Learning Embedded System using advanced Microcontroller and Real Time Operating System

Nivesh Dwivedi

Abstract— **This Paper emphasizes the learning of embedded system's programming and design through porting of µCOS-II on ARM Cortex M-3.This paper will help the engineering and Embedded System students to start their projects and designing of real time embedded system. It deals with the porting of Micro COS-II in ARM based microcontroller for the implementation of multitasking and time scheduling. Here the real time operating system 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 ARM Cortex M-3 microcontroller like LED, SYSTIC TIMER, BUZZER, UART, LCD, ADC, SPI etc. are tested. This paper mainly concentrates on the porting of µCOS-II.**

Index Terms— **Embedded systems, ARM Cortex M-3, KEIL IDE and real time Kernel.**

I. INTRODUCTION

Engineering students study many subjects like

- Microprocessor
- Microcontroller
- Operating Systems
- Embedded Systems etc.

But they are not able to make use of all these things in real time applications. And in fact it is not possible in very hectic schedule of their college life to cover all the perspectives. This paper inclusively will make them able to learn embedded system and design real time application monitoring systems. Reading this paper you will feel that you can confidently start to work on real time systems and their design. I will make it evident in detailed work of the paper using advanced microcontroller and real time kernel.

The important trait of using real time kernel is MULTITASKING.' Using a real time operating system we can design real time systems performing multiple tasks simultaneously like LED blinking/glowing, alarm, temperature sensor, displaying LCD and serial communication etc. Real time systems that are intensively used in critical areas like space research and defense applications etc. To realize an industrial real time application Monitoring Systems. The heart of the system is a real time kernel that uses preemptive scheduling to achieve multitasking on any embedded platform.
Earlier systems are non-real time operating systems which are often quite non-deterministic and slow responsiveness. So use of real time operating system is just an overcome on non-real time system which is based on performing mono-task mechanism that hardly satisfies the current requirements(one task) thus will cause more power consumption.
Related Works: - Already the porting of µCOS-II has been done earlier but the thing is- Can we apply some different way than to earlier? Yes, let's try it some different way.
Earlier, porting of µCOS-II is given in Micrium 'µCOS-II and ARM Cortex M-3'. They used IAR IDE and a different SOC. I really appreciate the book and author. But this paper intensively will give a constraint and whole idea to students to work on embedded system and its design.
Detailed Works: - To complete this project we need to have the knowledge of the followings-
1. ARM Cortex M-3 and its peripherals.
2. Familiar with Keil IDE.
3. µCOS-II, Real Time Operating System.

1. ARM Cortex M-3
What Is the ARM (advance RISC machine) Cortex M-3?
The microcontroller market is very vast. A bewildering array of vendors, devices, and architectures is competing in this market. The requirement for higher performance microcontrollers has been driven globally by the industry's changing needs; for example, microcontrollers are required to handle more work without increasing a product's frequency or power. In addition, microcontrollers are becoming increasingly connected, whether by Universal Serial Bus (USB), Ethernet, or wireless radio, and hence, the processing needed to support these communication channels and advanced peripherals are growing. Similarly, general application complexity is on the increase, driven by more sophisticated user interfaces, multimedia requirements, system speed, and convergence of functionalities.
The Cortex-M3 is a 32-bit microprocessor. It has a 32-bit data path, a 32-bit register bank, and 32-bit memory interfaces. The processor has a Harvard architecture, which means that it has a separate instruction bus and data bus. This allows instructions and data accesses to take place at the same time, and as a result of this, the performance of the processor increases because data accesses do not affect the instruction pipeline. This feature results in multiple bus interfaces on Cortex-M3, each with optimized usage and the ability to be used simultaneously. However, the instruction and data buses share the same memory space (a unified memory system). In other words, you cannot get 8 GB of memory space just because you have separate bus interfaces. For complex applications that require more memory system features, the Cortex-M3 processor has an optional Memory Protection Unit (MPU), and it is possible to use an external cache if it's required. Both little endian and big endian memory systems are supported. The Cortex-M3 processor includes a number of fixed internal debugging components. These components provide debugging operation supports and features, such as breakpoints and watch points. In addition, optional

Nivesh Dwivedi, B.Tech/Electronics IV year, Hans Raj College, University of Delhi.

components provide debugging features, such as instruction trace, and various types of debugging interfaces.
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. The Cortex-M3 processor has registers R0 through R15. R0–R12 are 32-bit general-purpose registers for data operations. Some 16-bit Thumb instructions can only access a subset of these registers (low registers, R0–R7). The Cortex-M3 contains two stack pointers (R13). They are banked so that only one is visible at a time. The two stack pointers are follows-

• Main Stack Pointer (MSP): The default stack pointer, used by the operating system (OS) kernel and exception handlers.
• Process Stack Pointer (PSP): Used by user application code.
R14 (The link register): - When a subroutine is called, the return address is stored in the link register.
R15 (The program Counter):- The program counter is the current program address. This register can be written to control the program flow.
Special registers: The Cortex-M3 processor also has a number of special registers. They are as follows-
• Program Status Register (PSRs)
•Interrupt Mask registers (PRIMASK, FAULTMASK, and BASEPRI)
• Control registers (CONTROL)
These registers have special functions and can be accessed only by special instructions. They cannot be used for normal data processing.

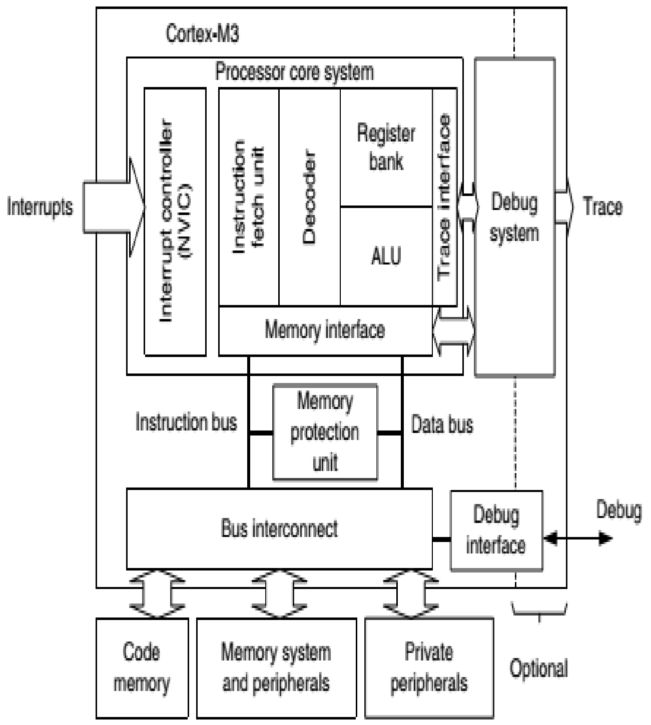

Fig.1:- A Simplified View of ARM Cortex M-3

The Cortex-M3 addresses the requirements for the 32-bit embedded processor market in the following ways-
Greater performance efficiency, Low power consumption, enhanced determinism, improved code density, Ease of use, Lower cost solutions, Wide choice of development tools
These above are the merits that make ARM Cortex m-3 suitable for our porting purpose.
The Cortex-M3 processor is based on one profile of the v7 architecture, called ARM v7-M, an architecture specification for microcontroller products. Cortex-M3 supports only the Thumb-2 (and traditional Thumb) instruction set. Instead of using ARM instructions for some operations, as in traditional ARM processors, it uses the Thumb-2 instruction set for all operations.
The details of the ARMv7-M architecture are documented in *The ARMv7-M Architecture Application Level Reference Manual*. This document can be obtained via the ARM web site through a simple registration process. The ARMv7-M architecture contains the following key areas:
- Programmer's model
- Instruction set
- Memory model
- Debug architecture

Processor-specific information, such as interface details and timing, is documented in the *Cortex-M3 Technical Reference Manual (TRM)*. This manual can be accessed freely on the ARM website.

Cortex-M3 Processor Applications
With its high performance and high code density and small silicon footprint, the Cortex-M3 processor is ideal for a wide variety of applications as-

• *Low-cost microcontrollers*
• *Automotive*
• *Data communications*
• *Industrial control*
• *Consumer products*
There are already many Cortex-M3 processor-based products on the market, including low-end
Products priced as low as US$1, making the cost of ARM microcontrollers comparable to or lower than that of many 8-bit microcontrollers.

II. Keil IDE Software

Keil IDE is a windows operating system (os) software program that runs on a PC to develop applications of 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.
The Keil compiler is the industry standard and supports more than 500 current 8051 device variants. Now, Keil software offers development tools for ARM.

Keil Software, world's leading developer of Embedded Systems Software, makes ANSI C compilers, macro assemblers, real-time kernels, debuggers, linkers, library managers, simulators, integrated environments, and evaluation boards for the 8051, 251, ARM7, and C16x/ST10 microcontroller families. Keil Software implemented the first C compiler designed from the ground-up specifically for the 8051 microcontroller.
Keil development tools offer a complete development environment for ARM Cortex-M, and Cortex-R

processor-based devices. They are easy to learn and use, yet powerful enough for the most demanding embedded applications.

In this project i will use keil IDE software micro Vision-5.

III. µCOS-II

Introduction: - µCOS-II (pronounced "Micro C O S 2") stands for Micro-Controller Operating System Version 2. µCOS-II is upward compatible with µCOS (V1.11) but provides many improvements over µCOS such as the addition of a fixed-sized memory manager, user definable callouts on task creation, task deletion, task switch and system tick, supports TCB extensions, stack checking and, much more.

If you currently have an application (i.e. product) that runs with µCOS, your application should be able to run, virtually unchanged, with µCOS-II. All of the services (i.e. function calls) provided by µCOS have been preserved. You may, however, have to change include files and product build files to ‘point ’to the new file names. µCOS-II was developed and tested on a PC; µCOS-II was actually targeted for embedded systems and can easily be ported to many different processor architectures.

It 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. Source code for µCOS-II is free but not for commercial purpose. If you want to use it as commercial purpose, you have to take permission.

Selecting µCOS-II: - There are the following features which make µCOS-II suitable/convenient to port-

- Portable
- ROMABLE
- Scalable
- Preemptive
- Multi-tasking
- Deterministic
- Task stacks
- Services
- Interrupt Management
- Robust and reliable

PORTING OF µCOS-II

Adapting a real-time kernel to a microprocessor or a microcontroller is called a port. Most of µCOS- II is written in C for portability; however, it is still necessary to write some processor specific code in C and assembly language. Specifically, µCOS-II manipulates processor registers which can only be done through assembly language.

Porting µCOS -II to different processors is not so much difficult task only because µCOS -II was designed to be portable.

If you are going to port µCOS-II for your processor, of course you need to know how µCOS-II’s processor specific code works.

A processor can run µCOS-II if it satisfies the following requirements:

1. You must have a C compiler for the processor and the C compiler must be able to produce reentrant code.
2. You must be able to disable and enable interrupts from C.
3. The processor must support interrupts and you need to provide an interrupt that occurs at regular intervals (typically between 10 to 100 Hz).
4. The processor must support a hardware stack, and the processor must be able to store a fair amount of data on the stack (possibly many Kbytes).
5. The processor must have instructions to load and store the stack pointer and other CPU registers either on the stack or in memory.

ARM Cortex M-3 satisfies all the above requirements so we can easily port µCOS-II in it.

Porting µCOS -II is actually quite straightforward once you understand the subtleties of the target processor and the C compiler you will be using.

If your processor and compiler satisfy µCOS -II’s requirements, and you have all the necessary tools, porting µCOS-II consists of the followings-

1. setting the value of 1 #define constants (OS_CPU.H)
2. Declaring 10 data types (OS_CPU.H)
3. Declaring 3 #define macros (OS_CPU.H)
4. Writing 6 simple functions in C (OS_CPU_C.C)
5. Writing 4 assembly language functions (OS_CPU_A.ASM)

All the source codes, you need not to write by your own but you should understand its working functionality well. These source codes are easily available so you can use these directly on your initial stage of porting because these are the processor independent codes. You need to work on processor dependent codes and your application codes. Also you have to add ‘INCLUDES.H’.

INCLUDES.H allows every .C file in your project to be written without concerns about which header file will actually be needed.

Depending on the processor, a port can consist of writing or changing between 50 and 300 lines of code.

Starting and Initializing µCOS-II

a. Starting µCOS-II: - µCOS-II starts in the same way as shown in the fig.2. First we will initialize both the hardware and software .Here the hardware i have used is the ARM Cortex M-3 and software is the real time operating system µCOS-II. The resources are allocated for the tasks defined in the application then the scheduler is started and it schedules the tasks in pre-emptive manner.

b. Initialization of µCOS-II: - The steps to initialize µCOS-II are shown in Fig.3. We will follow the corresponding steps to initialize it.

The Steps we will take to initialize µCOS-II through programming is shown below-

```
Void main (void)
  {

  /* User initialization*/
  OSInit ( ); /* kernel initialization */

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

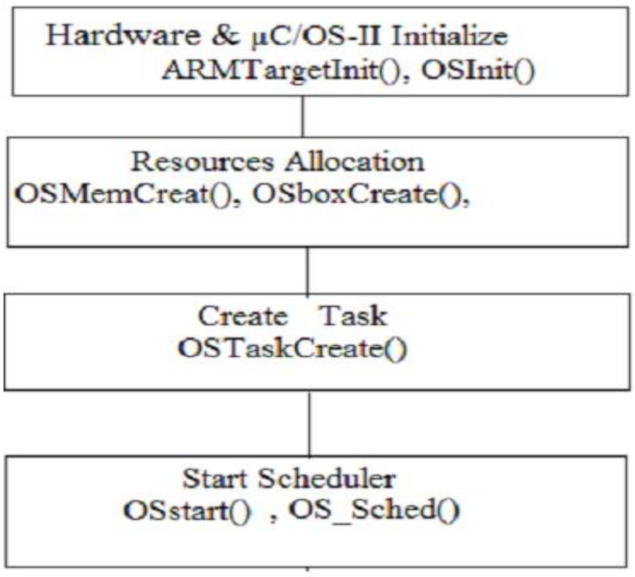
Hardware & µC/OS-II Initialize
ARMTargetInit(), OSInit()
Resources Allocation
OSMemCreat(), OSboxCreate(),
Create Task
OSTaskCreate()
Start Scheduler
OSstart() , OS_Sched()

Fig. 2:- Starting of µCOS-II

Creating Task:- For multitasking , the µCOS-II needs to have information about the task, its starting address, top-of-stack (TOS), priority, arguments passed to the task etc.
You can create a task by calling a service provider by µCOS-II in the following way-
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 can create the task before you start multitasking (at initialization time).

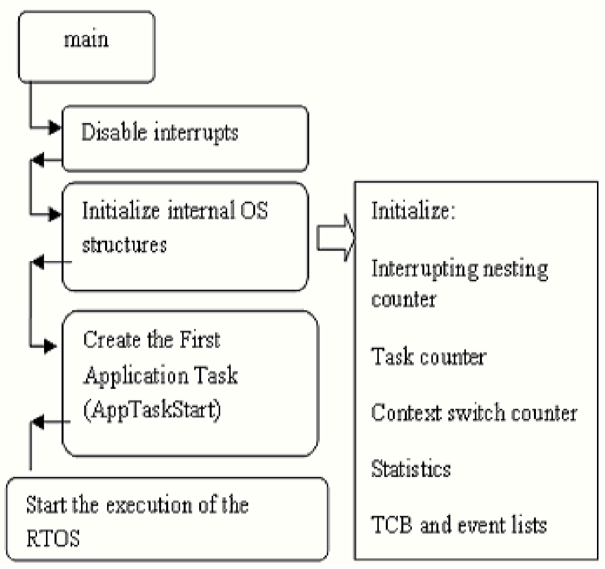
main
Disable interrupts
Initialize internal OS structures
Create the First Application Task (AppTaskStart)
Start the execution of the RTOS
Initialize:
Interrupting nesting counter
Task counter
Context switch counter
Statistics
TCB and event lists

Fig. 3:- Initializing µCOS-II

IV. Architecture

In every embedded systems, there is a board support package (BSP) for a given 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.

Hardware and Software Architecture:-Given fig. 4 shows a block diagram of the relationship between your application, µCOS-II, the µCOS-II port, the BSP (Board Support Package), the ARM Cortex-M3 CPU and the target hardware. APP.C is a standard test file for µCOS-II. APP.C would be where you would place main( ) but, of course, you can place main( )anywhere you want.
The two important functions are-
1. Main ( ) and
2. AppStartTask ( )

Function main ( ):-
```
void main (void)
{
#if OS_TASK_NAME_SIZE > 13
        INT8U err;
#endif
BSP_IntDisAll ( );
OSInit ( );
OSTaskCreateExt (AppStartTask,
           (void *) 0,
           (OS_STK                    *)&AppStartTaskStk
[APP_TASK_START_STK_SIZE-1],
APP_TASK_START_PRIO,
           (OS_STK *)&AppStartTaskStk [0],
APP_TASK_START_STK_SIZE,
           (void *) 0,
       OS_TASK_OPT_STK_CHK                          |
OS_TASK_OPT_STK_CLR);

#if OS_TASK_NAME_SIZE > 11
    OSTaskNameSet   (APP_TASK_START_PRIO,   "Start
Task", &err);
#endif

OSStart ( );
}
```

AppStartTask ( ):-
```
static void AppStartTask (void *p_arg)
{
  (void) p_arg;

BSP_Init ( );

OS_CPU_SysTickInit ( );

#if OS_TASK_STAT_EN > 0
    OSStatInit ( );
#endif
AppTaskCreate ( );

While (TRUE) {
   /* Do something 'useful' in this task */
LED_Toggle (1);
OSTimeDly (OS_TICKS_PER_SEC / 20);
    }
}
```

Once you have a port of µCOS-II for your processor, you will need to verify its operation. Testing a multitasking real -time kernel such as µCOS-II is not as complicated as you may think. You should test your port without application code. In other words, test the operations of the kernel by itself. Also you can test it by checking whether context switching is happening or not on your Register window in KEIL IDE. There are two reasons to do this. First, you don't want to complicate things any more than they need to be. Second, if something doesn't work, you know that the problem lies in the port as opposed to your application. Start with a couple of simple tasks and only the ticker interrupt service routine. Once you get multitasking going, it's quite simple to add your application tasks.

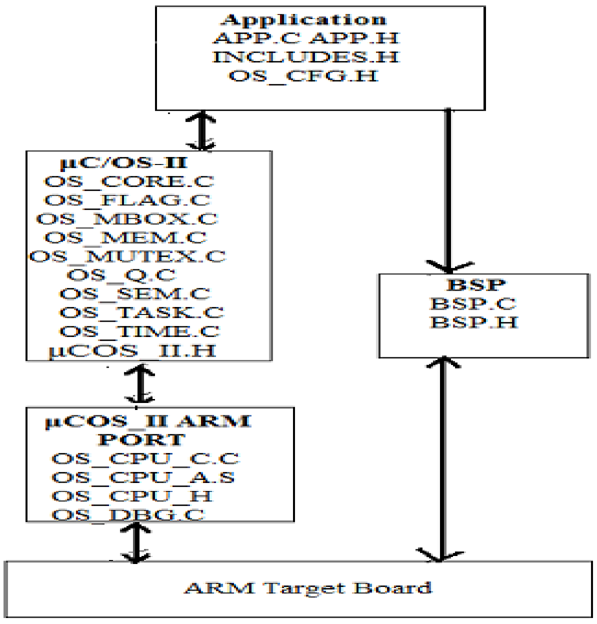

Figure 4:- Relationship between modules.

V. Implementation

The Real Time Kernel is the most important thing in any real time system that does pre-emtive scheduling to perform multitasking which is the real trait of RTOS. This research paper emphasizes the implementation of hardware and software together.

In µCOS-II maximum 64 tasks we can perform simultaneously but here we have six tasks in fig.5 shown below.

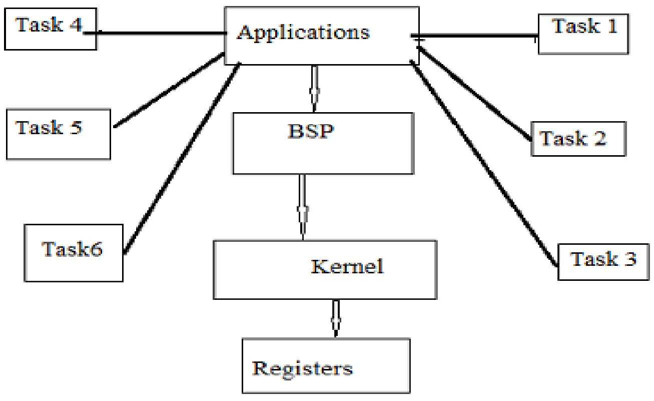

Fig.5:- Implementation of hardware and software.

Depending on our requirement we can vary the number of tasks at a time. Here, to verify our porting of µCOS-II we can perform several tasks like LED blinking, Systick Timer, Buzzer (we can generate desired music or alarm), UART, Displaying LCD etc. We can also perform many projects Like Home automation using Bluetooth and UART together , Noticeboard display using Bluetooth, LCD and UART together etc. But it is enough to perform two-three tasks to test porting of our µCOS-II. Thus we can port µCOS-II using development tools i.e. ARM Cortex M-3 and KEIL IDE.

VI. Conclusion

In this Research paper the porting of a real time operating system µCOS-II on ARM Cortex M-3 using software keil µvision-5 is presented. It mainly concentrates on development of an embedded monitoring system using ARM Cortex M-3 and Real Time Kernel. All the steps taken while porting the µCOS-II and implementation thesis are provided in the paper. The paper gives a detailed overview that will help the students to develop and design an embedded monitoring system using ARM cortex M-3 and Real time operating system .

Acknowledgement

I would like to acknowledge the Centre for Development and Advance Computing (C-DAC), Hyderabad, Govt. of India and their faculties to train and inspire for the work. I would also like to place on record my sincere thanks and gratitude to Shri Sanjay Kr. Vyas Scientist 'E' / Additional Director, HRD Division, Dept. of Electronics and Information Technology (DeitY), Ministry of Communication and IT, govt. of India for his continuous guidance and support for my project work.

References

[1] Micrium µC/OS-II for the ARM Cortex-M3 Processors and www.micrium.com
[2] www.arm.com
[3]µCOS-II, The Real Time Kernel, and http://www.uCOS-II.com
[4] Design of µC/ Os II RTOS Based Scalable Cost Effective Monitoring System Using Arm Powered Microcontroller, M. Venkateswara Rao, Dept. of ECM, K L University, A.P, India.
[5]Jean J Labrosse, MicroC/OS-II the Real-Time Kernel, Second Edition Beijing University of Aeronautics and Astronautics Press.
[6]Tianmiao Wang the Design and Development of Embedded System Based on ARM Micro System and µC/OS-II Real-Time Operating System Tsinghua University Press.
[7]The Definitive Guide to the ARM Cortex M-3, second edition by Joseph Yiu.

Nivesh Dwivedi, currently pursuing B.Tech Electronics IV year from Hans Raj College, University Of Delhi. His Current interest is to work in the field of Embedded Systems and Digital Image Processing. Currently He is also working on a project *"Image Registration using two stereo Camera"* with Helium Ink Company, Pune. He has been awarded by Mr Akhilesh Yadav, Chief Minister of U.P. with 'Award of Excellence' for his excellent performance in XII class. He is very active to participate in technical Seminars, Workshops as well as extracurricular activities.