Volume 3, Issue IX, Sep 2015
ISSN 2320-6802

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY
WINGS TO YOUR THOUGHTS…..

Analysis of porting Free RTOS on MSP430 architecture and study of performance parameters on small factor Embedded Systems

Nandana V.[1], Jithendran A.[2], Shreelekshmi R.[3]

[1]M.Tech Scholar, LBSITW, Poojappura, Thiruvananthapuram 695012
nandanatvm@gmail.com
[2]Principal Architect EDU,QuEST Global, Thiruvananthapuram 695581
jithendran.a@quest-global.com
[3]Professor and HOD, CSE, LBSITW, Poojappura, Thiruvananthapuram 695012
shreelekshmir@gmail.com

***Abstract**: Unlike the General Purpose Operation System (GPOS) which focuses on the amount of work done within a given time frame, an RTOS is more focused on the criticality of timeliness. This feature is highly essential in real time embedded systems. Extensive amount of research work has been carried out in the field of RTOS. Every RTOS has a distinct feature of its own. But the growth in this field is not fully utilized by the developers yet as most of the RTOS is licensed and highly expensive. Peripheral support and stack availability also varies widely from one RTOS to another. Out of the few available free/open source RTOS, support for proprietary protocols is minimal. This paper aims at familiarizing with FreeRTOS one of the widely used open source RTOS in the embedded world. The popularity of FreeRTOS is mainly attributed to the number of ports available for it as well as the development support provided by the FreeRTOS community. FreeRTOS is ported to MSP430 based hardware, and its performance and power management features are studied. The constraints of transition of software/stacks from scheduler to light weight real time operating system are also explored.*
***Keywords:** RTOS, GPOS, FreeRTOS, MSP430.*

1. INTRODUCTION

Embedded systems are widely used in our day to day affairs. It has application in smart home appliances, business, communication, entertainment etc. Since the demand is high, strict deadlines need to be met for embedded projects. Out of the numerous RTOS available in the market, determining which RTOS is optimal for their project requirements is a tedious task [1]. Different RTOS works efficiently in different hardware. Hence the target hardware need to be decided and it should be made available within the timeline of the project. Development tools required for the particular RTOS also need to be considered. Tool should be acquired well in advance and developers should be familiarized with the programming environment. Peripheral support and stack availability also varies widely from one RTOS to another. Out of the few available free/open source RTOS, support for proprietary protocols is minimal[2,3,4]. Hence it is the responsibility of the developer to ensure that the RTOS chosen can support the protocols required by the project.
Since developers do not have the luxury of time to solve all the above problems, a majority of them continue with the traditional design methods. This paper aims at familiarizing with FreeRTOS [5], one of the widely used RTOS available in the market. The efficiency of FreeRTOS is studied as compared to a normal scheduling algorithm. The constraints of transition of software/stacks from scheduler to light weight real time operating system are also explored.

2. LITERATURE REVIEW

FreeRTOS is one of the markets leading RTOS in the present world. It is licensed under GPL (General Purpose License) with an optional exception. The changes made to the core kernel must be made open source whereas the application code can be made proprietary. This feature makes it attractive to be used in commercial applications. The kernel is entirely written in C. The popularity of FreeRTOS is mainly attributed to the number of ports available with it as well as the development support provided by the FreeRTOS community. It is successfully ported to more than thirty microcontrollers.
The core RTOS code is contained in three files tasks.c, queue.c and list.c within the source folder [5]. There are three optional files timers.c, coroutine.c and eventgroups.c. The Demo directory of every port contains a special file called FreeRTOSConfig.h. This header filer defines all the configuration settings for that particular port. The configuration settings should be defined as per the application requirements. The entire files are not required for the operating system to run. The three core RTOS files, their corresponding headers and the application code is sufficient. Once the directory structure is familiar, the developer can extract the required files for project requirement. The organization of files in FreeRTOS is a clear indication of its ease of manageability. The core scheduling algorithm is contained within a single folder. The processor specific code is separated from the application code. Hence the changes can be easily made and the code is easier to comprehend.

3. SYSTEM DESIGN

For the purpose of study a scheduling algorithm is considered that uses event driven round robin technique. The transaction scheduler schedules tasks using simple round robin scheduling. An event setting mechanism for a task is used to ready it for scheduling. Hence it means that only a task which has an event set for it (either by any other task or by an ISR) can compete for the CPU. The scheduler checks for task in a round robin fashion and find if an event for them has been set. If an event is set for the task, the scheduler executes the task by calling its associated function. If there is no active task (no task has an event set for it) the scheduler waits in an infinite loop for events to be set for tasks. Both the scheduling algorithm and FreeRTOS is ported to a common hardware and the behavior is studied and compared.

4. HARDWARE

The comparison between the scheduling algorithm and FreeRTOS is studied by porting it to a common hardware. The hardware used is MSP430F5529LP by TI as shown in figure 1 [6]. FreeRTOS port available for the MSP430F5438 Experimenter's board is customized to MSP430F5529 Launch pad .Coding and debugging is done using Code Composer Studio Version 6.0.1.

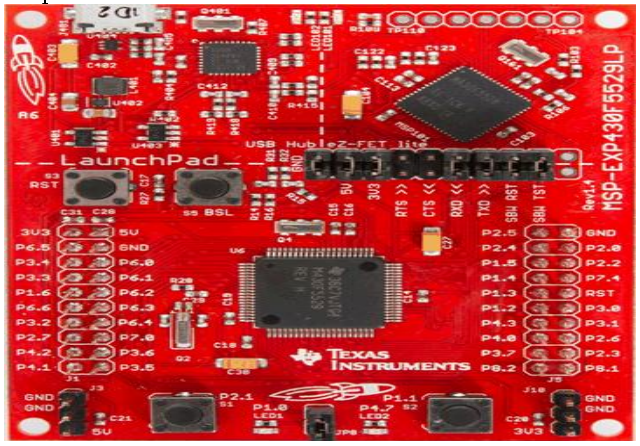

Figure 1: MSP430F5529LP

The board has the following features.

- USB-enabled 16-bit MSP430F5529 MCU.
- eZ-FET lite emulator.
- Up to 25-MHz System Clock.
- 1.8-V to 3.6-V operation.
- 128KB of flash, 8KB of RAM.
- Five timers.
- Serial interfaces (SPI, UART, I2C)
- 12-bit analog-to-digital converter.
- Analog comparator.

5. PORTING

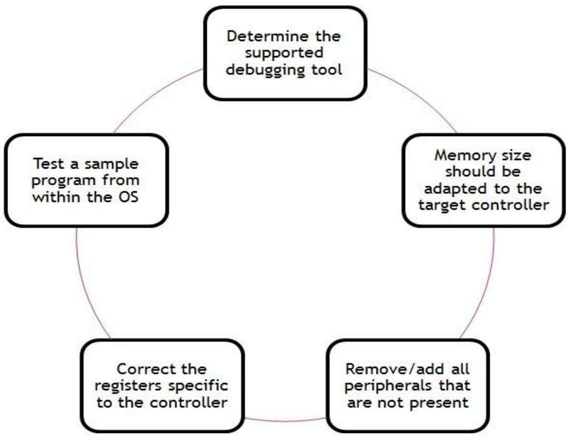
Determine the supported debugging tool
Memory size should be adapted to the target controller
Remove/add all peripherals that are not present
Correct the registers specific to the controller
Test a sample program from within the OS

Figure 2: Steps in porting

Determine the supported debugging tool: Each and every port is compiler dependent. All ports does not support all compiler definitions. Hence a developer should ensure that he has the necessary tools to support the port that he intends to use. In the case of MSP430, ports are available with GCC, IAR, Cross Works and CCS. CCS is selected because of the availability of tool, familiarity and ease of use.

Memory size should be adapted to the target controller: Hardware used contains processor MSP430F5529. But FreeRTOS port is available for MSP430F5438. Hence the port must be customized to the requirements of MSP430F5529. One of the main changes that needs attention is the memory requirement. MSP430F5438 has 16KB RAM and 256KB flash whereas MSP430F5529 has only 8KB RAM and 128 KB flash. Stack and heap size should be changed accordingly so as to meet this memory constraint.

Remove all peripherals that are not present: FreeRTOS port for MSP430F5438 is meant for MSP430F5438 Experimenter's Board whereas the hardware used in this paper is MSP430F5529 Launch pad. Peripherals available in the former might not be available in the latter. Hence changes need to be made accordingly. For e.g. MSP430F5438 Experimenter Board contains dot matrix LCD, microphone, audio output etc. which is not present in MSP430F5529 Launch pad. Code pertaining to all these peripherals must be removed from the port so as to ensure proper functioning.

Correct the registers specific to the controller: Registers in MSP430F5438 is different from that of MSP430F5529. Since all operations in the processor results in manipulation of registers, use of correct registers pertaining to the processor is of high importance. Also driver code needs to be developed as required by the project. For e.g. if the project needs to use UART driver, the code pertaining to the processor MSP430F5529 needs to be developed.

Port the firmware and test: Once the above steps are completed, OS needs to be tested. This is done by executing a simple program like "Hello World". Care must be taken to ensure that the function is executed by the scheduling algorithm of the OS and not from outside. Similarly additional tests can be done to ensure that all drivers are working properly.

6. EXPERIMENTAL RESULTS

6.1 Execution of normal scheduling algorithm

Execution is as shown in figure 3. Initially event 1 of task 1 is executed. This is followed by event 1 of task 2, event 1 of task 3 etc. up to event 1 of task n. Once event 1 of task n is completed, the pointer will again loop back to task 1 and event 2 of task 1 is executed. Such an algorithm is not adaptable to real time systems because the pointer will always move in a sequential fashion. It cannot attend higher priority task as soon as they arrive. Using these algorithm real time tasks of different priorities cannot be executed.

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY
WINGS TO YOUR THOUGHTS…..

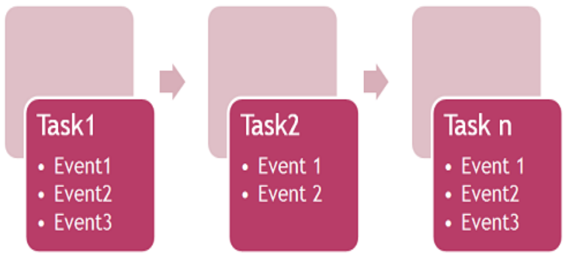

Figure 3: Execution of normal scheduling algorithm

6.2 Execution of FreeRTOS

A higher priority task is never kept waiting for a lower priority task. According to the figure 4 [5], idle task has least priority and vControlTask has highest priority. At time t6, vKeyHandlerTask is executing during which vControlTask is interrupting. Since vControlTask is of higher priority, vKeyHandlerTask is suspended and vControlTask is executed. Once vControlTask is finished, vKeyHandlerTask is resumed. In case of FreeRTOS, scheduling algorithm will be always priority oriented.

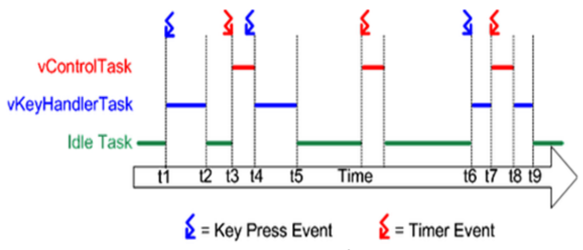

Figure 4: Execution of FreeRTOS

6.3 Code Organization

Scheduling algorithm can be considered as composed of mainly two parts.

- Core scheduling functions
- Header files

Code management of a normal scheduling algorithm is cumbersome. More the requirements of project, maintenance of code becomes difficult. Without proper documentation and familiarity, making even a simple change becomes tedious. Sometimes even locating a particular section of code becomes difficult.

FreeRTOS basically contains five parts. Most important is the core scheduler which is basically composed of task.c, queue.c and list.c. Associated with it, header files are given in a separate folder. A portable folder is present which contains all codes that change with respect to the architecture. This helps a developer to easily adapt a port to a different architecture. Memory management folder contains all files related to memory management. Finally a demo folder that contains demo files for the particular architecture. Due to this efficient code organization, developers can easily familiarize with the OS and make changes to it.

6.4 Queue Storage

In the normal scheduling algorithm, queue stores only pointer to the data, but in FreeRTOS, queue can either store the pointer or the data directly. Storing the data directly results in faster access thus improving efficiency.

6.5 Architecture specific code

In normal scheduling algorithm, architecture specific code is written mostly by the developer. If there is a change in hardware, the entire hardware specific code needs to be rewritten whereas in FreeRTOS, code is already ported to almost 30 architectures. Hence the developer can concentrate more on the application specific code.

6.6 Scheduling algorithm

In a normal scheduling algorithm, the only practical algorithm is round robin. The scheduler does not have any additional code to maintain concurrency or to implement context switching whereas in FreeRTOS, priority based algorithm is used. A higher task is never kept waiting. As soon as a higher task is ready, the currently running task is preempted, its context saved, and control is passed to the higher priority process. FreeRTOS also implements additional OS features such as mutex, semaphore, blocking etc so as to ensure proper concurrency.

6.7 Interrupt execution

In a normal scheduling algorithm, interrupt is executed within a single code. Until the interrupt code completes execution, no other task or interrupt can interrupt it. This is highly inefficient whereas in FreeRTOS interrupt is executed in two half's. A top half and a bottom half. All the immediate functions should be implemented in top half. While the top half is executing, no other task or interrupt can interrupt it. Once the top half finishes, OS will again go for context switch. It will check if any other higher priority task has woken up and passes control to it. If no higher priority task is currently alive, then control passes to the bottom half of the currently executed interrupt. The bottom half executes just like any other normal task. Any interrupt or a higher priority task can preempt it at any time. The bottom half executes with the priority of timer. Thus all the immediate functions in top half are executed as soon as interrupt occurs and the rest of the functions are deferred in bottom half.

Table 1: Comparison of normal scheduling algorithm and FreeRTOS

Feature	**Scheduling algorithm**	**FreeRTOS**
Code organization	Scheduler code, header files	Scheduler code, header files, portable files, memory Files, Demo files
Queue storage	Only pointer	Pointer or data
Architecture specific Code	Developers responsibility	Already ported
Algorithm	Round robin	Priority, Round robin, hybrid
Interrupt	Single code	Top half, Bottom half

7. THREAD METRIC

Thread Metric [7] is a free source benchmark suite for measuring RTOS performance. It is freely available from Express Logic. The test suite consists of 8 distinct RTOS tests that are designed to highlight commonly used aspects of an RTOS. The test measures the total number of RTOS events that can be processed during a specific timer interval. A 30 second time interval is recommended. The porting layer is defined in tm porting layer. This file contains shell services of the generic RTOS services used by the actual tests. The shell services provide the mapping between the tests and the underlying RTOS. The Thread-Metric source code may be freely used, providing its use complies with the stated copyright banner in each source file. For simplicity, each test is defined in its own file. Furthermore, each test is written in a generic C fashion. Performance of Express Logics ThreadX RTOS is provided for reference[8]. Following are the performance tests.

Table 2: Performance Comparison of Thread X and Free RTOS

TEST	Thread X (No of iterations)	Free RTOS (No of iterations)	Free RTOS optimized (No of iterations)
Basic processing	11842	18642	
Preemptive context switch	579190	300001	920712
Message processing	856342	300462	
Synchronization processing	1766942	1858888	
Memory allocation	1501724	3159499	
Interrupt handling	908127	1102823	
Interrupt preemption	358460	602812	

The hardware used for Thread X is ARM Versatile Board (ARM926 Processor, 200MHz). The results are provided by Express Logic for reference purpose [7]. The hardware used for Free RTOS is MSP430F5529 launch pad with a speed of 25MHz. The table represents total no of iterations executed for each test in 30 seconds. A higher value indicates larger number of iterations. Larger number of iterations in turn indicates higher efficiency.

For preemptive context switch, number of iterations can be increased by setting configIDLEHOOK to 0. By setting configIDLEHOOK to 0, idle task hook is disabled. An idle task hook is a function that is called during each cycle of the idle task. Hooking is used for many purposes including debugging and extending functionality. In preemptive context switch test, one of the five tasks is in idle priority. In this scenario, if configIDLEHOOK is set to 1, both the idle hook function and the task with idle priority will time slice in round robin fashion. Thus the share of processor time available for idle task is reduced and the number of iterations falls down rapidly. If configIDLEHOOK is set to 0, idle task hook is disabled. Hence the task with idle priority will get an equal share of time slice as the other four tasks thereby increasing the number of iterations.

8. TRNSACTION OF STACKS

Stack support varies from one RTOS to another. To ensure completion of project with the deadline, a developer must ensure that the RTOS chosen can support all the protocols or stacks needed by the project. For study purpose, transition of bluetooth stack to FreeRTOS is considered. Out of the number of bluetooth protocols available Bluetopia was considered. Bluetopia is the core product of the company Stonestreet One. It is a proprietary protocol hence source code is not available. The company provides the protocol integrated with a scheduling algorithm by default. The application code needs to be added to the scheduling algorithm. To integrate bluetooth with FreeRTOS, source code is needed which is not available. Stonestreet do not provide any stack support for FreeRTOS. FreeRTOS does not have any other in built bluetooth support. Hence implementation of bluetooth protocol in FreeRTOS is a bottleneck. The only solution is for the developer to develop a bluetooth protocol by himself which would be very time consuming. This truly reflects the difficulties a programmer would face if he chooses an RTOS without giving attention to the requirement of the project.

9. RUNTIME STATISTICS

Measurement of time is an important functionality in every OS [9]. By default Free RTOS port uses a timer that is sourced by 32768Hz (ACLK). An interrupt is generated every time the counter counts up to 32. Thus measurement of time takes place in milliseconds. To improve the accuracy, measurement of time can be made in micro seconds. To enable this, the speed of the processor is increased to 25Mhz (MCLK AND SCLK). A timer is sourced by the same frequency (SCLK) and interrupt is generated every time it counts up to 25. Thus time can be measured in microseconds.

10. CONCLUSIONS AND FUTURE WORK

Free RTOS is an OS with a simple structure, small footprint and easy to port. It’s free software license ensures easy availability. License structure also attracts commercial applications as the exception in GPL allows application code to be proprietary. Compared to a normal scheduling algorithm, it has much additional functionality supported by various macros. It provides all the positive attributes of an OS without any constraints in size. This feature is best suited for small factor embedded systems. The responsibilities of the programmer include prevention of starvation and proper manipulation of idle task. Idle task should always be given a proper amount of processor time. Care should be also taken to ensure that handlers to task are deleted as soon as task finishes completion.

RTOS is an area which has large scope for research work. In case of FreeRTOS in particular, the performance of FreeRTOS can be compared with other popular RTOS. By going deeper into the scheduler code, we can check how hard is the OS and whether it can be scaled up to a hard RTOS. Like bluetooth, compatibility of other protocols with FreeRTOS can also be checked.

References

[1] Nandana V., Jithendran A. and Shreelekshmi R., ``Survey on RTOS: Evolution, Types and Current Research'', International Journal of Computer Applications, Vol. 121, July 2015.

[2] A. Bonarini, M. Matteucci and M. Migliavacca,``R2P: An open source hardware and software modular approach to robot prototyping", Robotics and autonomous systems, Elsevier, 2013.

[3] A. Aguiar, S.J. Filho, F. Magalhaes and F. Hessel, ``On the design space exploration through the Hellfire framework", Journals of system architecture, Elsevier, Vol. 60, pp.94-107, 2014.

[4] D. Jensel, ``Adventures in embedded development", IEEE, Vol.11, Issue 6, pp.116-118, Nov 1994

[5] www.freertos.org

[6] www.ti.com

[7] rtos.com/PDFs/MeasuringRTOSPerformance.pdf

[8] rtos.com/news/detail/Express_Logics_ThreadXMCU_RTOS_Scores_Top_Marks_in_Microchip_Technologys_PIC24_Benchmarks/

[9] P. Kumar and M. Srivastava, ``Predictive strategies for low-power RTOS scheduling", Computer Design, International Conference, IEEE, pp.343-348, 2000