International Journal of Engineering Research and Development
eISSN : 2278-067X, pISSN : 2278-800X, www.ijerd.com
Volume 4, Issue 8 (November 2012), PP. 01-04

RTOS – RT Linux Porting On S3Cmini2440 ARM9 Board

Ravindra Prasad Bonthu[1] and Subrahmanyam Venkata Veeraghanta[2]
Department of Computer Science Engineering, Sphoorthy Engineering College, Hyderabad, Andhra Pradesh, India

Abstract:—In applications where time constraints are very important, needs rtos functionality and the need of the operating system in real time environment .A real-time operating system (RTOS) is key to many embedded systems today, provides a software platform upon which to build applications. Not all embedded systems are designed with RTOS. Some embedded systems with relatively simple hardware or a small amount of software application code might not require an RTOS. Many embedded systems, however, with moderate-to-large software applications require some form of scheduling of the processes, and these systems require a RTOS, to implement that type of applications it is mandatory to port rtos on the target machine to achieve greater throughput. In this paper we tried porting the RT Linux a RTOS on Samsungs S3CMINI2440 ARM9 Board.

Keywords:—RTOS, RTLinux, S3CMINI2440, ARM9, Porting.

I. INTRODUCTION

A REAL-TIME OPERATING SYSTEM (RTOS) IS A PROGRAM THAT SCHEDULES EXECUTION IN A TIMELY MANNER, MANAGES SYSTEM RESOURCES, AND PROVIDES A CONSISTENT FOUNDATION FOR DEVELOPING APPLICATION CODE. APPLICATION CODE DESIGNED ON AN RTOS CAN BE QUITE DIVERSE, RANGING FROM A SIMPLE APPLICATION FOR A DIGITAL STOPWATCH TO A MUCH MORE COMPLEX APPLICATION FOR AIRCRAFT NAVIGATION. GOOD RTOSES THEREFORE, ARE SCALABLE IN ORDER TO MEET DIFFERENT SETS OF REQUIREMENTS FOR DIFFERENT APPLICATIONS. FOR EXAMPLE, IN SOME APPLICATIONS AN RTOS COMPRISES ONLY A KERNEL, WHICH IS THE CORE SUPERVISORY SOFTWARE THAT PROVIDES MINIMAL LOGIC, SCHEDULING, AND RESOURCE-MANAGEMENT ALGORITHMS. EVERY RTOS HAS A KERNEL. ON THE OTHER HAND, AN RTOS CAN BE A COMBINATION OF VARIOUS MODULES, INCLUDING THE KERNEL, A FILE SYSTEM, NETWORKING PROTOCOL STACKS, AND OTHER COMPONENTS REQUIRED FOR A PARTICULAR APPLICATION. NORMALLY MOST OF THE RTOS KERNELS CONTAIN THE FOLLOWING COMPONENTS: SCHEDULER-IS CONTAINED WITHIN EACH KERNEL AND FOLLOWS A SET OF ALGORITHMS THAT DETERMINES WHICH TASK EXECUTES WHEN. SOME COMMON EXAMPLES OF SCHEDULING ALGORITHMS INCLUDE ROUND-ROBIN AND PREEMPTIVE SCHEDULING. OBJECTS-ARE SPECIAL KERNEL CONSTRUCTS THAT HELP DEVELOPERS CREATE APPLICATIONS FOR REAL-TIME EMBEDDED SYSTEMS. COMMON KERNEL OBJECTS INCLUDE TASKS, SEMAPHORES, AND MESSAGE QUEUES. SERVICES- ARE OPERATIONS THAT THE KERNEL PERFORMS ON AN OBJECT OR, GENERALLY OPERATIONS SUCH AS TIMING, INTERRUPT HANDLING, AND RESOURCE MANAGEMENT.

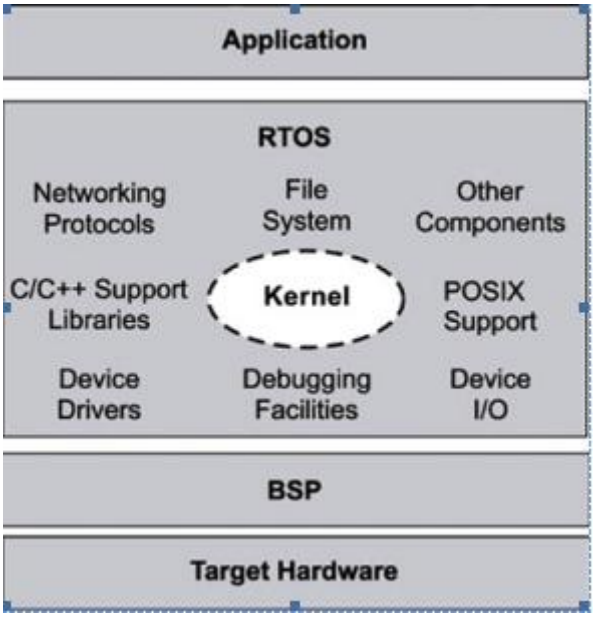

Fig1. *A High level view of RTOS*

II. WHAT IS REALTIME

1. Real time is a level of responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process.
2. Real time describes a human rather than machine sense of time.
3. It is the class of computers systems that interacts with the external world in a time frame defined by the external world.
4. It is the system in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced.
5. If the timing constraints of the system are not met system failure is said to have occurred.

III. WHAT IS RTOS

1. A real time operating system (RTOS) is an operating system that guarantees a certain capability within a specified time constraint.
2. An OS is a system program that provides an interface between application programs and the computer system (hardware)
3. The applications where dependability that a certain task will finish before a particular deadline is just as obtaining the correct results.
4. Besides meeting deadlines RTOS must also be able to respond predictably to unpredictable events and process multiple events concurrently.
5. A system application/computer/operating system operates in real time to the degree that those of its actions which have time constraints are performed with acceptable timeliness.
6. A system is real time the degree that it employs real time resource management .The resources are explicitly managed for the purpose of operating in real time.

IV. KEY CHARACTERISTICS OF AN RTOS

An application's requirements define the requirements of its underlying RTOS. Some of the key characteristics of an RTOS are ***Reliability:*** In general embedded system products must be reliable. the system need to run for longer periods. even when the systems are running for longer period there is no change in the throughput. hence in real time applications reliability is one of the requirement for getting obedient results. A common way that developers categorize highly reliable systems is by quantifying their downtime per year. While RTOSes must be reliable, note that the RTOS by itself is not what is measured to determine system reliability. It is the combination of all system elements-including the hardware, BSP, RTOS, and application-that determines the reliability of a system. ***Predictability:*** Because many embedded systems are also real-time systems, meeting time requirements is key to ensuring proper operation. The RTOS used in this case needs to be predictable to a certain degree. The term deterministic describes RTOSes with predictable behaviour, in which the completion of operating system calls occurs within known timeframes. ***Performance:*** This requirement dictates that an embedded system must perform fast enough to fulfill its timing requirements. Typically, the processor's performance is expressed in million instructions per second (MIPS). ***Compactness:*** Application design constraints and cost constraints help determine how compact an embedded system can be. In such embedded systems, where hardware real estate is limited due to size and costs, the RTOS clearly must be small and efficient. In these cases, the RTOS memory footprint can be an important factor. To meet total system requirements, designers must understand both the static and dynamic memory consumption of the RTOS and the application that will run on it. ***Scalability:*** Because RTOSes can be used in a wide variety of embedded systems, they must be able to scale up or down to meet application-specific requirements. Depending on how much functionality is required, an RTOS should be capable of adding or deleting modular components, including file systems and protocol stacks.

V. REQUIREMENTS FOR GOOD RTOS

Multitasking Capabilities: A RT application is divided into multiple tasks. The separation into tasks helps to keep the CPU busy.
Short Interrupt Latency : Interrupt Latency = Hardware delay to get interrupt signal to the processor + time to complete the current instruction + time executing system code in preparation for transferring execution to the devices interrupt handler.
Fast Context Switch: The time between the OS recognizing that the awaited event has arrived and the beginning of the waiting task is called context switch time(dispatch latency).This switching time should be minimum.
Control of Memory Management: an OS should provide way for task to lock its code and data into real memory so that it can guarantee predictable response to a interrupt.
Proper Scheduling: OS must provide facility to schedule properly time constrained tasks.

VI. FEATURES OF RTOS

6.1. Multitasking: There are two ways to achieve Multitasking to share CPU time between two or more tasks.
Pre-emptive Multitasking:
a) An external tick interrupt, interrupts task at an indeterminate point and passes control to kernel program.
b) The kernel will save the state of the interrupted task and then determine which task it should run next.
c) The Kernel restores the state of task and pass control of the CPU to that task.
d) Task will continue to run until it is interrupted by next external ticks interrupt or voluntarily gives up allotted time slice.

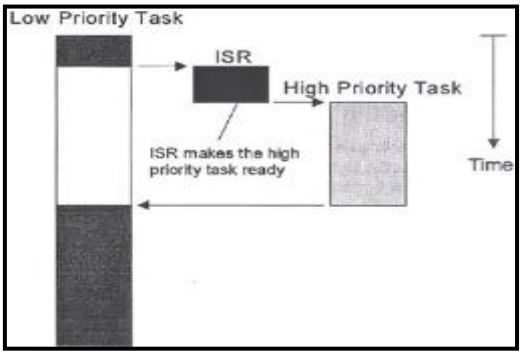

Fig2 : *Pre-emptive multitasking*

Non-preemptive Multitasking

a) In Non-pre-emptive multitasking a task is designed to relinquish control of the CPU to the kernel at regular intervals.
b) To the implementation of non-pre-emptive multitasking is the use of a message queue.
c) The kernel manages the message queue which is essentially a queue of message numbers & small amount of data.

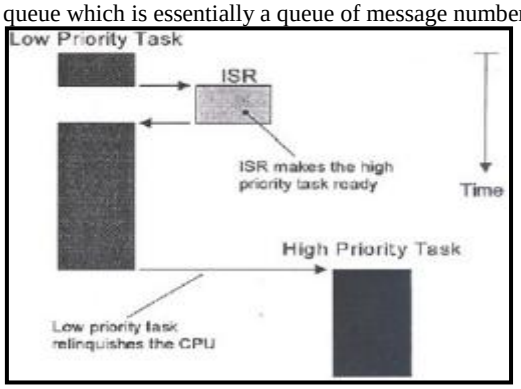

Fig3 *: Non-Pre-emptive multitasking*

6.2. Process thread can be prioritized

a) A OS schedules threads/or processes based on their relative priorities.
b) Processes are forced to relinquish the processor if a higher priority process becomes ready to run.
c) Highest priority runnable process will be executing on the system at all times

6.3. Sufficient number of interrupt levels

a) Advantage of interrupt threads is that they permit the OS to be more responsive to the devices it services.
b) ISR typically disable other interrupts while executing using interrupt threads permits interrupts to be responded at more regular intervals.

VII. BUILDING RT-IMAGE

In this paper we are basically focusing the rtos of the type RT_Linux. since linux is an open source we can get the linux source code easily (since it is open source) from http://www.kernel.org. After getting the source code we can compile the source code in two ways: ***Native Compilation***-In this we can compile, install the image and boot the image within same system. ***Cross Compilation***-In this case we can compile the source code in one machine and will port on another machine (Like ARM).

To port RT_Image on ARM9 (Samsung's s3cmini2440) the target code will undergo the cross compilation process. To do the cross compilation of the required RT_Image changes can be done in the Make File of the kernel source code. For example, the target Architecture (ARM) and the required Cross-Compiler (ARM GCC) are two important parameters to be changed in the kernel source code to convert the given kernel to RTOS Kernel. The kernel source code will be configured as per the target board specification, compiled to get the board specific image. To convert the given board specific image, the RT Linux Patch will be added to the board specific image and recompiled. Now the RT_Image will be available in the path of ARCH/ARM/BOOT folder of the kernel source (where we compiled and added the rtlinux patch). Now this final image will be ported to the target board, i.e. Samsungs s3cmini2440 board in our experiment.

Following process will explain how to port the rt_image onto the target board i.e Samsungs s3cmini2440 arm board using USB connectors. In our experiment the Target machine will be connected to the host by using USB cable. To get the options connect a serial cable to the PC and the target board (here Samsungs s3cmini2440). Open the terminal in PC to check the different options provided by the board manufacturer to port the rt_image on to the board. For example: K: Loading the kernel. B: Booting the kernel and so on…. Following the sample output from the Samsungs s3cmini2440 ARM board when connected to PC using Serial Cable.

\0xfe\0xe0
FriendlyARM BIOS 2.0 for 2440
[x] Format NAND FLASH for Linux

[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[a] Absolute User Application
[n] Download Nboot for WinCE
[l] Download WinCE boot-logo
[w] Download WinCE NK.bin
[d] Download & Run
[z] Download zImage into RAM
[g] Boot linux from RAM
[f] Format the nand flash
[b] Boot the system
[s] Set the boot parameters
[u] Backup NAND Flash to HOST through USB (upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
[i] Version: 1138-2K
Enter your selection:
From the above options select the option k as …
Enter your selection: k
USB host is connected. Waiting a download.

It will show the above message in cutecom. Now open another terminal to port the image on the board and give the command as follows.

ravi@ravi-OEM:/usr/src/linux-2.6.33.9/arch/arm/boot$ sudo ./usbpush zImage 0x30008000
[sudo] password for ravi:
csum = 0x2230
send_file: addr = 0x30008000, len = 0x0022e4b0
ravi@ravi-OEM:/usr/src/linux-2.6.33.9/arch/arm/boot$
After check out the cute com we will get the information as follows…
Now, Downloading [ADDRESS: 30000000h,TOTAL:2286778]
RECEIVED FILE SIZE: 0\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08 65536\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08
589832\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08
1114120\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08
1638408\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08\0x08
2286778 (97KB/S, 23S)
Downloaded file at 0x30000000, size = 2286768 bytes
Write to flash ok: skipped size = 0x60000, size = 0x22e4b0
It indicates that the image has been successfully ported on to the target board.

If we want boot the image then select option b from the list. Now the image will boot from the target board. If the image is successfully booted then our porting is successful.

VIII. CONCLUSIONS

In this paper we presented the information about rtos, and key characteristics of rtos, requirements of good rtos and features of rtos that is multi tasking nature. How to build the rt_image and the configuration details. After configuring how to port the image from the host machine to the target machine. How to choose the options for the loading the image and booting the image .if once real time image is ported on the target machine can use for the typical applications .and also will get the reliability and scalability depending on the application.

ACKNOWLEDGEMENTS

We would like to thank everyone in Department of Computer Science and Engineering, Sphoorthy Engineering College, Hyderabad. Also would like to thank the research team at Vector India (P) Ltd, Hyderabad. who helped in understanding the embedded systems concepts, and other concepts related to Wireless Communication.The author would like to thanks the anonymous referees for their valuable comments, which greatly improved the readability of the paper.

REFERENCES

[1]. Real-time concepts for embedded systems / Qing Li; with Caroline Yao.p. cm.
[2]. Dedicated Systems Experts, *What makes a good RTOS.* Brussels, Belgium: Dedicated Systems Experts, 2001.
[3]. Jean J Labrosse, *The Real-Time Kernel* . 2nd ed. Gilroy, CA: CMP Books, 2002.
[4]. The ARM Linux Project – http://www.arm.linux.org.uk/