Development and Performance Analysis of on-Board Computer Software for Real Time Auto-Pilot Control System

E. Mohamed, H. Taha, S. A. Shedeid, K. Shehata

Abstract: On board computer (OBC) is an essential component of any On-Board Control System that involves several tasks that needs to be executed in precise order and for accurate periods of time. Most of the real time control systems require instantaneous execution of commands either through hardware or software. To ensure that these requirements are fulfilled, it is necessary to measure not only the execution time of individual tasks, but also establish the overall real time performance of the system as well. These measurements may then be used as a basis for accurate real time scheduling analysis and to identify timing problems or to spot which code segment needs to be optimized. The most common reasons for measuring execution times are to refine, estimates, optimize code, analyze real time performance, and to debug timing errors. Oscilloscope is one of the best tools for accurately measuring execution time with nano-second's result, especially when accurate timing is critical. This method requires extra hardware features, Oscilloscope analysis features, and additional software code segments. Depending on the real time operating system (RTOS) and hardware, it is possible that code is not executing at the proper rate or during the right period as specified by the designer. These drawbacks may result due to limitations in the operating system, the application software or the hardware used. This proposal presents hyper threads, a unifying programming model for specifying application threads running within a hybrid CPU/FPGA system that is used to control the real time operation of an auto-pilot control system.

Nomenclature

ARINC	Aeronautical Radio, Incorporated
ADC	Analog to Digital Converter
ASIC	Application-Specific Integrated Circuit
FDM	Flight Dynamic Model
FPGA	Field Programmable Gate Array
SoC	System on a Chip
SBC	Single Board Computer
OBC	On Board Computer
ISR	Interrupt Service Routine
PID	Proportional Integral Differential
RTOS	Real Time Operating System
RMA	Rate Monotonic Analysis

1. Introduction

In this paper, a proposal for developing an on-Board computer software for real time auto-pilot control system based on incorporating an FPGA chip with a CPU is introduced. Schedulability of the proposed architecture for real time applications is evaluated using two performance indices; Basic RMA test and Extended RAM tests [1].

First, a brief introduction about RTOS is introduced in the next section. Then, the two performance indices used in analyzing the Schedulability of the proposed system under real time operating conditions are presented. Afterward, a brief summery about the software developing tools used in this work is outlined.

In section 2, a description of the developed Real Time Auto-Pilot Control System specifications is introduced along with some of the tasks managed by the On Board Computer

operating System. Then, in section 3, hardware specifications in addition to communication and interfacing protocols are presented.
In section 4, two proposals of the real time control system are introduced under two different operating systems, MS-DOS and RT/Linux. Performance of the two proposals is evaluated and a comparison between the two algorithms is introduced supported by the experimental results. Finally the conclusion of this work is given in section 5.

1.1 Operating Systems

One of the major problems that takes place in real time operating system is the deviation of task execution time from its planned value. For example, a task which is supposed to be executed every 10 msec may be detected to be executing every 20 msec or every 5 msec. This type of error is so complicated and hard to observe through any method other than a timing analysis. A task that is running too fast might use critical CPU resources, and causing other tasks to fail to meet timing constrains. On the other hand if the execution period is slower than expected, the performance of the system might be degrading in control system and this would result in a reduced accuracy of the application. This problem may appear in the MS-DOS operating system with the multitasking and memory management. Linux and RT/Linux context switch are slower than other operating system as VxWorks and QNX but they are closed source and very expensive. Linux and RT/Linux exhibit the highest operation throughput for all data around 10,000 operations / sec more than other operating systems.

A real time operating system (RTOS) is multitasking operating system that is intended for real time applications. Such applications include embedded systems, mobile phones software, Robot control, aircraft, missiles, space shuttle, and satellites. The heart of any real time OS is the kernel. The kernel is the central core of any operating system, and it takes care of all the OS jobs as:

- Booting
- Task Scheduling
- Standard Function Libraries

RTOS is an operating system kernel that supports task scheduling, task dispatching and inter task communication. In order to achieve these objectives, real time kernel design follows one of the following strategies [2]:

- Polled loop system
- Interrupt driven system
- Multi tasking
- Foreground/ background system
- Full features RTOS

Commonly, On-board hardware consists of sensors, a computer with interfacing electronics, a battery, and a power distribution module. All mechanical control actuation is accomplished with standard RC servos. A Single Board Computer is plugged into a backplane to provide for I/O cards.

A common configuration for very-high-volume embedded systems is a system on a chip (SoC) which contains a complete system consisting of multiple processors, multipliers, caches and interfaces on a single chip. SoCs can be implemented as an ASIC or using a FPGA.

1.2 Schedulability Analysis / Rate Monotonic Analysis

Analysis of the schedulability for a system tasks plays an important role for real time concepts. After an embedded application has been decomposed into ISRs and tasks, the tasks must be scheduled to run in order so that they perform the required system functionality. Schedulability analysis determines if all tasks can be scheduled to run and meet their deadlines based on the deployed scheduling algorithm while still achieving optimal processor

utilization. The schedulability analysis only examines how systems meet temporal requirements, not functional requirements.

Functional requirements for real-time systems are commonly analyzed using the RMA method. In this method, the model is developed over a scheduling mechanism called Rate Monotonic Scheduling; RMS, which is the preemptive scheduling algorithm with rate monotonic priority assignment as the task priority assignment policy. Rate monotonic priority assignment is the method of assigning a task its priority as a monotonic function of the execution rate of that task. In other words, the shorter the period between each execution, the higher the priority assigned to a task.

RMA is usually evaluated using one of two tests; Basic RMA Schedulability test and Extended RAM Schedulalibility test.

1.2.1 Basic RMA Schedulability Test

Basic Schedulability RMA is associated with set of assumptions [4]. These assumptions are:

1- All the tasks are periodic,
2- Tasks are independent of each other and that no interactions occur among tasks.
3- A task's deadline is the beginning of its next period.
4- Each task has a constant execution time that does not vary over time,
5- All tasks have the same level of criticality.

Aperiodic tasks are limited to initialization and failure recovery work and that these aperiodic tasks do not have hard deadlines.
This Equation is used to perform the basic RMA schedulability test on a system .

$$\left(\frac{C_1}{T_1}+\frac{C_2}{T_2}+\cdots+\frac{C_n}{T_n}\right) \le U(n) = n\left(2^{1/n}-1\right) \qquad (1)$$
$$1 \le i \le n$$

Where:
C_i= worst-case execution time associated with periodic task *i*
T_i= period associated with task *i*
n = number of tasks
U(*n*) is the utilization factor.

The right hand side of the equation is the theoretical processor utilization bound. If the processor utilization for a given set of tasks is less than the theoretical utilization bound, this set of tasks is schedulable. The value of *U* decreases as *n* increases and eventually converges to 69% when *n* becomes infinite.

1.2.2 Extended RAM Schedulability Test

Basic RMA test evaluation suffers from several shortcomings. For example, based on the second assumption, basic RMA is impractical because tasks in real-time systems have inter-dependencies, and task synchronization methods are part of many real-time designs. Task synchronization, however, lies outside the scope of basic RMA.

Deploying inter-task synchronization methods implies some tasks in the system will experience blocking, which is the suspension of task execution because of resource contention. Therefore, the basic RMA is extended to account for task synchronization. Equation provides the equation for the extended RMA schedulability test [5].

$$\left(\frac{C_1}{T_1}+\frac{C_2}{T_2}+\cdots+\frac{C_i}{T_i}+\frac{B_i}{T_i}\right) \le U(i) = i\left(2^{1/i}-1\right) \qquad (2)$$
$$1 \le i \le n$$

Where:
C_i= worst case execution time associated with periodic task *I*

T_i= period associated with task *i.*
B_i= the longest duration of blocking that can be experienced by *I*
n = number of tasks

1.3 Software Developing Tools

Generally, designers of OBC system use compilers, assemblers, and debuggers to develop real time software. However, they may also use some more specific tools based the application. In our case, an FPGA software development package is used to design , simulate and implement a digital control circuit. This circuit is used to control the communication (Arinc-429) between the OBC and other peripherals. In addition, it is also used to handle ADC and DAC conversion, manage the interrupt service routine (ISR) and direct memory access manipulation as well.

In this proposal two implementations for the suggested algorithm are provided. The fist one is a real time control software that works under MS-DOS 6.22 operating system. This implementation is accomplished using the Turbo C++ 3 tool [6]. The second version works under RT/Linux real time operating system. This version is realized using GCC compilers 4.4.3 under the 2.6.28 Linux kernel [7].

2. The Developed Real Time Auto-Pilot Control System

Figure 1 illustrates a general overview of general real time auto-pilot control system. An Auto-pilot control system represents an excellent practical example of a real time control system. Necessity of RTOS in real time autopilot control systems stems from the several tasks which need to be executed periodically in a very precise order within the specified periods. Some of these tasks are; manage the execution of actuators control commands, acquiring navigation sensors data, in addition to the navigation computations, as well. Definitely, a RTOS is required to direct the execution of all those tasks since each task duration and priority is different than the other.

Fig.1 The overall developed real time control

The main task among the jobs of the proposed RTOS is the calculation of the PID controller whose output $\delta(t)$ is given by [8];

$$\delta(t) = K_p \mathrm{e(t)} + K_I \int_0^t \mathrm{e}\,(\tau)\mathrm{d}\tau + K_d \frac{\mathrm{d}}{\mathrm{dt}}\,\mathrm{e}\,(\mathrm{t}) \qquad (3)$$

Where:

K_p, K_I, K_d Gains, tuning parameters

Subjected to the following constraints

- Autopilot-PID controller- calculation period of 10 msec
- Sensors data extraction with variable timing of 10 and 2.5 msec
- Range control calculation with period of 2.5 msec
- GPS data simulation at every 1 sec
- ADC & DAC operation
- Actuators control

On Board Computer (OBC) which has the embedded control algorithm that handles the execution of the previously mentioned tasks consists of the following sub-systems:

- Acceleration and angle measuring sensors.
- FDM simulator which is used for testing.
- Actuation system.
- On board telemetry system.
- Test computer.

3. The Developed on-board Computer (OBC)

The developed OBC is enclosed in a sturdy metal container with the following boards; as shown in figure (2):

a) Single Board computer (SBC)

- Pentium III class 800 MHz, 256 MB RAM, system memory with 100MHz memory bus.
- 512KB 16-bit wide integrated flash memory for BIOS and user programs.
- Communicates externally over the ISA bus, 33MHz PCI Bus and I/O ports.
- Generates on-board RGB video for CRT display systems.
- Plug and play BIOS with IDE auto detection, 32-bit IDE access, and LBA support.
- 256 MB IDE flash disk for program storage.
- Programmable watchdog timer.
- Power supply: 5VDC operation from the PC/104 bus or a power connector.
- Ruggedized for vibration and Extended temperature range operation: -40 to +85°C.

b) A PC-104 backplane board

This board holds the SBC and communicates with other boards through the main backplane.

c) Arinc-429 Communication board

It contains three Arinc-429 chips as a control and drivers for the transmitters and receivers. FPGA chip is mounted on the board to control the ARINC-429 communication with the sensors, on-board telemetry and test computer.

d) DAC and ADC board

It is used to control the command and feed back of the actuation system and FPGA chip is designed and implemented to control the function of the conversion. In addition the control of discrete input and discrete output commands signal. This chip is mounted on that board.

e) A Signal Conditioning board

It adapts the command and the status signals from 27 volt to 5 volt. The control of this function is achieved from the last board.

f) Servo Control Loop board

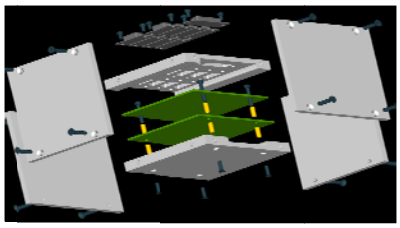
It has the real time closed loop for each actuation channel individually. It controls the commands and the feedback of the actuation system controlled ADC and DAC.

g) Power Supply board

It generates all necessary DC voltages for other boards using DC-DC converter such as 27, 5, 15, -15 volts.

h) Main Backplane board

It is the main board that holds all others boards and transferring signals (power, data, control, status) and its connection. An internal wiring connection between backplane and external connection is implemented.



On board computer as a parts

On board computer after integration

Fig. 2 On board computer hardware architecture.

The proposed On-board computer communicates with and interfaced to the outside world via peripherals, such as:

- ARINC-429,
- RS-232
- ISA and PC-104 bus
- Discrete IO
- Analog to Digital/Digital to Analog (ADC/DAC)

4. Proposed Implementations of Real Time Control Software based on MS-DOS and RT/Linux Operating Systems

According to Quing [2],

4.1 Software Components

The software consists of operating system that has kernel space functions which include the drivers needed to communicate with other devices and user space functions for the application itself. The proposed real time software consists of the following function categories:

a) Real-time nucleus functions:

- CP and OS initialization.
- Program control.
- Task control.
- Memory control.
- Interrupt control (IRQ).
- Direct memory Access (DMA).
- Time service.
- Task synchronization.
- Inter task synchronization.
- Input/output control.
- CP peripheral equipment control.
- File control.

- Read/Write EEPROM support.
- Exclusive situation processing.
- Support of program debugging modes when interacting with workstation (WS) hardware and software.

b) System function library:
- SP calling the functions of real-time nucleus.
- EEPROM access SP: read, write, deletion.
- Standard C/C++ language SP that are system-dependent.
- Other standard SP as part of the system-independent library.
- Of standard SP of the programming system.

c) Functional s/w needed for the devices

4.2 Calculations Management

The designed system is supposed to control and manage the assessment of the following tasks;

4.2.1 Tasks performed on specified time schedule

That set of tasks composes the main calculated part. Two calculation cycles can be emphasized: 400 Hz, 100 Hz. The cyclic tasks can be distributed as follows:

400 Hz cycle:
- Sensor (accelerometers) data sampling
- Velocity integration and range control
- Test of calculation cycle.

100 Hz cycle:
- Navigation algorithm
- Calculation of output parameters (navigation parameters)
- Autopilot calculations.
- Test of a system state.
- Generation of telemetric information

4.2.2 Tasks performed according to events -operation with peripheral equipment

The tasks performed depending on particular events define the logical management of OBC functioning, as well as the interface with external systems.

These tasks include:
- Time service (procession of time marks)
- ARINC-429 receivers and transmitters service
- supplied message (package) coding
- Received message (package) decoding

4.2.3 Tasks composing the cycle of background calculations

Background calculations are the tasks intended to provide monitoring of OBC state. Here again the tasks of the recovery of system operability are fulfilled in different exceptions. Background tasks are the most low-priority calculations which include: functional device test, test task fulfillment (computer monitoring), CP RAM and ROM monitoring and other tasks requiring no strict synchronization with real time or any events.

4.3 Software algorithm and tasks scheduling

The proposed real time control algorithm is illustrated in the flow chart shown in Figure 3.

Fig. 3. The flow chart of the real time OBC S/W control

From a managerial perspective, there are five main tasks which the suggested algorithm should manage through the real time control scheme which are:

Task 1: Autopilot calculations
Task 2: Profile generation
Task 3: Range control calculations using fixed step Runge Kutta 4^{th} order
Task 4: Actuation command control (DAC)
Task 5: Read Feedbacks (ADC)

As previously mentioned, two different implementations are realized on two different operating systems.

The first one shown in figure 4.1 is realized with the GCC compilers 4.4.3 under the 2.6.28 R/T Linux kernel. As shown in figure since R/T Linux is a multi-threaded operating system, the execution of various tasks is accomplished on priority basis regardless of the order of each task.

Fig. 4.1 Tasks Schedule Pseudo Concurrent Execution Single Processor Linux

On the other hand, figure 4.2, illustrates the sequential execution of tasks when the proposed control algorithm is realized under MS-DOS 6.2.2. This demonstrates the advantage gained from using R/T Linux rather than MS-DOS.

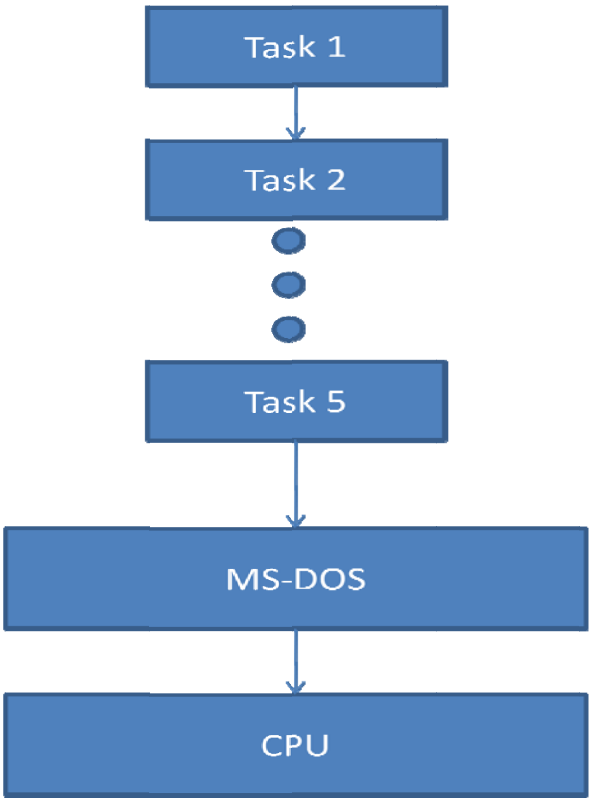


Fig. 4.2 Tasks Schedule Pseudo Concurrent Execution Single Processor MS-DOS based

5. Experimental Results

The implemented real time control software is developed using C language based on two main operating systems: MS-DOS 6.22 and Ubuntu 9.04 kernel 2.6.28 with the GCC compiler 4.3.3, Linux. It's tested using FDM simulator that generates the readings of the sensors in real time. The main objective of this test is to evaluate the execution time of each task and measure the performance of the overall system.

The following two methods were used for measuring the execution time:-

A. Measurement using internal and external hardware with Agilent oscilloscope 1GHZ-4 GHS/S; it is the time elapsed between rising-falling edge of the oscilloscope signal and the rest is the wait period as shown in figure 7 [10]. In addition, a software C code is implemented as:

```
outport (Buffer_Address, 0x0001);
    (required task to measure execution time)
outport (Buffer_Address, 0x0000);
```

B. Measurement of execution time in MS-DOS 6.22 with order of micro-seconds is not available while a real time clock (RTC) function can be provided by the RT/Linux:

```
clock_gettime (CLOCK_REALTIME, Time_arg)
```

Table 1 illustrates the execution time of selected tasks for the real time system. An RMA test is performed to check the schedulability. To insure that the system is always stable, the CPU utilization should be calculated using Equation (1) as such that:

$$5.45\ \% \leq U(5) = 74.35\ \%$$

Moreover, a RAM test is performed as in Equation (2) to check the schedulability in case of dependence between task 1, task 2 and input-output bus resources. A blocking time for task1and task2 with input-output bus B_1, B_2 are measured 15 µSec and 240 µSec simultaneously as illustrated in figure 5. The results are as follows:

$$2.46\ \% \leq U(1) = 100.0\ \%$$

$$0.12\ \% \leq U(2) = 82.84\ \%$$

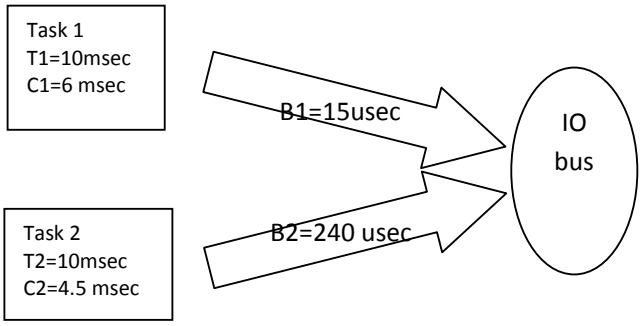


Fig. 5 Blocking time for two tasks using same i/o bus.

Table 1. Tasks periodic in mSec and execution time in µSec

Periodic Task	Period [T]	Execution Time [C]			
		MS-DOS 6.22		RT/LINUX	
Measurements Method		A	B	A	B
Task 1	10	11	N/A	6	5.485
Task 2	10	7	N/A	4.5	3.924
Task 3	2.5	9	N/A	9	8.273
Task 4	10	20	N/A	18	17.150
Task 5	10	490	N/A	480	470.893

Based on the results of Table 1 and the output of Equations 1 and 2 the selected tasks are schedulable. The schedulable algorithm is used in this work is the preemptive priority.
The upper part of Fig 5 shows the frequency of the tasks of 400HZ and the lower shows the tasks of 100HZ running in MS-DOS 6.22 operating system using the oscilloscope. They are based on the occurrence of the ISR handling these tasks. The upper part of Fig 6 shows the frequency of 400HZ tasks served in kernel domain moreover; the lower part presents periodic execution time in user domain. The overall execution time for all tasks under RT/LINUX in user domain is shown also in the lower part of Fig 6. The interrupt response time for the ISR between the kernel and user domain for the developed real time system is 80 µSec as shown in Fig 7. The data was saved for sufficient number of iterations.

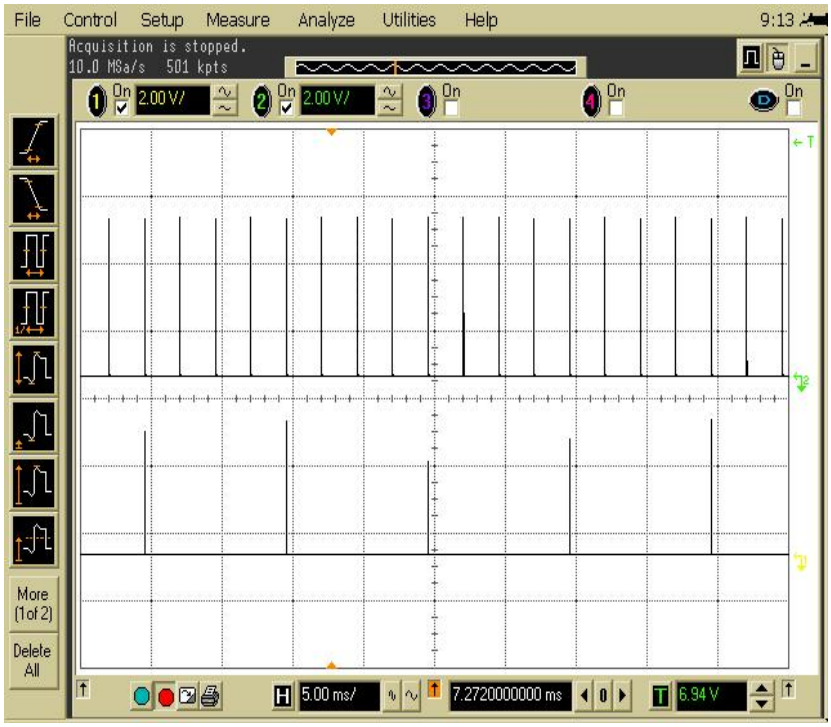


Fig. 5 The frequency of the tasks 400HZ and 100HZ running in MS-DOS 6.22

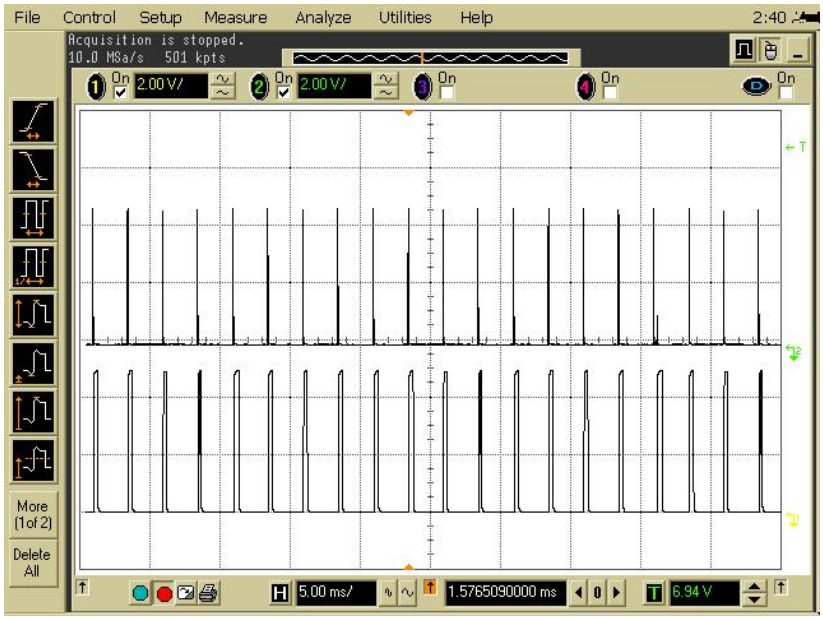


Fig. 6 The frequency of 400HZ tasks in kernel and user domain in RT/LINUX

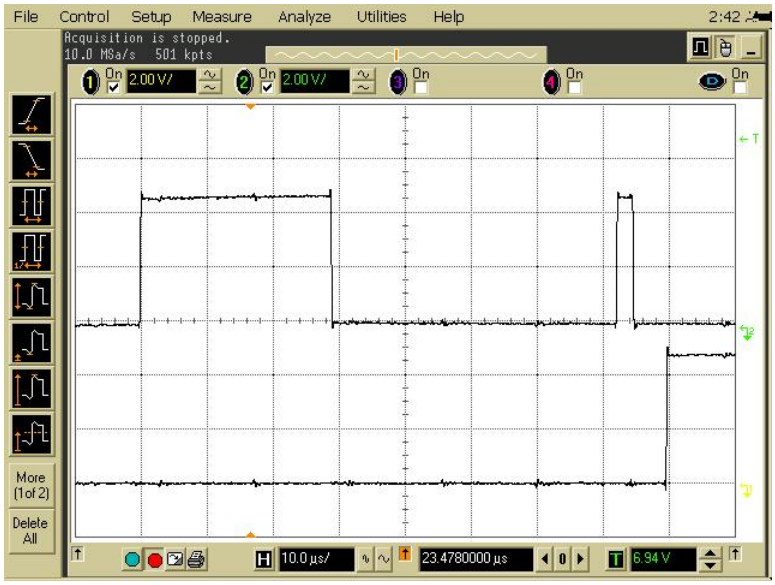


Fig. 7 Interrupt response time for the ISR between the kernel and user domain

6. Conclusion

A real time control system simulation framework has been established under MS-DOS 6.22 and RT/Linux environments. This includes are all constituents of real time control system. The performance was evaluated in these two operation system using hardware based measurements as well as real time function for measuring the execution time for the system tasks. The performed analysis shows that the developing of the real time system using RT/Linux was substantially better than MS-DOS 6.22. However, RT/Linux is an open source RTOS and active community, thus it was selected for simulation.

References

[1]. Lui, C.L. and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of Association for Computing Machinery 20, no. 1 (January 1973):46-61.

[2]. Quing L. and Caroline Y., "Real time concepts for embedded systems", published by CMP book and imprint of CMP media LCC, 2003.

[3]. Lehoczky, J.P., L. Sha, J.K. Strosnider, and H. Tokuda.1991."Fixed Priority Scheduling Theory for Hard Real-Time Systems.

[4]. klein,M.H, T. Ralya, B. Pollak, R. Obenza, and M.G Harbour.1993.A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-time system.Boston,MA:Kluwer Academic Publishers,ISBN 0-7923-9361-9

[5]. "Foundations of Real-Time Computing,scheduling,and Resource Management.Andre M. Van Tilborg,Gary M. Koob,editors.Boston,MA:Kluwer Academic Pulishers,ISBN 0-7923-9166-7

[6]. Turbo C Reference Manual

[7]. P.N Daly, Interfacing Real-Time Linux And LabVIEW, 2nd RTL workshop and conference, 2001.

[8]. Rafael Yanushevsky, "Modern Missile Guidance" CRC Press; 1st edition ,September 20, 2007

[9]. G. Tao, S. H. Chen, X. D. Tang, S. M. Joshi, Adaptive Control of Systems with Actuator Failures, Springer, April, 2004 (ISBN 1-85233-788-5)

[10]. Tarun Uppal and Hemendra Arya, " A Comparative Analysis of Real Time Operating Systems on a Mini- Areal Vehicle Hardware in the Loop Simulation", Proc. Of the International Conference on Aerospace Science and Technology, 26-28 June, 2008, Bangalorie, India.

[11]. Vishisht V Gupta , Prasanna G Gandhi , Hemendra Arya, et al , "Hardware in the Loop Simulator for Autonomus navigation of Mini-Areal Vehicle", 2nd international Conference of Computational Intilligence, Robotics, and Autonomus Systems, "CIRAS 2003", Singapore.

[12]. P. Sirkumar and C. D. Deori, 2000, Simulation of mission and navigation functions of Nishant, In Proc., National Workshop on Aerospace Flight Simulation, VSSC, TRIVANDRUM, INDEA.