
Start Scheduler
OSStart()

Figure 1: Starting µC/OS-II

*D. Intializing µC/OS-II*

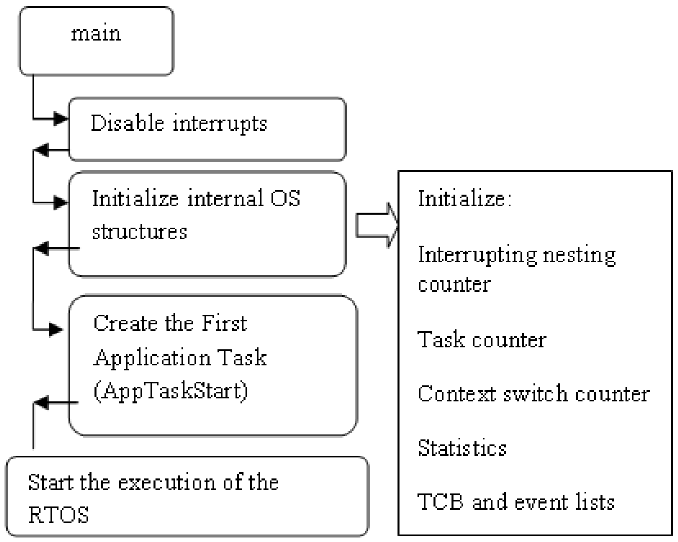
main
Disable interrupts
Initialize internal OS structures
Create the First Application Task (AppTaskStart)
Start the execution of the RTOS
Initialize:
Interrupting nesting counter
Task counter
Context switch counter
Statistics
TCB and event lists

Figure 2: Initializing µC/OS-II

µC/OS-II can be initialized as shown in the figure 2. The detailed steps are shown in the figure. Below shows the sample program for the steps shown in the figure.

```
void main (main)
 {
       /*    user initialization            */
         OSInit();     /*   kernel initialization  */
      /*   Install interrupt vectors       */
     /*    Create at least 1 task (start task)       */
    /*     Additional User code          */
        OSStart();  /*  start multitasking    */
             }
```

*E. 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, other information about the task.

You create a taskby calling a service provider by µC/OS-II:
OStaskCreate(void (*task) (void *parg),

| | |
|---|---|
| void *parg, | Address of the task |
| OS_STK *pstk, | Pointer to task's Top –Of-Task' |
| INT8U prio); | Priority of task (0--64) |

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

*F. Implementation through µC/OS-II*

In embedded systems, a board support package (BSP) is implementation specific support code for a given (device motherboard) board that conforms to a given operating system. 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. Some suppliers also 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). A board support package

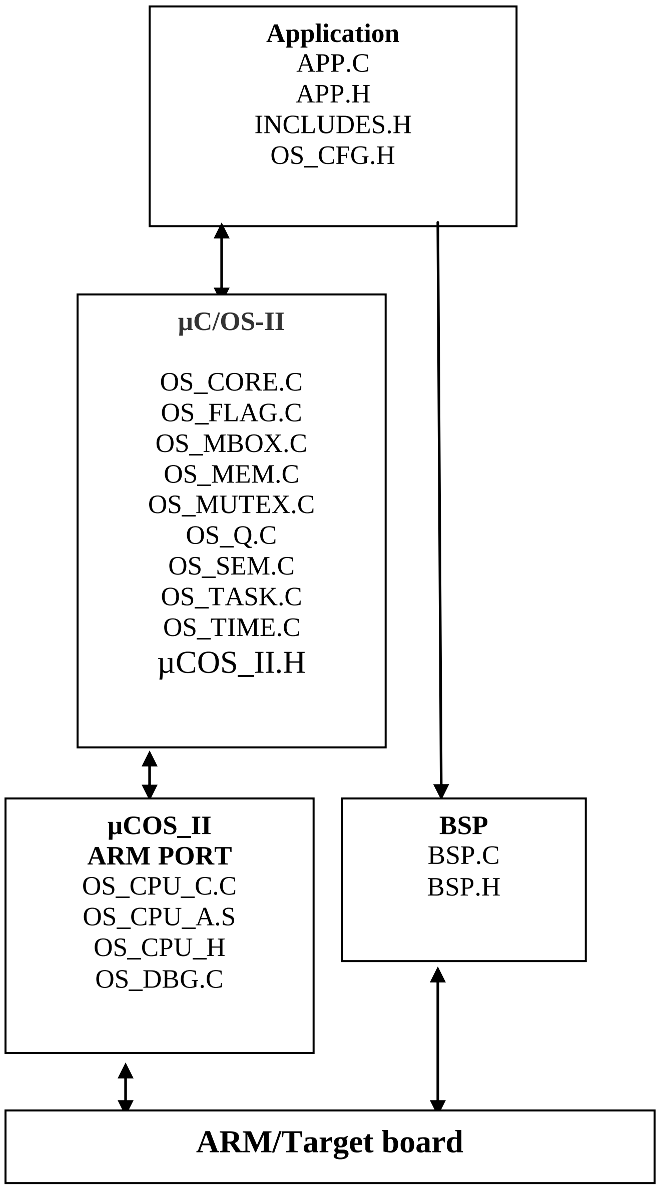

Figure 3: The architecture of hardware and software when using μCOS_II

## III. SYSTEM ARCHITECTURE

The heart of the system is a real-time kernel that uses preemptive scheduling to achieve multitasking on hardware platform. The previous sections dealt with μCOS_II porting to the application desired. This section deals with the implementation of hardware and software.

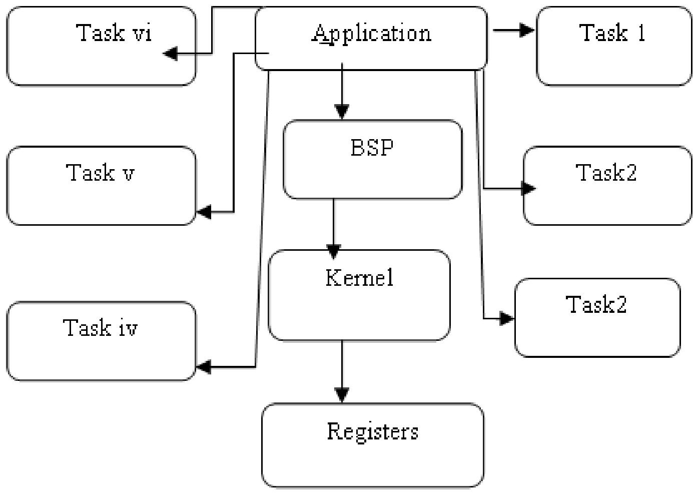

Figure 4: Block diagram

In Micro C/OS-II maximum number of tasks is 64. In the figure shown above the application has 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 (ie., 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. Micro C/OS-II kernel is ported in ARM powered microcontroller for the implementation of multitasking and time scheduling as shown in previous sections.

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 and digital signal controller. 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 focus on designing an embedded monitoring system using ARM 7 and μC/OS-II. The steps involved in porting the RTOS and final implementation details are provided. This paper provides an detailed overview for developing a 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 Em bedded 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] www.uCOS-II.com

## AUTHORS

**First Author** – Ch. S. L. Prasanna, Research Scholar, Dept. of ECM, K L University, A.P, India,
Email id - ch.prasannaece@gmail.com

**Second Author -** M. Venkateswara rao, Assistant professor, Dept. of ECM, K L University, A.P, India,
Email id - venkatt.vlsi@gmail.com