

# One implementation of a Hybrid On-line process monitoring in PC based Real-Time systems

Bojan Jovanović<sup>1</sup>, Milun Jevtić<sup>1</sup>

1: Faculty of Electronic Engineering University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia

**Abstract:** This paper presents one way of FPGA implementation of hybrid (hardware-software based) on-line process monitoring in real-time systems (RTS). The reasons for RTS monitoring as well as RTS testing strategies followed by RTS monitoring approaches are presented at the beginning. Timing parameters of RT task along with monitoring scenarios for pre-emptive and non pre-emptive RT tasks are explained after. At the end, monitoring module is described in details. Also, FPGA implementation results and some useful monitoring system applications are mentioned.

**Keywords:** on-line hybrid monitoring, real-time systems, VHDL, FPGA

## 1. Introduction

For proper functionality of RTS it is necessary not only to give the correct results on the outputs, but to give them in exactly defined time interval. This is especially true for hard real-time systems (HRTS), because untimely execution of the tasks can lead to disaster. Tracking the course of events in RTS while RTS running we can make conclusions about meeting the timing requirements. Therefore, can be said with good reason that on-line monitoring (monitoring while system is running) of processes and events in HRTS is of enormous importance because it provides its predictable behaviour. It is needed in phase of system designing and verification to prove the fulfilment of timing requirements. It is also needed during the HRTS operation to detect errors that caused deviations of the predicted in time system behaviour. Implementing on-line monitoring we can check the execution time of every process/task or defined program code segments, both from upper (maximum execution time) and lower (minimum execution time) side.

## 2. RTS testing strategies

Monitoring system is the process or set of possible distributed processes whose main function is dynamic acquisition, interpretation and participation in information concerning application, during the application execution [1]. Therefore can be said that monitoring system improve vitality, security, fault tolerance and adaptability of RTS. Since the testing of the timing requirements of RTS directly depends

on processes, tasks and events monitoring, it is necessary to say a few words about RTS testing strategies. There are three basic strategies for testing and correct functioning validation of RTS. Sequential environment shown in Figure 1 is least complex for implementation. Testing scenario is created in off-line mode and test excitation is generated before execution of the test procedure. During the test, system response (the result of the test) is stored in real time and later analyzed in off-line mode to make a conclusion about functioning of RTS. The disadvantage of this approach is inability of the dynamic changes in the test scenario as well as inability of tracking the course of the test events. Also, the success or the failure of the test is known only after the analysis.

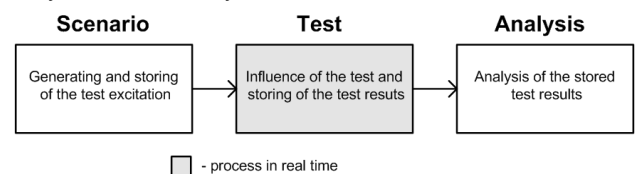


Figure 1: Sequential organization of the testing process

More complicated testing systems generate data for test scenario in real-time, during the system operation (Figure 2). Only the test results are analyzed in off-line mode.

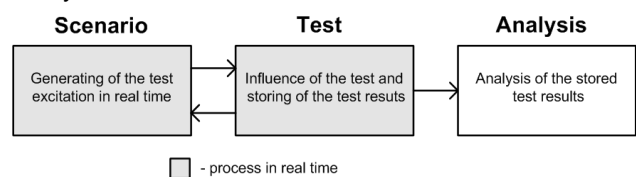


Figure 2: Testing with generating the test excitation in real time

Only configuration shown in Figure 3 provides complete testing in real-time and therefore on-line RTS testing.

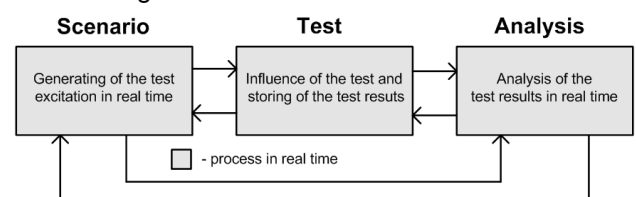


Figure 3: Testing scenario when all processes run in real time

## 2.1 RTS process monitoring

Monitoring system is intrusive if it requires the use of application resources (CPU time, I/O devices, communication channels etc.). Monitoring systems are mainly intrusive in some level. Completely non-invasive monitoring system use specialized hardware designed for monitoring. Ideal monitoring system which is completely transparent to the target system is very difficult to achieve in practice.

There are three basic approaches in implementation of system monitoring:

- Software,
- Hardware and
- Hybrid approach.

Software implementation of RTS monitoring is flexible, but largely intrusive and therefore significantly disturb RTS timing characteristics.

Hardware based approach in implementation of RTS monitoring is non-invasive in some level but requires specialized hardware. Whereas the target system must support the possibility of its installation, the use of this approach is inflexible and clumsy. It should be planned during the design of the target system.

Hybrid monitoring enables both, non-invasive nature of the hardware approach and the flexibility of a software approach. That's why the hybrid monitoring system is some kind of trade-off between pure hardware and pure software monitoring approaches.

## 3. RT tasks and events

Total correctness of an RTS operation depends not only upon its logical correctness, but also upon the time in which it is performed. This is especially true for HRTS where the completion of a task after its deadline is considered useless. Ultimately, this may cause a critical failure of the complete system. Dasarathy gave a classification of timing constraints for a RTS [2]. In general, there are two categories of timing constraints:

- Performance constraints that set limits on the response time of a system and
- Behavior constraints that make demands on the rates at which users apply stimuli to the system.

Checking the timing parameters of RT tasks and events, on-line monitoring checks correctness of their execution.

### 3.1 Timing parameters

RT task  $\tau_i$  can be characterized with the following timing parameters (Figure 4):  $r$  – moment of occurrence of the request for task execution;  $B$  – maximum delay to the start of task execution;  $C$  – task execution time (needed CPU time);  $D$  – time limit for task execution;  $T$  – period of occurrence of periodic tasks.

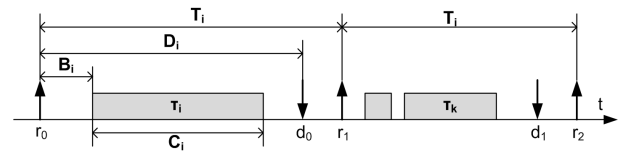


Figure 4: Timing parameters of RT task

### 3.2 RT task monitoring scenarios

According to its importance which they have in RTS as well as according to their timing parameters, RT tasks can be divided into pre-emptive and non pre-emptive tasks.

Concerning its execution time, pre-emptive tasks, unlike the non pre-emptive, do not have strict limits. Also, their possible failure in execution would not affect significantly the proper functioning of RTS. Therefore, scheduler can pause the execution of such tasks when receiving execution request from some higher priority RT task. After the execution of high priority task scheduler continues the execution of previously paused task. On the other hand, non pre-emptive tasks execution failure, or execution outside given time limits can lead to whole RTS failure. Because of this, high priorities are assigned to these tasks, and they can not be paused while running.

Non pre-emptive RT tasks: Possible course of non pre-emptive task ( $\tau_i$ ) execution is shown on Figure 5.

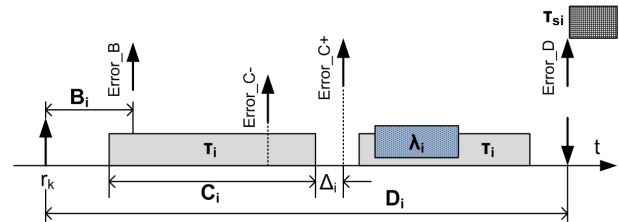


Figure 5: Monitoring scenario of non pre-emptive task execution

From the moment – event  $r_k$  when request for task  $\tau_i$  execution occurred, allowed delay to starting the task execution can be checked at first. This is important for the tasks that do not initiate with some external interrupt event. These tasks are „set“ in the queue for execution by some internal event. In the case of exceeding the interval  $B_i$ , monitoring timer-counter generates a hardware interrupt request, and error Error\_B is detected. Another monitored time interval is task execution time (CPU time). For task execution time which is shorter than  $C_i$  (minimum required time for correct task execution), marker Error\_C- is set. In the case of exceeding the task execution time  $C_i + \Delta_i$  (maximum time for correct task execution) monitoring module generates interrupt request to detect error Error\_C+.

Such monitor performs over each RT task. Upon detection of any of these errors, it is the policy of the planner and available time what will be taken.

Restarting of the same task or starting some alternative task ( $\lambda_i$ ) execution which will overcome given situation can be done. For each task, deadline  $D_i$  for his execution should also be monitored. Special counter-timer is most suitable for this purpose. In the case of his exceeding, interrupt request is generated and hardware-software security task ( $\tau_{si}$ ) is started. This security task should recover RTS or place it in a safe condition.

**Pre-emptive RT tasks:** Monitoring of pre-emptive tasks  $\tau_i$  (Figure 6) differs from the previous monitoring scenario. While his execution is stopped because of higher priority task  $\tau_j$ , its monitoring timer-counter should be stopped (during  $C_j$ ).

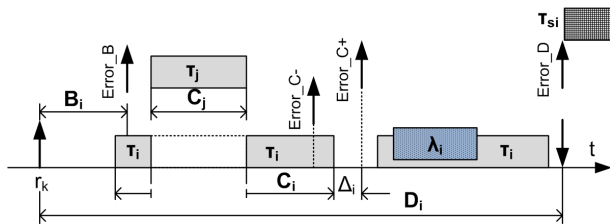


Figure 6: Monitoring scenario of pre-emptive task execution

#### 4. Related work

Can be said with a good reason that nowadays RTS have a wide variety of applications. For each RTS it is less or more useful to have some monitoring module. Monitoring modules are particularly essential for RTS in industrial applications since prediction and avoidance of potential system failures is of enormous importance. A good and efficient approach to analyzing the behavior of a real-time system is to use a monitor: a system that observes and analyzes the behavior of target system. Such a monitor could be used either as an "oracle" during system testing, or as a "supervisor" to detect and report system failure during operation [3],[4]. It is necessary to obtain it as online monitor – system monitoring in runtime without degradation of HRTS properties. Monitoring can be achieved at various levels. Low-level monitoring fetches every signal transmitted on the buses. High-level monitoring detects only process-level events.

Authors in [5] present a tool for real-time monitoring and diagnostics of dynamic systems with particular emphasis on gas turbine condition monitoring. In order to detect changes or deteriorations and predict the development of faults this software based monitoring system performs both on-line system continuously monitoring and software based simulations of turbine dynamic response.

At least, for a distributed RTS characterized by non-determinism, timing constraints and the low visibility of system behavior, authors in [6] propose a non-invasive monitoring architecture. Monitoring module is non-invasive because it extracts information

directly from traffic on the internal buses of a target distributed RT system. This module is general purpose microprocessor based system that consists of interface module and development module. The main function of interface module is to copy the internal states of the target node's processor and, under predefined trigger conditions, to start recording data from the buses on the target node onto the memory buffer unit. Development module uses recorded data to perform monitoring.

#### 5. Hybrid on-line process monitoring module

Depending on the application and environment, timing constraints imposed on a RTS vary widely. Here presented FPGA based monitoring module would be applicable to each RTS determined to meet strict timing constraints imposed by the real-world processes. FPGAs are chosen because of their low cost and ability for reconfiguration.

##### 5.1 General descriptions

Posing the demand that on-line monitoring do not require significant CPU time and clumsy additional specialized hardware, this paper presents one way of FPGA implementation of hybrid on-line RTS monitoring. It is intended for RTS based on an industrial PC and Linux operating system which is widely accepted and available open source system in RTS.

Implemented system monitors up to 32 processes i.e. RT tasks and events that execute in parallel. The number of monitored processes is relatively small, but it should be said that HRTS in industrial applications do not have a lot of processes. But since our monitoring module for 32 processes requires only 23% of FPGA resources, as will be seen later, number of monitored processes can be easily expanded up to 150.

The system is based on additional hardware module with 32 programmable timer-counters and interrupt logic [7]. Each monitored process has assigned his own timer-counter. Timers-counters are used as devices for defining the moments of events' time occurrence as well as watchdog i.e. monitoring timers for checking the correct timing execution of the processes. In addition, simple software primitives for on-line monitoring implementation are realized. They can be activated from the desired place in application program code. For monitoring of the processes and tasks in RTS without modification of application program code, simple modification of the operating system task scheduler and dispatcher is predicted. Modification ensures that scheduler or dispatcher, with every change of process/task status, activates appropriate software primitive for controlling timer and checking the time constraints.

For minimal intrusion and using of CPU time during monitoring, hardware module for PCs PCI slot is realized as shown on Figure 7.

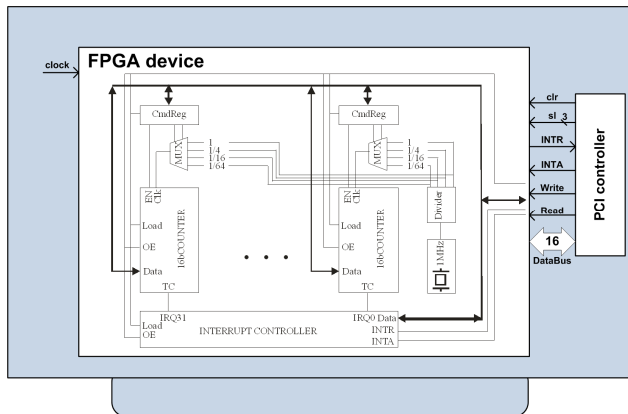


Figure 7: PCI card with hybrid on-line monitoring module

From Figure 7 can be seen that the interface from monitoring module to RTS consists of the following signals: Data Bus, Read, Write, INTR, INTA, sl and clr.

DataBus is a 16 bit bidirectional bus. It transmits the data from RTS to monitoring module and vice versa. RTS activates Read (Write) signals each time when need to read data from (write data to) monitoring module.

Monitoring module sets INTR (Interrupt request) signal each time when any of currently executing tasks do not execute properly or execute outside of required time interval.

As a response, RTS reads the message from Data Bus and sets INTA (Interrupt Acknowledge) signal. Message contains information about the interrupt nature and the ID of the task that caused interrupt. It is now scheduler policy to determine the actions that will be taken.

When receiving Interrupt Acknowledge, monitoring module resets INTR signal and continues to monitor RTS.

sl signal is 3 bit select used by RTS when selecting the register from which want to read data (or selecting the register to write data to). The use of this signal will be later explained with more details.

clr signal has a function of clear signal and it is used by RTS to reset monitoring module.

Monitoring module is controlled by software primitives from RTS and has the following functions:

- Setting the working mode of the timers-counters,
- Setting the time constraints,
- Enabling the timers-counters,
- Disabling the timers-counters,
- Reading the timers-counters,
- Timers-counters interrupt processing and

- Comparison of the timers-counters state with time constraints.

During the system verification phase monitoring system provides information about system timing characteristics and creates a log file. During the system operation it should detect deviations from predicted timing behaviour. These deviations could be the possible consequence of a failure in RTS. Thereby, monitoring system has two working modes. First mode refers to the system analysis. It performs with the purpose to measure the execution time of every RT task. Obtained information can be used for the future control of the RTS. In the second mode monitoring module has the function of built-in self-testing based on a watchdog function. It checks the upper and lower time limit at the tasks and periodic and quasi-periodic events level. The activation of each task initiates the procedure of starting his assigned timer-counter. Monitoring timer-counter sets to previously defined maximum task execution time and starts its countdown. If excess of the time interval happens, monitoring module sets interrupt request. If the task is complete before time excess, timer-counter stops its countdown with the end of task execution. Monitoring module reads its state and checks whether the task is executed before the minimum needed execution time. If the task is executed in regular time intervals RTS continues to work. Otherwise, scheduler starts provided procedure for system recovery from detected error. In this way, predicted behaviour of HRTS is ensured.

## 5.2 Monitoring module architecture

In monitoring module architecture we can clearly distinguish Data-path and Control Unit. But before we describe them both separately, let's still consider communication interface between monitoring module and RTS. From Figure 8 can be seen that monitoring module communicates with RTS using four different 16 bits registers: DataRegRead, DataRegWrite, CommandReg and StatusReg. All four registers have enable (EN) signals for activation. DataRegRead and DataRegWrite are registers for data storage. Using DataRegRead RTS reads data from monitoring module while using DataRegWrite register RTS sends data to monitoring module. For sending command to monitoring module CommandReg is used. Monitoring module status can be read from StatusReg. To access any of these registers RTS uses Read, Write and sl signals as shown in Table 1. So, setting sl to 000 and Read to 1 RTS activates DataReadReg (EnDataRegR='1') register. Its content is now available for RTS through DataBus. With sl=001 and Write=1 RTS ensures that DataRegWrite register is activated. 16 bit data from DataBus are now stored into this register. With sl=010 and Write=1 RTS stores data from DataBus

to CommandReg, while with sl=011 and Read=1 RTS reads the data from StatusReg.

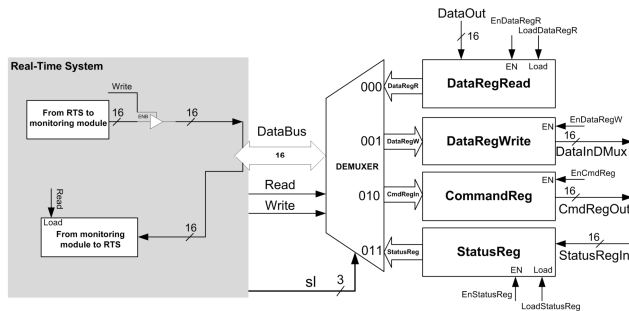


Figure 8: Monitoring module – RTS communication interface

Figure 9 shows bit-level structure of CommandReg and StatusReg registers. 10 MSB bits of StatusReg are not used. StatusReg(5) stores TCmin bit. This bit is set to 1 each time when RT task is executed faster than minimal required time for proper task execution. The meaning of this bit will be explained with more details when considering monitoring module Data-path. Since module monitors up to 32 processes it is necessary minimum 5 bits for identify each of them. So, 5 LSB bits of StatusReg store the identification of the RT task (process) which caused the interrupt – Interrupt ID.

Table 1: RTS – monitoring module communication

sl	Read	Write	Selected register	Action
000	1	0	DataRegRead	Reg to DataBus
001	0	1	DataRegWrite	DataBus to Reg
010	0	1	CommandReg	DataBus to Reg
011	1	0	StatusReg	Reg to DataBus

Concerning CommandReg, his 5 MSB bit are not used. CommandReg(10..9) bits store information about the time quantum which is used when measuring different time intervals. This two pace bits will be considered later. CommandReg(8..5) bits contain the code of the command while 5 LSB bits of this register address the task to which the command applies.

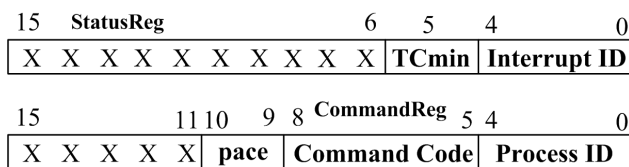


Figure 9: Status and Command register bit-level structures

**Monitoring module Data-path:** To show the whole monitoring module Data-path on a single figure would be complicated. Therefore, Figure 10 shows the part of the Data-path needed for a single RT process. Each of 32 processes has the same architecture. Data-paths of all processes are wired to 16 bits wide DataIn and DataOut buses so they

could communicate with RTS. All wires attached to DataIn and DataOut buses are in high-impedance state except one which in this moment uses the bus for communication. 1MHz clock and frequency divider are common for Data-paths of all processes.

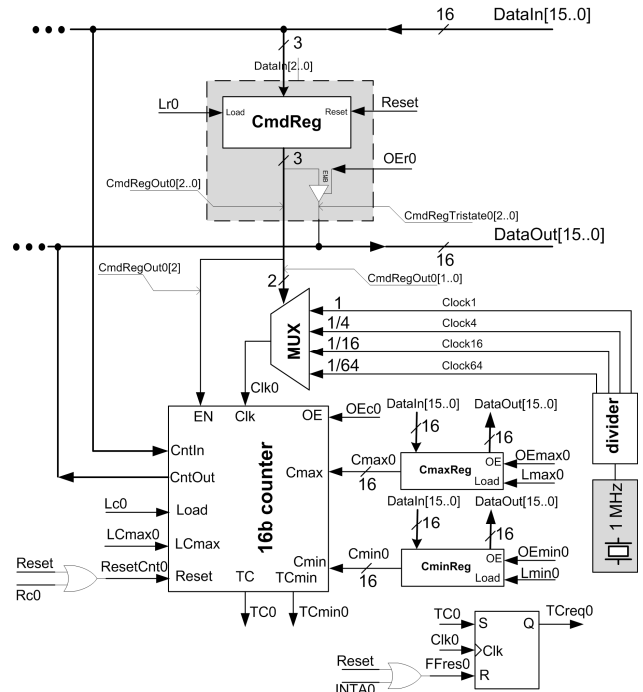


Figure 10: The part of the Data-path needed for a single RT task

As shown on Figure 10, single process Data-path consists of one 3 bits wide CmdReg, one 4in1 multiplexer, one 16 bit counting-down counter, two 16 bits wide registers for storing constants  $C_{max}$  and  $C_{min}$  and one RS flip-flop.

CmdReg MSB bit is used to enable/disable counter while 2 LSB bits are attached to multiplexer select signal in order to determine counter clock frequency (pace bits). By setting Lr to 1 CmdReg can be loaded from DataIn bus. Similarly, by setting OEr his content is available through DataOut bus. After reset, CmdReg content is 000. Using 1MHz clock source time quantum for time measuring can be 1, 4, 16 or 64  $\mu$ s. Accordingly, maximum time for task execution can be 65.5, 262, 1048 or 4194 ms. It should be noted that by changing clock source we can obtain different time quantum and different maximum task execution times.

CmaxReg and CminReg are 16 bits registers intended for storing the constants that determine maximum and minimum task execution time, respectively. They are also wired to DataIn and DataOut buses so they can be loaded through DataIn bus by setting Lmax (Lmin) or its content can be read through DataOut bus by setting OEmax (OEmin). They are also connected to counter. Counter is 16 bit and of counting-down type. Its starting value can be set either from DataIn bus (by

setting Lc to 1) or from CmaxReg (by setting LCmax to 1). Through DataOut bus his current state can be read (by setting OEc to 1). TC bit is set to 1 when counter, counting backward, reach the zero. This means that maximum allowed task execution time has expired. In all other counter states TC bit is 0. Being S input of RS flip-flop, TC bit controls his TCreq output. When TC=1, TCreq which, as will be seen later, has a direct impact to INTR bit, is also set to 1. TCmin counter output gives us the information whether or not minimum task execution time has expired. Since the counter can monitor both CmaxReg and CminReg registers, it can measure time expired from task execution beginning (Equation 1) and compare it with minimum needed one stored in CminReg.

$$\text{expired\_time} = \text{CmaxReg} - \text{counter\_state} [1]$$

While expired time is less than minimum required task execution time ( $\text{expired\_time} < \text{CminReg}$  value), TCmin is set to 1, else to 0. After reset, counter is in 111...111 state. The part of the monitoring module Data-path which is common to all tasks is shown on Figure 11.

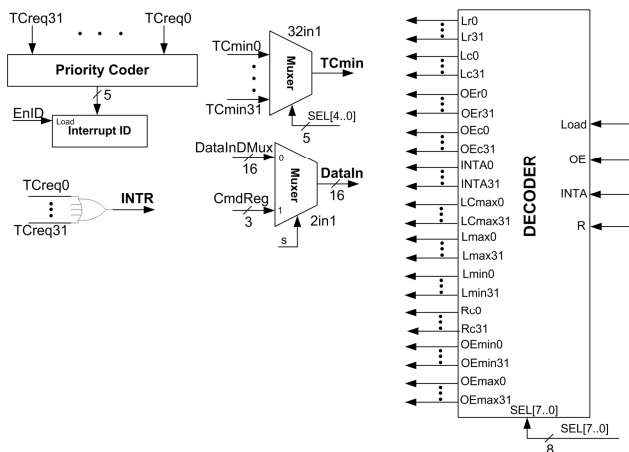


Figure 11: The part of the Data-path common to all RT tasks

Decoder which is on the right-hand part of the Figure 11 is used to decode the way in which the inputs (Load, OE, INTA and R) are connected to the outputs. This is all done using 8 bits wide SEL signal. Load input signal can be connected on the following outputs: one of 32 different Lr signals, one of 32 different Lc signals, one of 32 different LCmax signals, one of 32 different Lmax signals or on one of 32 different Lmin signals. OE input signal can be connected on the following outputs: one of 32 different OEr signals, one of 32 different OEc signals, one of 32 different OEmax signals or one of 32 different OEmin signals. INTA input signal can be connected on one of 32 different INTA outputs while input R can be connected on one of 32 different Rc output signals. To select which of the 32 different outputs will be connected with the input 5 LSB bits of

SEL signal are used. In the case of Load and OE signals, since they can be connected on different types of Load and OE outputs, 3 MSB bits of SEL signal are used to determine its connection to the output. The way of determination is shown in Table 2.

Table 2: Load and OE signal connection protocol

SEL(7..5) bits	Load connects to	OE connects to
000	Lc	OEc
001	LCmax	
010	Lr	OEr
011	Lmax	OEmax
100	Lmin	OEmin

So for example, let it SEL signal be equal to 01100111. With 5 LSB bits (00111) the 7<sup>th</sup> of a 32 different output is selected. R7 is connected to R and INTA7 is to INTA. In the case of Load and OE outputs, since 3 MSB bits of SEL signal are equal to 011, Lmax7 is connected to Load and OEmax7 to OE.

With 32in1 multiplexer and using 5 LSB bits of SEL signal, one of the 32 different TCmin bits is connected to unique TCmin output.

32 different TCreq signals are connected to the priority coder inputs. Priority coder gives 5 bits identification of the process that caused the interrupt. In the case when two or more processes require interrupt, priority coder will identify the process with the highest priority. 32 TCreq signals are also connected to 32 inputs of OR logic gate with INTR output.

So, if maximum allowed task execution time has expired, TCreq is set to 1. Consequently, INTR is also set to 1, and priority coder gives us the 5 bits identification of the task that caused the interrupt. These 5 bits are stored in Interrupt ID register.

**Monitoring module Control Unit:** For monitoring module Control Unit finite state machine (FSM) is used. FSM can be in the one of the following states: s0, s1, s2, Start Process, Pause, End Process, Start Measuring Time, Continue, Stop Measuring Time, Load Counter, LoadRegMax, LoadRegMin, Read Counter, ReadRegMax and ReadRegMin. FSM clock source is equal to 50MHz. Consequently, FSM moves from current to the next state every 20ns. First part of algorithmic state machine (ASM) chart of FSM is presented on Figure 12. After reset in s0 state, next state is s1. In the s1 state FSM monitors whether interrupt occurred. If so (INTR=1) FSM goes to Interrupt state for interrupt processing. If there is no interrupt (INTR=0), FSM waits for the command to be received. If the command is received next state is s2, else s1. In s2 state FSM reads the command and moves to the next state according to received command stored into CommandReg (Figure 9). Pace bits from this register determine 2 LSB bits of CmdReg, thus defining counter clock frequency. Process ID bits determine RT task the command



applies to, while 4 Command Code bits from CommandReg define desired command.

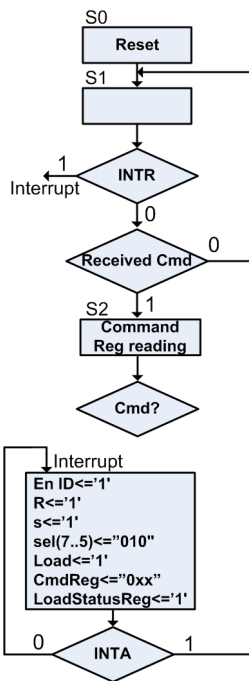


Figure 12: The first part of ASM chart

In interrupt state FSM loads Interrupt ID (En ID=1) and status registers (LoadStatusReg=1) as well as reset counter. Also, by writing 0xx (xx are pace bits) into CmdReg counter is disabled. From interrupt state FSM goes to s1 state after receiving interrupt acknowledge (INTA) from RTS. Depending on the received command, FSM can move from s2 to any state shown on Figure 13. When Command Code is 0000, FSM moves to Start Process state. In this state FSM loads the counter with Cmax value from CmaxReg (LCmax=1) and enables counter to start counting down (CmdReg=1xx). For 0001 Command Code FSM is in Pause state. Here, FSM disables counter by writing 0xx to CmdReg. In End Process state (Command Code=0010) counter is disabled and the value of TCmin bit is stored to StatusReg. If TCmin=1 RTS knows that task was executed faster than minimum required time for correct task execution. When in Start Measuring Time state (0011), FSM resets the counter (Rc=1) and enables its counting down. Continue state (0100) is opposite with Pause state. Here, FSM enables previously disabled counter. Similarly, Stop Measuring Time (0101) state is opposite with Start Measuring Time. Here, counter is disabled and its current state is loaded to DataRegRead (LoadDataRegR=1) register through DataOut bus (OEc=1). From here, it is available to RTS.

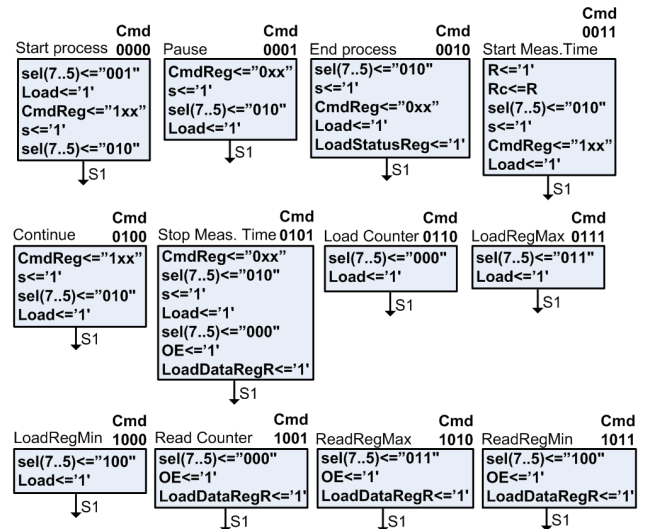


Figure 13: The rest of ASM chart

In Load Counter state (0110) data is loaded from DataIn bus to counter, in LoadRegMax (0111) from DataIn to CmaxReg, while in LoadRegMin state (1000) from DataIn to CminReg register. FSM in Read Counter state (1001) stores counter state through DataOut bus to DataRegRead register. When in ReadRegMax (1010) or ReadRegMin (1011) state, FSM stores data from CmaxReg or CminReg to DataRegRead register.

## 6. FPGA implementation

Each part of monitoring module communication interface as well as of Data-path and Control Unit is described in VHDL programming language and implemented in EP2C35F672C6N FPGA chip on Altera DE2 board [8]. The results of implementation are shown in Table 3.

Table 3. FPGA implementation results

	Total logic elem.	% FPGA	Clock setup
Interface	165/33216	< 1%	241.25MHz
Data-path	7169/33216	22%	78.47MHz
FSM	172/33216	< 1%	280.50MHz
Σ	7506/33216	23%	

From the Table can be seen that Data-path is the most critical part of FPGA implementation. It requires the most of FPGA resources and also determines maximal operating frequency.

In order to prove its correct functionality monitoring module was tested using DE2 board. From the board commands were sent to the module and its response were observed using registers of monitoring module communication interface. For all possible commands monitoring module responded as expected. Since monitoring module was successfully tested it is now needed to chose one of many possible development boards with PCI interface to implement it in. As a low cost solution authors propose some of the Raggedstone1 PCI development boards [9]. PCI

core for communication can be additionally ordered or found as an open core on [10].

## 7. Monitoring module applications

Some possible monitoring module applications were not mentioned so far. That is because the monitoring module was not developed for some particular applications. Author's intention was to make it appropriate, with less or more changes, to as much different RTS applications as possible.

The online monitor realization as quite independent system of the objective HRTS, may result in very complex and expensive real-time system (whose affect to the system reliability would be very interesting for considering), or in system that would have a weak access to the events inside the HRTS. Here, the realization of the event monitoring in time is considered, and the attention is paid on monitoring realization and application as a system for checking the HRTS behavior in time. Through the monitoring of the running tasks, faults in software running can be detected and predictive behavior of HRTS can be provided.

Much greater number (up to one thousand) of timers can be placed on a single FPGA integrated circuit. On that way, even one thousand processes (internet links) in some server computer, can be monitored.

## 8. Conclusion

The need for an effective RTS monitoring is obvious, especially in the case of HRTS. In order to be as less intrusive as possible and as much flexible as possible, one hybrid approach on RTS monitoring is proposed. Intended for PC based RTS, monitoring module uses PCI slot. Monitoring module is described in details, along with its FPGA implementation and some possible applications. It should be said that with PCI Express standard emerging proposed monitoring module can be less intrusive and more efficient.

## 9. References

- [1] Jane W., Liu S.: "*Real-Time Systems*", Prentice Hall, 2000.
- [2] Dasarathy B.: "*Timing constraints of Real-Time Systems*", IEEE Trans. Software Engineering, Vol. 1, pp. 80-86, 1985.
- [3] Dennis K. Peters, David Lorge Parnas, Requirements-Based Monitors for Real-Time Systems, IEEE Transactions on Software Engineering, Vol. 26, No. 2, February 2002, pp. 146-158.
- [4] Milun Jevtic, Volker Zerbe, Sandra Brankov, "*Multilevel Validation of On-Line Monitor for Hard Real Time Systems*", Proc. 24th International Conference on Microelectronics - MIEL 2004, Vol. 2, pp. 755-758, Nis, Serbia and Montenegro, 16-19. may, 2004.

- [5] Miline R., Nicol C.: "*TIGER: a model based and a rule based real-time tool for monitoring and diagnostics of dynamic systems*", IEE Colloquium on Real-Time Knowledge Based Systems, London, 2002.
- [6] Tsai J., Fang K., Chen H.: "*A Noninvasive Architecture to Monitor Real-Time Distributed Systems*", IEEE Computer Society, Vol. 23, pp. 11-23, 2004.
- [7] Jovanovic B., Jevtic M.: "*Module for run-time monitoring in PC hardware based real-time system*", Int. Scientific Conference - Unitech, Gabrovo, Bulgaria, 2009.
- [8] Altera DE2 user's manual:  
[ftp://ftp.altera.com/up/pub/Webdocs/DE2\\_UserManual.pdf](ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf)
- [9] Raggedstone1 user's manual:  
<http://www.enterpoint.co.uk/moelbryn/raggedstone1.html>
- [10] Opencores PCI core:  
[http://www.opencores.org/project,pci32lite\\_oc,overview](http://www.opencores.org/project,pci32lite_oc,overview).

## 10. Glossary

- ASM*: Algorithmic State Machine  
*ARM*: Advanced RISC Machine  
*FPGA*: Field Programmable Gate Array  
*FSM*: Finite State Machine  
*HRTS*: Hard Real Time System  
*IC*: Integrated Circuit  
*LSB*: Least Significant Bit  
*MSB*: Most Significant Bit  
*PCI*: Peripheral Component Interconnect  
*RTS*: Real Time System  
*RT task*: Real Time task  
*USB*: Universal Serial Bus  
*VHDL*: Very high speed integrated circuits Hardware Description Language