ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA150

INTRODUCING FAST INTERLOCKS IN THE UNICOS-CPC FRAMEWORK

M. Vázquez Muñiz, J. Ortolá Vidal, E. Blanco Viñuela, CERN, Geneva, Switzerland

Abstract

The CERN UNified Industrial COntrol System framework (UNICOS) with its Continuous Control Package (UNICOS-CPC) is the CERN standard solution for the design and implementation of continuous industrial process control applications. The need of adapting the framework capabilities to the different processes at CERN has brought new challenges. Reacting as fast as possible to an interlock situation to protect equipment is a new requirement which has been introduced in UNICOS-CPC.

This paper presents the challenges, design and test results of the seamless integration of fast interlocks capabilities in the current UNICOS-CPC package based on conventional PLCs (Programmable Logic Controllers), with a heightened level of flexibility and maturity. The first implementation is employing SIEMENS PLCs but the underlying technique is extensible to the other UNICOS-CPC compliant platforms.

INTRODUCTION

UNICOS is a CERN framework to develop industrial control applications and UNICOS-CPC is the framework package devoted to continuous process control. It provides developers with means to design and develop full control applications and operators with ways to interact with all items of the process. In addition UNICOS-CPC offers a suite of tools at the supervision level to diagnose the process and the control system itself. [1]

The methodology to develop process control applications proposed by UNICOS-CPC is based on the model provided by the ISA-88 standard for batch control systems. The basis of these standards embraces the methodology known as "Modular Functions", which supports multi-use instances to minimize and simplify the coding. The standard defines a hierarchy of objects that standardize the interaction between the layers, thus simplifying the overall system. These objects are classified according their functionality (i.e. Input/Output, Interface, Field and Control Objects) and are used as a common language by process engineers and control system programmers to define the functional specification of the process control.

In addition to the method, offline tools have been produced to automate the instantiation of the objects in both supervision and process control layers, and generate the Programmable Logic Controller (PLC) programs.

The UNICOS-CPC package can be deployed to different platforms. For the control layer, Siemens and Schneider PLCs are supported together with controllers compatible with Codesys development environment. At the supervision layer, the Siemens Supervisory And Data Acquisition (SCADA) WinCCOA is used and it also includes a full library for Siemens and Schneider local operator panels.

Fast Interlocks

Fast interlocks are defined as critical events in the process that require detection, evaluation and a response by the control system in a time window that cannot be achieved by a standard PLC application. They are used to set the equipment under control in a safe state in response to an abnormal situation. In addition to evaluating and responding to those events, it is necessary to provide an accurate time stamp of the event which will be used to diagnose the root cause.

TSPP

The Time Stamp Push Protocol (TSPP) is an event driven communication protocol for process control. TSPP provides both, optimized data transfer from the PLC to the SCADA and time-stamped data at source.

The protocol detects data changes, associates the time stamp and then sends it in a single telegram. The time stamp allows better diagnostic capabilities than classic polling data.

TSPP is designed to send three distinct types of time stamped data: Events (Boolean changes in the state of the UNICOS-CPC objects), Status (analogue changes of the objects) and Watchdog (connection alive message). The three types of TSPP data buffers are managed in parallel but only one send-channel is used. To manage this mechanism, a first in – first out (FIFO) queue has been designed.

Events are individually time-stamped, buffered and then sent to the SCADA, which allows a comprehensive event analysis in case of failure. However, statuses are time-stamped in blocks and sent to the data server without buffering, therefore only the most recent status values are ensured.

On Siemens PLCs the protocol is implemented using a standard S7 communication function, BSEND, which sends large amounts of data to the SCADA layer (WinCC OA).

CRITICAL EVENT DETECTION

The UNICOS-CPC standard application time granularity is the one imposed by the PLC sampling time, which depends mostly on the application size. In general, the PLC sampling time does not comply with the fast interlocks time requirements. Two implementation solutions for the detection of critical events have been evaluated.

Hardware Interrupts

The hardware catalog of Siemens provides a set of input modules with hardware interrupt capabilities, a program which is adapted to suit the event can be called in real time. If an alarm-triggering event occurs during the main PLC program processing, the operating system calls the alarm

OB 40, interrupting the processing of the program cycle or lower-priority program blocks. The alarm-triggering event is specified more precisely via the alarm OB 40 temporary local data which can be evaluated by the user program in the alarm OB. This functioning principle is shown in Fig. 1.

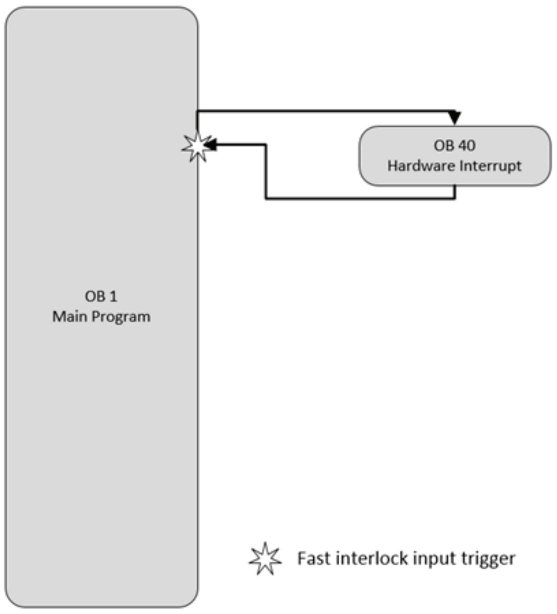


Figure 1: Hardware Interrupt execution.

Cyclic Interrupts

Siemens CPUs provide cyclic interrupt OBs that interrupt the cyclic program processing at certain intervals. Each of the available cyclic interrupt OBs has a default interval that becomes effective when the cyclic interrupt OB assigned to it is loaded into the CPU. The equidistant start times of the cyclic interrupt OBs are determined by the interval and the phase offset. The user must make sure that the run time of each cyclic interrupt OB is significantly shorter than its interval otherwise in case a cyclic interrupt OB has not been completely executed before it is due for execution again, a time error OB will be started.

This implementation relies on a periphery update during the execution of the cyclic interruption in order to achieve the detection of the critical events. As opposed to the previous implementation based on hardware interrupts, the cyclic interrupts cannot ensure a fixed time window for the event detection but a maximum time window based on the cyclic interrupt interval set by the user. The event will be detected at the next cyclic interruption execution. In general the minimum interruption interval available is 1 ms. The functioning workflow is shown in Fig. 2.

If the input (I) and output (Q) address areas are accessed in the user program, the program does not scan the signal states on the digital signal modules but accesses a memory area in the system memory of the CPU and distributed I/Os. This memory area is known as the process image. One of the internal tasks of the operating system (OS) is to read

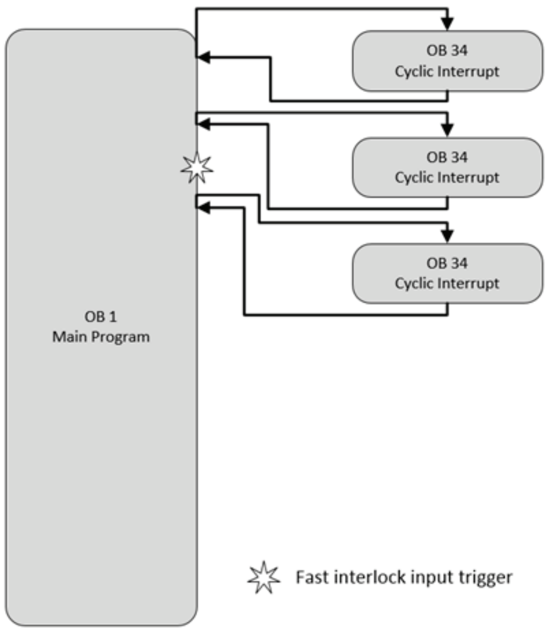


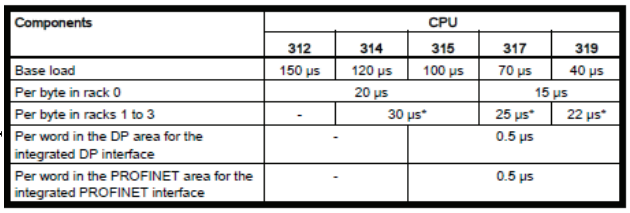
Figure 2: Cyclic Interrupt execution.

the status of inputs into the process image input table (PII). Once this step is complete, the user program is executed with all blocks that are called in it. The cycle ends with writing the process image output table (PIQ) to the outputs for the modules. Reading in the process image input table and writing the process image output table to the outputs for the modules is all independently controlled by the operating system.

For either of the two solutions (hardware or cyclic interrupts) introduced in this paragraph, the inputs and outputs readout update corresponding to the signals related to the event detection (inputs) and the control system response (outputs) need to be treated during the execution of the interruption. A mechanism to update first the inputs at the interruption execution and then the outputs after the evaluation of the event is necessary in order to minimize the overall reaction time applied to the execution chain: inputs readout, evaluation and output to the process. In order to achieve this purpose, three implementations have been studied and identified for Siemens PLCs.

Process Image Update

Using the system functions SFC 26 "UPDAT_PI" (update inputs process image) and SFC 27 "UPDAT_PO" (update outputs process image) it is possible to update the OB1 process input/output image or a process input/output image partition previously defined. Time needed for the process image transfer depends on the CPU used. The updating of the OB1 process image input table and the process image input sections that are assigned to an interrupt OB is not influenced by SFC 26 calls. In Fig. 3 the time needed for the CPU 31x is shown.



Components	CPU				
	312	314	315	317	319
Base load	150 µs	120 µs	100 µs	70 µs	40 µs
Per byte in rack 0	20 µs			15 µs	
Per byte in racks 1 to 3	-	30 µs*		25 µs*	22 µs*
Per word in the DP area for the integrated DP interface	-		0.5 µs		
Per word in the PROFINET area for the integrated PROFINET interface	-		0.5 µs		

Figure 3: Process image transfer time for CPU 31x. [2].

Process Image Partition

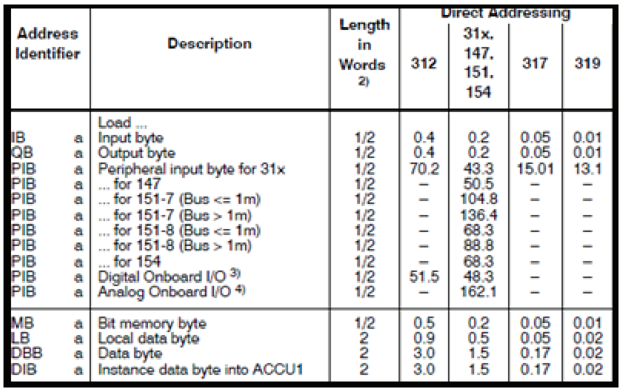
As an alternative to having the process image (process-image input table, PII, and process-image output table, PIQ) automatically updated by the operating system, inputs and output addresses can be assigned to a partial process images (PIP). Each PIP can contain multiple IO addresses or module assignments although it can only be assigned to a unique OB. Input and output addresses, which have been assigned to a process image partition, no longer belong to the OB1 process image of the inputs and outputs. All the input and output addresses can only be assigned once for the OB1 process image and for all the process image partitions. The behavior described applies to the S7-400 and to some S7-300 CPU modules like CPU317 and CPU319. The CPU modules S7-317 and S7-319 support the process image partition only with the OB61 and not with the other alarm OBs. The number of process image partitions available depends on the CPU used.

Peripheral Addressing

If access is made to the inputs and outputs via the operands "I" or "Q" in the user program, then there is no direct access to the input-output modules. In this case, access is made to the process image of the inputs (PII) and the process image of the outputs (PIQ). The contents of PII and PIQ do not reflect the actual values of the inputs/outputs, but the values at the time the process image was updated. Whenever more recent values at the interruption execution are required, direct peripheral addressing is also possible. In addition I/O addresses outside the process image to the inputs and outputs of I/O modules can be assigned and must be accessed via "Peripheral addressing" and therefore it is always ensured that the actual value is read immediately or an output is implemented immediately. To represent this in the user program a "P" in front of the area of the address to be addressed shall be set. An important drawback to note is, direct access to a peripheral address involves a much higher access time than access to the process image, as shown in Fig. 4. In addition, access is restricted to byte, word and double word data types, thus addressing of any individual peripheral bits is not possible.

IMPLEMENTATION

In regards to the critical event detection solutions depicted in the previous chapter, both solutions are based on interrupts (hardware or cyclic). Hardware interrupts present a



Address Identifier		Description	Length in Words 2)	Direct Addressing			
				312	31x. 147. 151. 154	317	319
		Load ...					
IB	a	Input byte	1/2	0.4	0.2	0.05	0.01
QB	a	Output byte	1/2	0.4	0.2	0.05	0.01
PIB	a	Peripheral input byte for 31x	1/2	70.2	43.3	15.01	13.1
PIB	a	... for 147	1/2	–	50.5	–	–
PIB	a	... for 151-7 (Bus <= 1m)	1/2	–	104.8	–	–
PIB	a	... for 151-7 (Bus > 1m)	1/2	–	136.4	–	–
PIB	a	... for 151-8 (Bus <= 1m)	1/2	–	68.3	–	–
PIB	a	... for 151-8 (Bus > 1m)	1/2	–	88.8	–	–
PIB	a	... for 154	1/2	–	68.3	–	–
PIB	a	Digital Onboard I/O 3)	1/2	51.5	48.3	–	–
PIB	a	Analog Onboard I/O 4)	1/2	–	162.1	–	–
MB	a	Bit memory byte	1/2	0.5	0.2	0.05	0.01
LB	a	Local data byte	2	0.9	0.5	0.05	0.02
DBB	a	Data byte	2	3.0	1.5	0.17	0.02
DIB	a	Instance data byte into ACCU1	2	3.0	1.5	0.17	0.02

Figure 4: Typical execution time in µs for a byte load instruction using process image or peripheral addressing in CPU 31x. [3].

streamlined event detection mechanism although it requires a specific set of hardware modules, whereas the cyclic interrupt performance is lower. The process image update required at the interrupt execution has been implemented using the peripheral addressing due to its better performance compared to the alternative solutions studied.

UNICOS-CPC Integration

The integration of fast interlocks functionality into the UNICOS-CPC framework is based on the execution in the interrupt OB (cyclic or hardware interrupt) of the UNICOS-CPC objects instances involved in the interlock chain (detection, evaluation and reaction). This chapter describes the object types than can be part of the interlock chain, together with the issues found at the implementation phase.

UNICOS-CPC Objects

Among the objects composing the UNICOS-CPC object types library, those shown in Figure 5 have been defined as conforming with a fast interlock execution chain according to the UNICOS-CPC model. The user must build the chain composed by a minimum of one instance of each object type in the list.

- DI: The Digital Input object type connects the input periphery to the PLC application.
- DO: The Digital Output object type connects the PLC application to the output periphery.
- DA: The Digital Alarm object type indicates the interlock and propagates the alarm to the OnOff object type.
- OnOff: The OnOff object type evaluates the interlock logic conditions and connects the interlock to the output.

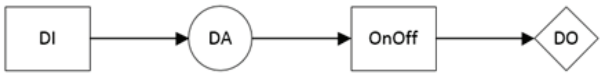


Figure 5: Fast interlock chain of UNICOS-CPC objects.

Consistency

One of the typical issues to be handled when using interrupts is the inconsistency of the data used at the main program execution and the interrupt execution. The data shared by both programs (main and interrupt) need to be identified and measures have to be considered to avoid inconsistencies that could jeopardize the functionality of the system. Peripheral addressing is a good example of this issue.

The peripheral addressing of inputs and outputs is restricted to bytes as the minimum unit of information. In order to avoid data consistency issues when updating the periphery image, only the input bits involved in the fast interlock chain corresponding to the Digital Input Objects must be updated. To serve this purpose a mechanism has been implemented for which the input byte corresponding to the relevant input bits of the interlock chain is stored in an internal variable. The fast interlock input bits are then identified in the periphery byte and copied back to the initial input image byte. In this way, the input image bytes updates only the relevant bits related to the fast interlock Digital Inputs involved in the fast interlock chain. The mechanism implemented is shown in Figure 6.

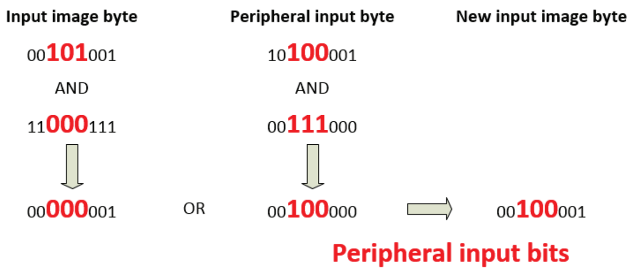


Figure 6: Peripheral input byte treatment.

Concurrency

The treatment of the fast interlocks implies the execution of object instances and additional shared resources during the interrupt execution, which generally occur during the execution of the main program OB (OB 1). In this context and considering that interrupts can be executed at any time during the OB1 execution, the access to those shared resources must be handled to prevent unexpected misbehavior of the program, thus a mechanism to synchronize the access to those resources has been implemented.

The mechanism is based on the use of two system functions which allow the disabling of processing of new interrupts of both synchronous and asynchronous interrupts in such a way that the interrupts are only permitted in the execution of the OB1 whenever concurrency issues are not possible.

- SFC 39 "DIS_IRT" and SFC 40 "EN_IRT".
  With these system functions it is possible to disable and enable respectively the interrupts, such as hardware or cyclic interrupts. While the interrupts are disabled, no OB is triggered by an interrupt trigger occurrence.
- SFC 41 "DIS_AIRT" and SFC 42 "EN_AIRT"
  With these system functions it is possible to disable (until the end of OB execution or until re-enabling the interrupts) and enable respectively the interrupts, such as hardware or cyclic interrupts. Unlike SFC 39 and SFC 40, interrupts triggered during interrupt disabled period will be queued and executed once the interrupts are enabled again.

UNICOS-CPC Objects Interactions

In general terms, as detailed in the previous paragraph, the connection between objects instances used in the fast interlock chain and the other objects instances may rise potential concurrency and consistency issues. Thus, the final object architecture of the control system for fast interlock applications has been limited to the connections shown in Figure 7.

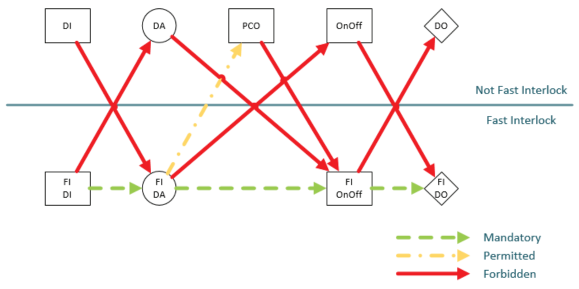


Figure 7: Connection schema among the fast interlock objects chain and other objects.

In order to be able to evaluate a fast interlock alarm in the standard execution processing of the program, the connection between the Digital Alarm object and the PCO (Process Control Object) is allowed. On enabling this feature, some parts of the instance execution are prevented against interrupts using the aforementioned system functions, however, after a deep analysis of the execution steps, consistency issues may remain in the particular case of two or more PCO instances connected to the same fast interlock digital alarm. As a result, the connection between a Fast Interlock Digital Alarm and a PCO is not advised even if it has not been disabled for functionality purposes. If necessary it is advised to use only one connection to a PCO. Several semantic check rules have been included in the generation process to prevent the user from misconfiguring the application.

TEST RESULTS

A test bench containing a standard UNICOS-CPC application with a defined fast interlock chain has been established. The code is executed by a Siemens S7 317-2 PN/DP CPU (typical CPU processing times for: bit operations 0.025 µs, word operations 0.03 µs, fixed point arithmetic 0.04 µs, floating point arithmetic 0.16 µs). The response time measurements have been performed with an oscilloscope.

Reaction time measurements have been taken for different PLC cycle times. The CPU cycle time was modified dynamically with a dedicated routine.

Standard Application

In general, the response time for an event (non-fast interlock) should be observed to be between the PLC cycle time and twice that value. In Figure 8, the theoretical response time range is highlighted in grey and the results from the test performed in a standard application are represented with dots. As expected, the overall response time depends on the cycle time of the PLC and does not exceed two cycles execution time.

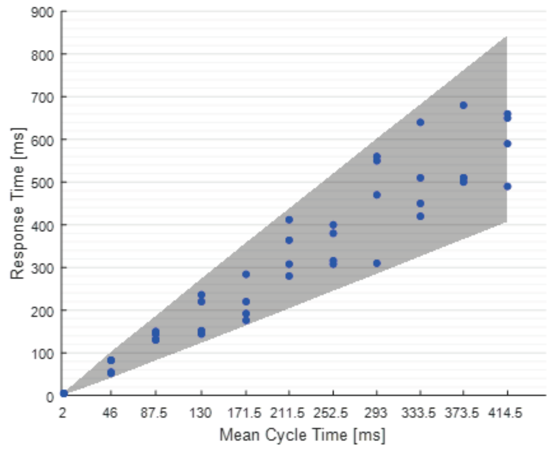


Figure 8: Response time for the standard object execution (theoretical range in grey).

Hardware vs. Cyclic Interrupts

As designed, the response time for fast interlock is independent of the PLC cycle time, due to the fact that the interrupt is always executed irrespective of the PLC run-time status. For the hardware interrupt implementation, most tests performed show an average response time of around 3 ms. For the cyclic interrupt implementation, the results confirm the expectations; the final response time is longer as the interval of the cyclic interrupt (1 ms) must be added, obtaining a response time that represents the values achieved for the hardware interrupt plus the interval time of the cyclic interrupt. In the Figure 9, maximum, minimum and average response times of hardware interrupts and cyclic interrupts vs PLC cycle time are represented.

Impact on PLC Cycle Time

The use of interrupts in general, and cyclic interrupts in particular, has an impact on the overall execution performance and therefore may cause an increase of the PLC cycle time. In order to evaluate the impact of the cyclic interrupt interval defined by the user, comprehensive tests have been performed with multiple input bytes. The results show a correlation between the increase of the total cycle time and the reduction of the cyclic interrupt interval. In addition,

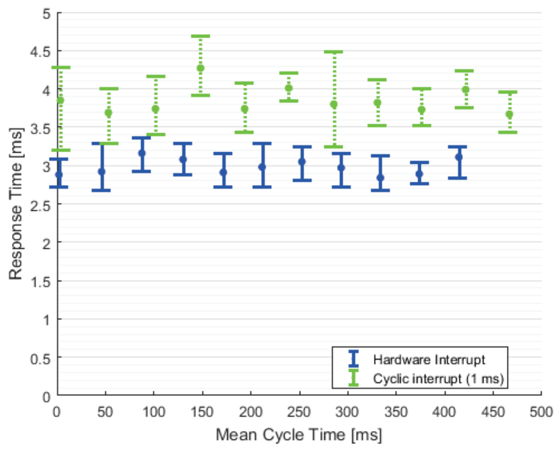


Figure 9: Response time for hardware and cyclic interrupts.

when the cyclic interval is significant compared to the total cycle time, the impact of the cyclic interruption is negligible. Low cyclic interrupt intervals (e.g. 1 ms) ensure a fast interlock response penalizing the system run-time performance. As opposed to this, high cyclic interrupt intervals (11 ms) reduce to the minimum the impact of the interrupt on the system run-time performance, increasing the fast interlock chain response times accordingly. The results are represented in Figure 10.

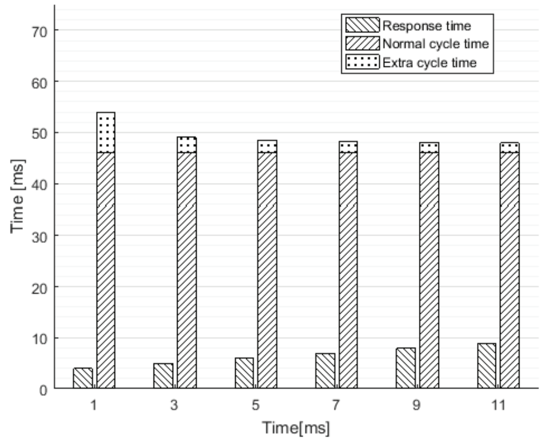


Figure 10: Cycle time increase vs cyclic interval decrease.

Another interesting effect detected during the test phase is the impact of the amount of fast interlock input bytes readout processed by the interrupt routine, based on the delays provoked by the periphery access. The overall PLC cycle time increases due to the impact of the interrupt routine which holds the periphery access. As a consequence, the fast interlock response time is affected, increasing as expected with the increase of input bytes readout. Figure 11 shows the relation between cycle time increase and amount of fast interlocks input bytes readout.

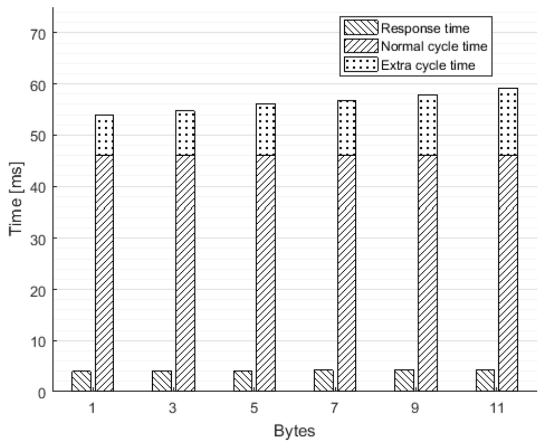


Figure 11: Cycle time extension vs fast interlock bytes read-out.

CONCLUSIONS

A synthesis of the analysis of the different possible solutions together with a description of the main design challenges has been given. The test results confirm that the seamless introduction of fast interlocks in UNICOS-CPC provides a valuable new feature to cope with new challenges in terms of fast reaction in protection equipment control systems. Fully satisfactory reaction times irrespective of the PLC cycle time are achieved based on hardware interrupts, using appropriate hardware input modules and configuring the hardware interrupts accordingly. An additional solution based on cyclic interrupts provides a heightened level of flexibility and a satisfactory compromise between response performance and the simplicity of installation and configuration. With this feature, it is now possible to treat fast interlock events in the UNICOS-CPC framework without additional development by the end user. New control systems with demanding constraints on response time (typically equipment protection control systems) can benefit from the features provided by UNICOS-CPC in terms of programming principle and methods, automatic code generation and visualization capabilities at the supervision layer.

REFERENCES

[1] UNICOS-CPC Team, CERN BE/ICS, "UNICOS-CPC Documentation," https://unicos.web.cern.ch/unicos-cpc

[2] Siemens AG 2008. "S7-300 Instruction List manual for CPU 31xC, CPU 31x, IM 151-7 CPU, IM 151-8 CPU, IM 154-8 CPU, BM 147-1 CPU, BM 147-2 CPU," Order number: 6ES7398-8FA10-8BA0, 6ES7198-8FA01-8BA0, 06/2008, A5E00105517-10, pp. 60-62.

[3] Siemens AG 2008. "S7-300, CPU 31xC and CPU 31x Technical specifications manual," Order number: 6ES7398-8FA10-8BA0, 03/2011, A5E00105475-12, p. 176.