Operating System Performance Analyzer for Low-End Embedded Systems

Shahzada Khayyam Nisar[†], *Maqsood Ahmed*[††], *Huma Ayub*[†], *and Iram Baig*[††]

[†]*Department of Software Engineering, University of Engineering & Technology, Taxila 47050 –Pakistan*
[††] *Department of Computer Engineering, University of Engineering & Technology, Taxila 47050 –Pakistan*

Abstract: RTOS provides a number of services to an embedded system designs such as case management, memory management, and Resource Management to build a program.
Choosing the best OS for an embedded system is based on the available OS for system designers and their previous knowledge and experience. This can cause an imbalance between the OS and embedded systems.
RTOS performance analysis is critical in the design and integration of embedded software to ensure that limits the application meet at runtime. To select an appropriate operating system for an embedded system for a particular application, the OS services to be analyzed. These OS services are identified by parameters to establish performance metrics. Performance Metrics selected include context switching, Preemption time and interrupt latency. Performance Metrics are analyzed to choose the right OS for an embedded system for a particular application.

Key Terms: *Embedded Systems, Metrics Number, Performance Analysis, RTOS.*

1. Real-time Operating Systems

An operating system is said to be real time when it schedules the execution of programs in time, handles system resources and gives a reliable basis for the development of software code. [1][2]

1.1 Components of RTOS

Most RTOS kernels consist of the following components:

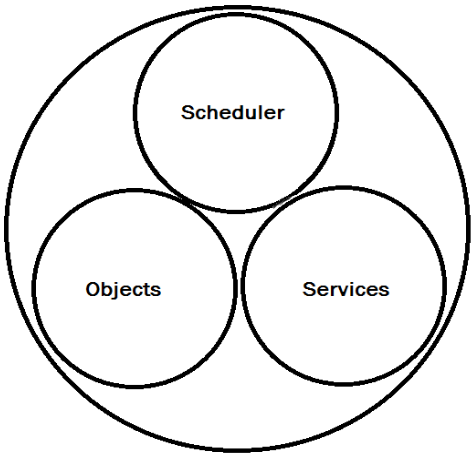

Figure 1: The normal component of the RTOS

i. Scheduler
ii. Objects
iii. Services

1.1.1 Scheduler

Scheduler is at the center of each kernel. A scheduler allows algorithms that are needed to determine what role do when.

1.1.2 Objects

The most common RTOS kernel objects are:
• *Information* --- is simultaneous and independent threads of execution that can compete for CPU execution time.
• *Semaphores* --- is a token-like object that can be raised or charged by information for synchronization or mutual exclusion.
• *Message Queues* --- are buffers that data structures that can be used, mutual exclusion, synchronization and communication by sending messages between tasks. [3]

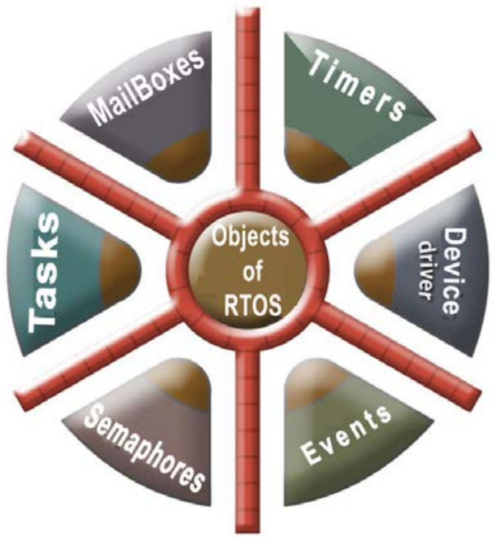

Figure 2: The Objects of the RTOS

1.1.3 Services

Most kernels provide services to assist developers for creation of real-time embedded applications. These services comprise of API calls that can be used to perform operations on kernel objects and can be used in general to facilitate the following services:

- *Timer Management*
- *Interrupt Handling*
- *Device I / O*
- *Memory Management*

Embedded systems are used for different applications. These applications can be proactive or reactive, depends on the interface requirements, scalability, connectivity, etc. Selecting OS for an embedded system is based on an analysis of the operating system itself and the requirements of the application. [4]

2. Embedded Systems:

Embedded systems for a particular purpose are strictly monitored by the device consists of its inclusion. Embedded systems have specific requirements and pre-defined tasks unlike general purpose personal computers.
Embedded systems are programmed hardware devices. A programmable hardware chip the 'raw material' is programmed for specific applications. This is understood in comparison to older systems with hardware or systems fully functional hardware and general purpose software loaded externally. Embedded systems are a combination of hardware and software that facilitates the mass production and variety of applications. [5]

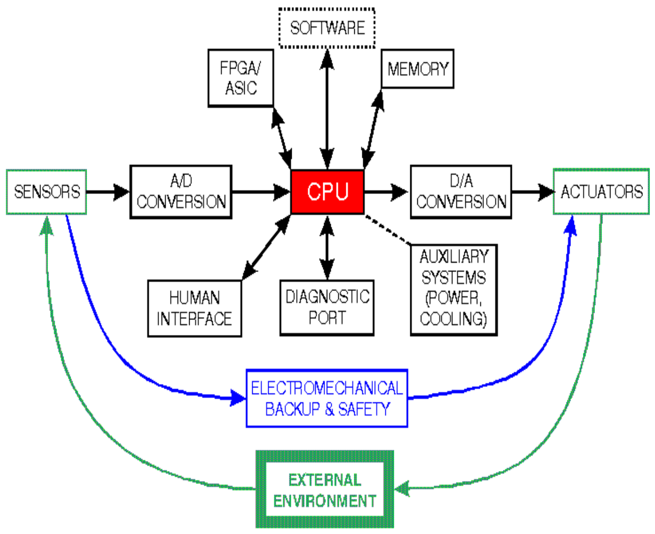

Figure 3: Schematic Embedded System

3. Selected Performance Measures for RTOS

There is a set of performance parameters that are used to analyze an operating system.

In this research, Performance Metrics consists of the following features:

i. Context Switching
ii. Preemption Time
iii. Interrupt Latency

3.1 Context Switching

It is the average time the system takes to switch between two independent active (i.e. not suspended) tasks of equal priority. Task switching is synchronous and non-anticipatory implements real-time control software for some time for slice algorithm multiplexing equal priority tasks. [6]

Task switching is fundamental performance measure of a multitasking system. Measurement attempts to assess the efficiency. The executive manipulates data structures in saving and restoring context. Data exchange is also affected by the host CPU architecture, instructions and functions.

In addition, the task is to change a measure of the manager's competence list management, as an executive normally organize their data structures in ordered lists and mixes nodes depending on the circumstances.

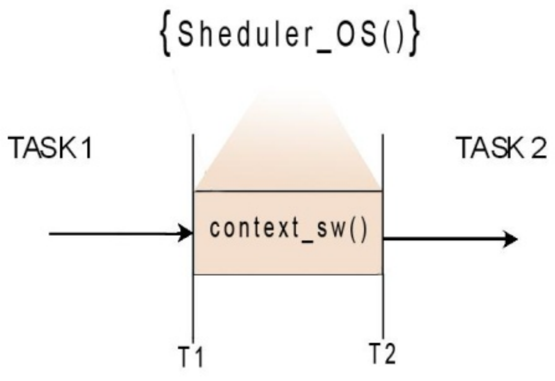

Figure 4: Context Switch Time

3.2 Preemption Time

It is the average time for a task of higher priority to wrest control of the system produces a running task a lower priority. Preemption usually occurs when the higher priority task is related to a sleep mode to a ready state in response to some external event such as when a connected device generates an interrupt, the ISR effort to wake up to the task to service the request. Preemption Time is the average time it takes the President to recognize an external event and switch control of the system produces a running lower priority task to an idle task with higher priority. [7]

Although conceptually similar to the task switch, takes first refusal usually longer. This is because the executive must first recognize waking measures and assess the relative priorities started and asked details and only then change position if necessary. Virtually all multi-use / multitasking executives assign task priorities and many allow the program designer priorities change dynamically. For this reason, together with the interrupt latency preemption is the most important real-time performance parameter. [9]

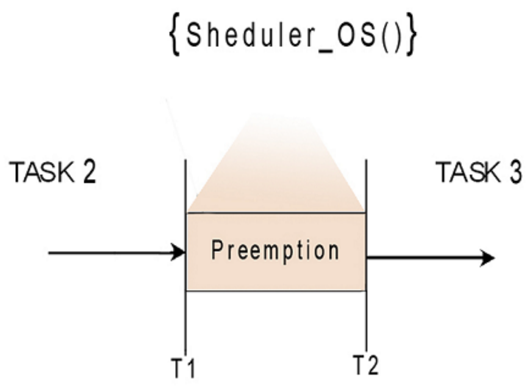

Figure 5: Preemption Time

3.3 Interrupt Latency

It is the time between the CPU receiving an interrupt request and the implementation of the first instruction in interrupt service routine. Interrupt latency reflects only the delay introduced by the executive and the processor and does not include delays on the bus or external devices. [8]

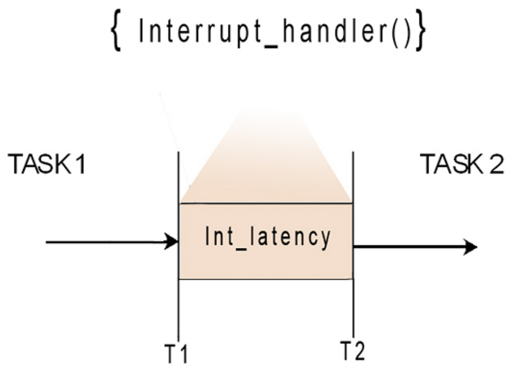

Figure 6: Interrupt Latency

4. Aims & Objectives

To choose the right operating system for an embedded system selected performance metrics are analyzed. The performance parameters are actually related to the services provided by the operating system. Improving the services provided is better operating system. In this research a Metrics Number is generated for the grade of the operating system by measuring time required for each service to occur and the number of times the service units in a complete loop. That Metrics code helps to choose the right operating system for an embedded system for a particular application.

5. Research Platform

To identify and analyze performance parameters, the environment of software and hardware is created. Three RTOS for embedded systems have been selected.

- **(i)** SALVO
- **(ii)** PICOS18
- **(iii)** FREERTOS

These RTOS have free version available and also used same compiler, simulator, language and hardware platform.

6. How one RTOS differs from the other?

- **(i)** RTOS' differ in main architechure.
- **(ii)** Type of scheduling algorithm used in it. (Pre-emptive scheduling or co-operative scheduling).
- **(iii)** Number of instructions of kernel without any task written to it formed after compilation of complete code. It will ultimately occupy space in ROM and RAM, so it effects memory and execution speed .
- **(iv)** Number of tasks it can run without degrading the performace like response time. [9]
- **(v)** Performance metrices that we have choosen i.e. Context switching Time, Preemption time and Interrupt Latency.

Applications are generated to perform multitasking. The application is analyzed in real time and monitored to extract the desired results.

In testing the performance of RTOS on 8 bit microcontroller we choose the most commonly used microcontroller family, microchip PIC 18Fxxxx class. And created a scenario in which maximum number of hardware module connected with it. The PIC18 family is being used in tremendous marketable products. So in order to check its efficiency in managing the modules controlled through the RTOS a comparison is done between different RTOS running on the same platform i.e. PIC18f452 / PIC18F4620 microcontroller, hardware modules and MPLAB simulator and PIC-C18 Compiler [10].

These modules consisting of Temperature sensor, Real Time CLOCK, UART Communication, LCD and Keypad user interfaces. This allows the 8 bit microcontroller to test the effective utilization of these resources mostly when modules are used in parallel under RTOS.

Real Time Clock and keypad working on interrupt may have same or different priorities. As this PIC range can support up to two level of interrupt handling. TIME is updated on LCD after each second through interrupt and displayed on LCD. On keypad when key is pressed interrupt is generated; shows button is pressed on LCD. Temperature sensor module and SERIAL UART running in parallel displaying data on LCD.

On hardware level the notable parameters that affect the working of RTOS in handling different modules are as follow:

7. Microcontroller

- **(i)** Processor Clock Speed determines execution speed of RTOS.
- **(ii)** Amount of ROM available specify the number of instructions can be stored including RTOS and task instructions.
- **(iii)** RAM size allows the number of processes that can be run for given RTOS. [11]

(iv) STACK size provides the process local variable to accommodate.

(v) Timer used in interrupt handling and in calculating task execution time during multitasking. Timer depends upon the clock speed. [12]

(vi) That microcontroller which has no cache has RTOS architecture which is free from cache at design time.

8. Compiler and Coding Benefits

(i) For efficiency most tasks to be written in assembly.

(ii) If C language is used then optimized then good and efficient compiler that maps the C language to assembly in minimum no. of instruction is the ultimate goal.

(iii) Optimized coding technique used in defining tasks. [13]

9. Hardware Information

(i) PIC18F452 has Harvard architecture (has a separate instruction and data bus).

(ii) 40 pin IC.

(iii) 1536 BYTES on chip RAM.

(iv) 32KBYTES FLASH memory for Program storage.

(v) Maximum 16384 single word instructions can be placed in FLASH memory.

(vi) 2 interrupt priority levels.

(vii) 8MHZ internal oscillator or up to 20 MHz external oscillator can be used.

Block Diagram

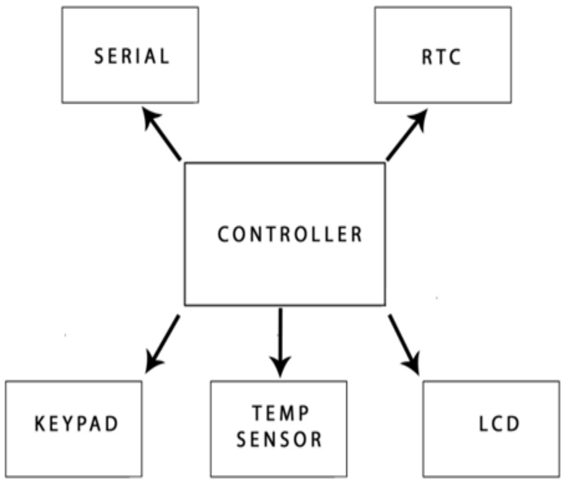

Figure 7: Block Diagram of Hardware

10. Analysis

As the application comprises of multi tasking and interrupts therefore, the Performance Metrics selected comprises of three performance parameters. These are

1. *Context Switch time*
2. *Preemption time*
3. *Interrupt Latency*

For measuring the Metrics Number there are two requirements

1. *Time "T" for each parameter*
2. *Number of times "W" the parameter is called*

11. Calculation of Time for Each Parameter

In order to calculate the time required for each parameter it is necessary to identify where these parameters are called. To calculate the time hardware timers are used. Break points are given to the entry and exit of a parameter. Timer is initialized on the entry break point and terminated on the exit. The time calculated gives us the time for a parameter. Same procedure is followed for rest of parameters.

For Example: The Context Switch Time is calculated by marking Breakpoints on the start and end of the context switch service of the operating system. Hardware Timer is used for calculating the time. Timer is initialized at the start break point and terminated at the exit break point. This will give us the time required by the operating system for context switch for two tasks.

12. Calculation of Weight of Each Parameter

To find out the number of times the parameter is called a variable is defined in each parameter. For each time the parameter is called that variable is incremented. The incremented value is displayed on the Display screen. For each parameter that variable is defined and measured for one complete iteration of the application.

RTOS was made working in this WAY For testing of performance:

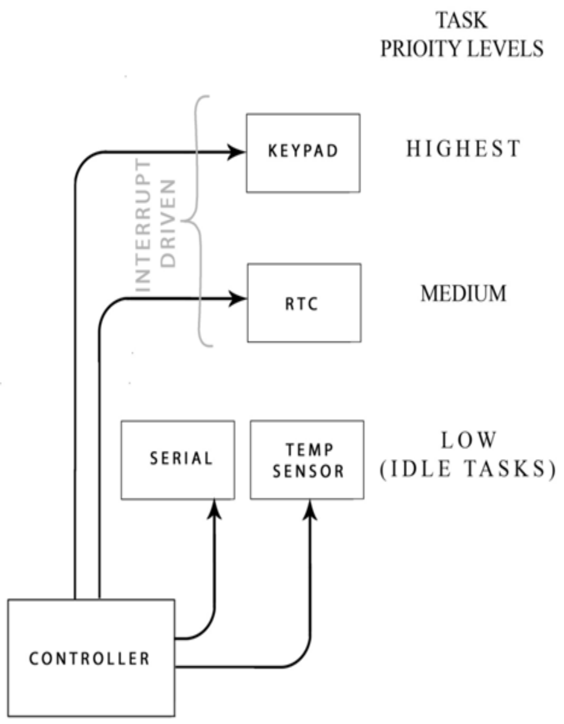

Figure 8: RTOS Scheduling Policy

To find the time necessary to provide the required service timers are initiated at the beginning and end when the loop is complete. This will allow time an operating system to provide the service that is their own ability. A variable is initialized for each service for which time is expected to learn that the number of times that the service has been called. This will give the following values:

Time T are the observed values of Performance Parameters in microseconds. Number of times the performance parameter occur is given by the **Weight W,** the product of T and W is given by **TxW** and total sum of all the three product is given by **ΣTxW** in microsecond.

13. Metrics Number

For understanding we did comparison by analyzing the real functionalities / usage of context switching, preemption and interrupt latency in applications and compared accordingly.

14. Generation of Metrics Number for Free RTOS

- ***Context Switching*** is measured by using yield() functions. It means to give control to other task when both tasks have the same priorities. Execution time of task yield() has to be considered because it is the part of kernel or scheduler.

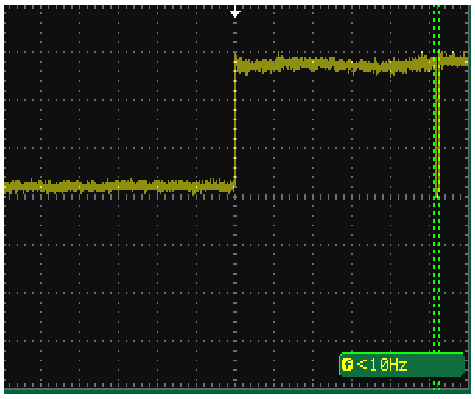

Figure 9: Context switching for Free RTOS

- ***PreemptionTtime*** is measured when task priority change or giving execution time to higher priority task by changing the task priority to higher level. we also include the time of maketask_priorityhigh() function.

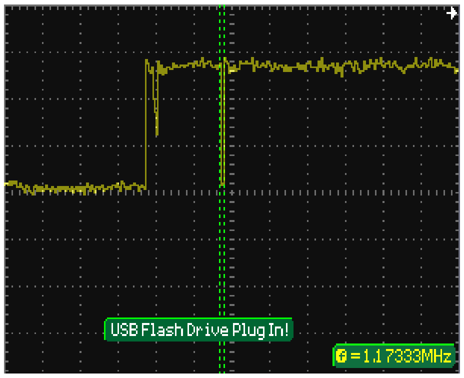

Figure 10: Preemption Time for Free RTOS

- ***Interrupt Latency*** is the measured time between when external interrupt came and ISR related to that interrupt start executing.

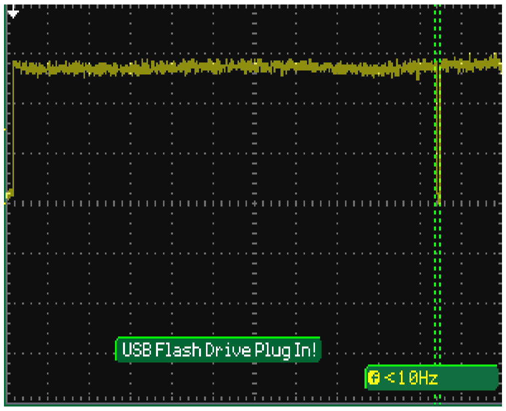

Figure 11: Interrupt Latency for Free RTOS

Table 1: Generation of Metrics Number for FREE RTOS

Performance Parameter	**Time T (μ sec)**	**Weight W**	**T X W**	**ΣTxW (μ sec)**	**METRICS NUMBER 1/ ΣTxW**
Context Switching Time	T_{CS} =7	W_{CS} = 55	385	5359	**186.60**
Preemption Time	T_P =15	W_P = 67	1005		
Interrupt Latency	T_{IL}= 3.5	W_{IL}= 1134	3969		

Generation of Metrics Number for PICOS18

- ***Context Switching*** is measured by then each time the task 0 sends an event to the task 1 the time to switch from task 1 to task 0.

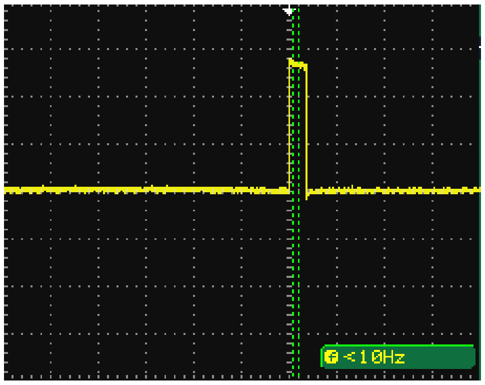

Figure 12: Context switching for PICOS18

- ***Preemption Time*** is measured when task priority change or giving execution time to higher priority task by changing the task priority to higher level.

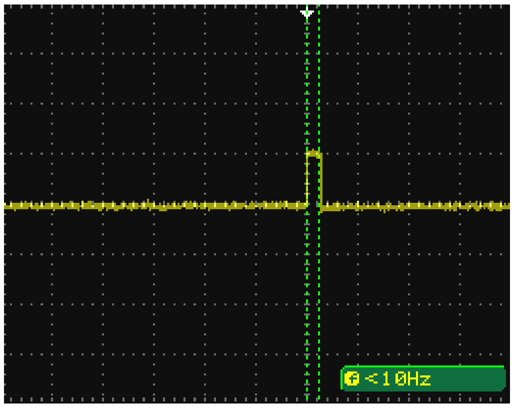

Figure 13: Preemption Time for PICOS18

- ***Interrupt Latency*** is the measured time between when external interrupt came and ISR related to that interrupt start executing.

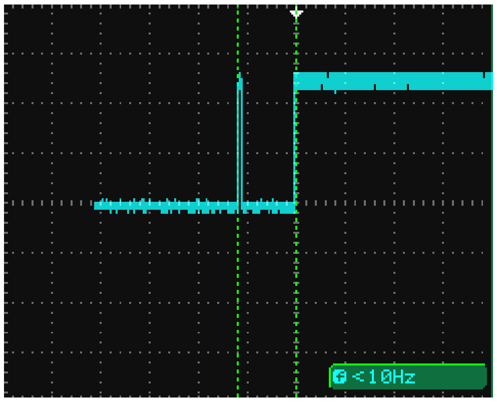

Figure 14: Interrupt Latency for PICOS18

Table 2: Generation of Metrics Number for PICOS18

Performance Parameter	**Time T (µ sec)**	**Weight W**	**T X W**	**ΣTxW (µ sec)**	**METRICS NUMBER 1/ ΣTxW**
Context Switching Time	TCS =47	WCS = 55	2585	60290	**16.58**
Preemption Time	TP =15	WP = 67	1005		
Interrupt Latency	TIL= 50	WIL= 1134	56700		

15. Generation of Metrics Number for SALVO RTOS

Co-operative context switching depends on the task that is currently running. The current task calls for other to switch to other for its working. But in preemptive context switching the scheduler do not take care of the running task when higher priority task occurs. SALVO is the only RTOS that is not preemptive but co-operative RTOS. There are upto 15 levels of priorities.

- ***Context switching*** is measured by Using OS_Yield() functions. Its mean giving control to other task when both task have the same priorities. Execution time of task yield() has to be considered because it's part of kernel or scheduler.

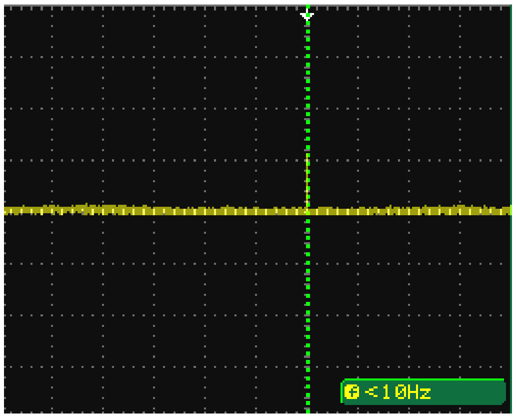

Figure 15: Context switching for SALVO RTOS

- ***Preemption time*** is measured when task priority change or giving execution time to higher priority task by changing the task priority to higher level. We also include the time of OS_SetPrio() function.

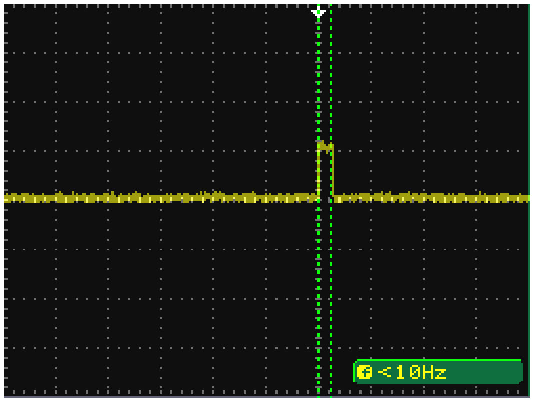

Figure 16: Preemption Time for SALVO RTOS

- ***Interrupt latency*** measured the time between when external interrupt came and ISR related to that interrupt start executing.

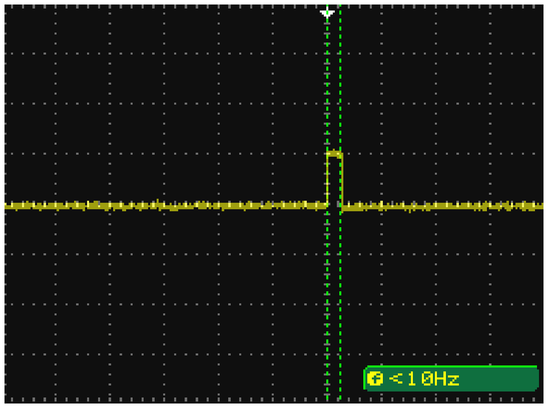

Figure 17: Interrupt Latency for SALVO RTOS

Table 3: Generation of Metrics Number for SALVO RTOS

Performance Parameter	**Time T (µ sec)**	**Weight W**	**T X W**	**ΣTxW (µ sec)**	**METRICS NUMBER 1/ ΣT xW**
Context Switching Time	T_{CS} =10	W_{CS} = 55	510	4149	**241.02**
Preemption Time	T_P =12	W_P = 67	804		
Interrupt Latency	T_{IL}= 2.5	W_{IL}= 1134	2835		

In Tables, Column 1 lists performance parameters, for which the Metrics numbers must be generated. In column 2, the time T required for each performance parameter in micro-seconds is listed. "T" is the measurement of time intervals between time initialization and termination for each performance parameter. Column 3, the number of times the performance parameter occurred is denoted by "W". "W" is the weight given to each performance parameter for that specific application. Column 4 is the weighted value of each performance parameter. Column 5 is the summation of all weighted measurements. The inverse of the sum of the weight measurements gives us a number Metrics. *Larger Metrics numbers are better operating system for that specific application*.

16. Conclusion

The measurement of this Metrics Number has a great significance in selection of right operating system for a specific application. If another operating system is selected and same application is used with same hardware then the weights will remain same however the time observed for each performance parameter will be different. If the resulting Metrics number is greater, then this operating system is best for that environment. The Metrics Number generated will help us in rating the operating system. This will help us in deducing a procedure for selecting the right Performance Metrics. Having right performance metrics will help us to calculate metrics number which will help us in selecting right operating system for an embedded system for a specific application.

17. Future Directions

In this paper a method to analyse performance metrics of operating system in real-time embedded systems is described. For future work it is recommended that if the methodology for the application processing time is formulated then right processor can also be selected for the embedded system. This will help the designer to make an efficient embedded system with a right Real Time Operating System.

References

[1] Wei-Tsun Sun; Zoran Salcic; , "Modeling RTOS for Reactive Embedded Systems," *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on* , pp.534-539, Jan.2007 doi:10.1109/VLSID.2007.111

[2] Su-Lim Tan; Tran Nguyen Bao Anh; , "Real-time operating system (RTOS) for small (16-bit) microcontroller," *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on* , pp.1007-1011,25-28 May 2009 doi: 10.1109/ISCE.2009.5156833

[3] Baynes, K.; Collins, C.; Fiterman, E.; Brinda Ganesh; Kohout, P.; Smit, C.; Zhang, T.; Jacob, B.; , "The performance and energy consumption of embedded real-time operating systems," *Computers, IEEE Transactions on* , vol.52, no.11, pp. 1454- 1469, Nov. 2003 doi: 10.1109/TC.2003.1244943

[4] Hessel, F.; da Rosa, V.M.; Reis, I.M.; Planner, R.; Marcon, C.A.M.; Susin, A.A.; , "Abstract RTOS modeling for embedded systems," *Rapid System Prototyping, 2004. Proceedings. 15th IEEE International Workshop*, pp. 210- 216, 28-30 June 2004 doi:10.1109/IWRSP.2004.1311119

[5] He, Z.; Mok, A.; Peng, C.; , "Timed RTOS modeling for embedded system design," *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE* , pp. 448- 457, 7-10 March 2005 doi:10.1109/RTAS.2005.52

[6] Suk-Hyun Seo; Sang-won Lee; Sung-Ho Hwang; Jae Wook Jeon; , "Analysis of Task Switching Time of ECU Embedded System ported to OSEK(RTOS),"*SICE-ICASE,2006. International Joint Conference* , pp. 545-549,
18-21 Oct. 2006 doi: 10.1109/SICE.2006.315544

[7] Kavi, Krishna; Akl, Robert; Hurson, Ali; “Real-Time Systems: An Introduction and the State-of-the-Art” *John Wiley & Sons, Inc. Wiley Encyclopedia of Computer Science and Engineering;* SN: 9780470050118; 2007;
doi: 10.1002/9780470050118.ecse344

[8] El-Haik, Basem; Shaout, Adnan; “Design Process of Real-Time Operating Systems (RTOS)” *John Wiley & Sons, Inc. Software Design for Six Sigma;* pp. 56-76; SN: 9780470877845; 2010; doi: 10.1002/9780470877845.ch3

[9] Edwards, Stephen A.; “Real-Time Embedded Software”; *John Wiley & Sons, Inc.; Wiley Encyclopedia of Electrical and*

Electronics Engineering; SN: 9780471346081; 2001; doi: 10.1002/047134608X.W8113

[10] Weiss, K.; Steckstor, T.; Rosenstiel, W.; , "Performance analysis of a RTOS by emulation of an embedded system ," *Rapid System Prototyping, 1999. IEEE International Workshop on* , pp.146-151, Jul 1999 doi:10.1109/IWRSP.1999.779045

[11] Stepner, D.; Rajan, N.; Hui, D.; , "Embedded application design using a real-time OS," *Design Automation Conference, 1999. Proceedings. 36th* , pp. 151-156, 1999 doi:10.1109/DAC.1999.781301

[12] Elsir, M.T.; Sebastian, P.; Yap, V.V.; , "A RTOS for educational purposes," *Intelligent and Advanced Systems (ICIAS), 2010 International Conference on* , pp.1-4, 15-17 June 2010 doi:10.1109/ICIAS.2010.5716166

[13] Cena G., Cesarato R., Bertolotti I.C. "An RTOS-based design for inexpensive distributed embedded system" (2010) *IEEE International Symposium on Industrial Electronics*, art. no. 5636340, pp. 1716-1721.