Universidad de Oviedo

Manual del programador de Step 7 del Trabajo Fin de Máster realizado por

Manuel Vázquez Muñiz

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Upgrade of the UNICOS Time Stamp Push Protocol (TSPP) broker to include ultra-fast events

Junio 2017

INDEX

FIGURE INDEX

1. Introduction

1.1. Project identification

- Title: Upgrade of the UNICOS Time Stamp Push Protocol (TSPP) broker to include ultra-fast events
- Author: Manuel Vázquez Muñiz
- Advisor: Víctor Manuel González Suárez
- Co-advisor: Jerónimo Ortolá Vidal
- Date: June 2017
- Organization: CERN

1.2. Project overview

The current project objective is to solve the issue of the fast interlocks (or ultra-fast events) by improving the Time Stamp Push Protocol (TSPP) used to communicate the control and supervision layers. This protocol is used in the framework UNICOS, and this framework should also be modified as to support this new feature.

With this new feature, the organization will be able to fulfil the requirements of the internal clients who need this capability as to have a proper use of their equipment.

1.3. Document overview

This document explains the changes made to the SCL code used in the PLC as to make a functional fast interlock treatment using the UNICOS framework.

For an insight of the full code that is generated with the solution implemented, see the code existent in the example Step 7 application generated.

2. Implementation

Multiple SCL files have been modified as to implement the fast interlock feature. Changes made to them are described in this section.

The implementation of the application is the same in most of its part for both hardware and cyclic interrupts. As such, a general implementation with the common modifications will be described. The specific implementation for each solution will be described in its own subsection.

2.1. General

2.1.1. Program creation

An initial application was created with normal objects, and the code generated was modified as to support the fast interlocks in some of those objects. With those modifications, the validity of the implementations was checked and the response time, was measured. The modifications done to the code are explained via this example application.

The program was generated with 6 digital inputs (3 for the fast interlocks), 6 digital outputs (3 for the fast interlocks), 3 OnOff objects (just for the normal outputs) and a PCO (for the normal processing). An excel document with the specifications of the different objects was created and used to generate the Step 7 application (with all normal objects) as described in the UCPC creation of a Step 7 application document [1].

A WinCC OA application with the existent objects was created for the supervision of the program as described in the UCPC creation of a WinCC OA application document [2] and in the UNICOS creation and configuration of a PVSS WinCC OA application document [3].

The following steps were followed when the application was created:

- The execution of the fast interlock digital inputs was excluded from the function block that executes them (“FB_DI_all”) and from the data block that contains its data (“DB_DI_all”) and included in a different function block and data block for fast interlocks processing (“FB_DI_FI” and “DB_DI_FI”). A function for calling that function block was created (”FC_DI_FI”).
- Same as for digital inputs was done for digital outputs (“FB_DO_FI”, “DB_DO_FI” and “FC_DO_FI”).

- The logic for connecting inputs and outputs (for this approach will just be a direct connection of the input to its respective output) was put directly in the OB, although it could have also been included in a function created and that could be called "FIL" (fast interlock logic).
- Inside the OB for the fast interlock processing (either OB 40 for hardware interrupt or OB 34 for cyclic interrupt), the input byte that included the different fast interlock inputs (connected to inputs 0.3, 0.4 and 0.5) was updated by peripheral addressing. After the execution of the digital input objects, the logic and the digital output objects, the output byte that included the different fast interlock outputs (connected to outputs 0.3, 0.4 and 0.5) was updated by peripheral addressing as well.
- After the processing of the fast interlock events inside the PLC, the part of the TSPP necessary to treat inside the fast interlock event processing OB was called ("FC_Event_FI").

During the testing, some issues were corrected, such as the proper update of the input bits (as not to include input bits of the byte not used for the fast interlocks), and the call of the FI objects inside the main program in OB 1 (as to be able to interact and execute changes inside WINCC OA to the application objects, so that they are consistent with the rest of the objects in the application).

Having tested the first approach, other approaches were done with different changes:

- OnOff objects were included in the fast interlock processing. Just the ones relevant in the fast interlock processing were included, which meant 3 new OnOff objects were created and included in the fast interlock processing ("FC_ONOFF_FI").
- Digital alarms for both fast interlock and normal processing were created (3 for normal processing and 3 for fast interlock processing).
- Including logic for the fast interlock processing as a new module inside the PCO ("PCO_FIL"). Finally the whole relevant PCO was included inside the fast interlock processing, discarding this new function.
- Coordination between the fast interlock PCO and the no fast interlock objects was included trying to avoid the possible concurrency issues treated in section 2.1.4.
- The use of a fast interlock PCO was discarded and the logic will have to be placed inside the dependent logic of the OnOff objects.

The relevant changes were tested to compare the results and decide the best set of objects to implement as a solution for the project.

2.1.2. TSPP treatment

There are 3 types of messages considered in the TSPP protocol, which are shown in Fig. 1.

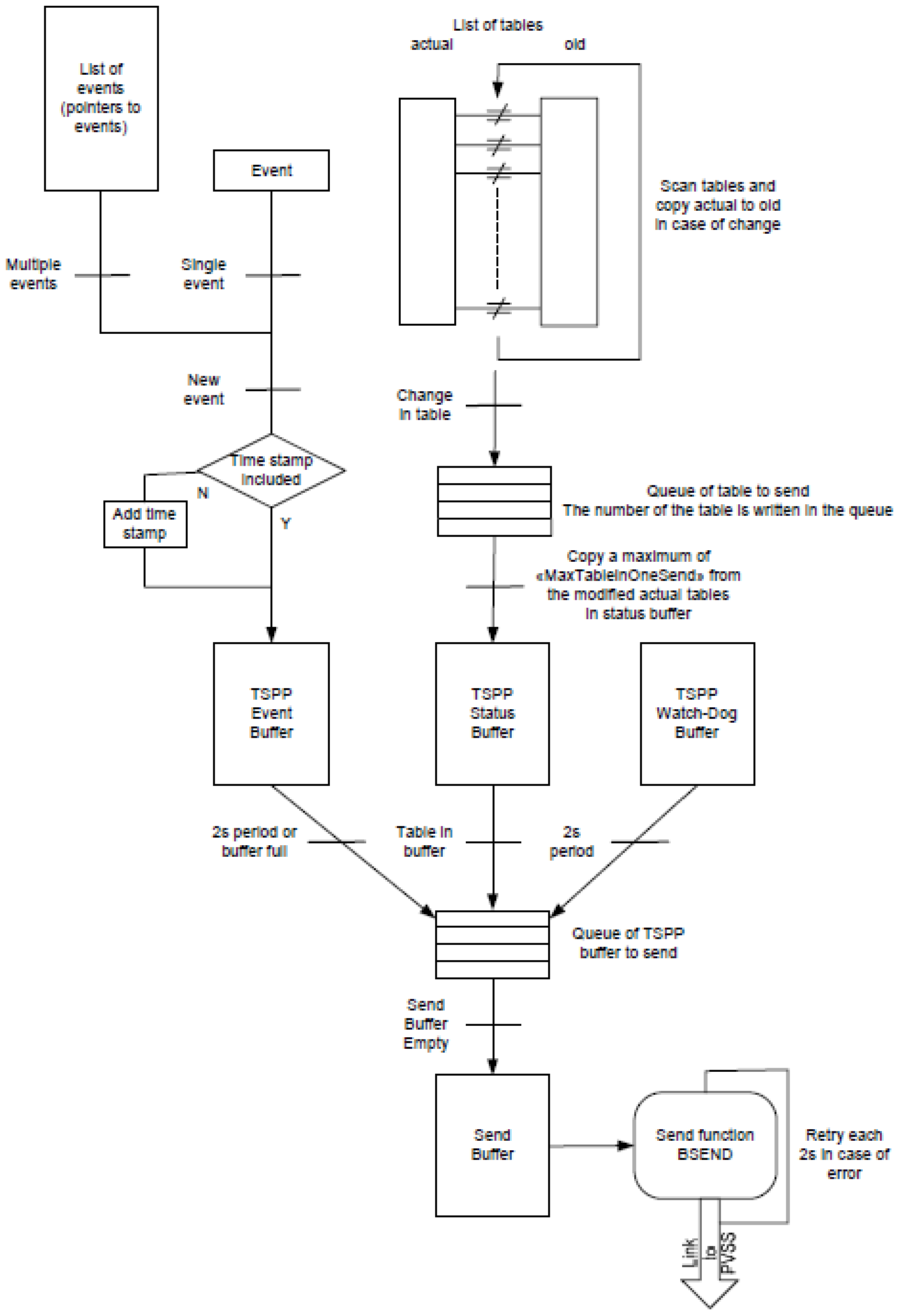


Fig. 1. TSPP messages storage and sending diagram [4]

- Events

Event messages are sent whenever a signal changes inside the status register of a UNICOS object. The event includes information of the previous and the current state of the object. The status register is updated every time the object itself is called. The check of changes is done inside the function "FC_Event" and both the storing and sending of the events are done inside the TSPP processing function block ("TSPP_Unicos_Manager").

- Status tables

Status tables store the current and old status of the objects. They are divided into groups and one group is processed each time the TSPP is called. While processed, the current value of the table is compared to the old value. If there is a change, the full table (of a predefined size) is stored as to send to the WINCC OA. It is both sent and copied to the old table. The current value is updated inside the UNICOS objects, but the checking, storing and sending of the tables are done inside the TSPP processing function block ("TSPP_Unicos_Manager").

- Watchdog

A watchdog event is sent every time an IP is received from the WINCC OA. That makes WINCC OA and the PLC make sure the connection is still alive. Watchdog's storing and sending is done inside the TSPP processing function block ("TSPP_Unicos_Manager").

The ANY pointer inside the TSPP processing is used as to point a part of a data block of a defined size that is going to be copied to the frame to be sent. This pointer is used for doing that process regardless of the type and the size of the data. The Any pointer has the following fields:

- S7 identificator: 10h for Step 7.
- Data type: The type of data to be transferred. For example, 06h for double word.
- Repetition factor: The amount of elements of the data type specified that are pointed (used generally for arrays or structures. 1 for just an element).
- DBNumber: The number of the data block to be transferred.
- Memory area: The area of the memory of the PLC where the specified data is stored. For example, 84 for a data block.
- Address: The start address of the data. The last 3 bits are used to specify the bit position.

The transfer of the data is then done with the BLKMOV standard function block.

Inside the fast interlock processing, the watchdog is not something critical (does not have to do with the objects used). On the other hand, events need to be treated as to set the proper time stamp, and status tables should all be treated as to make sure changes are detected and stored to send, as the tables for the fast interlock objects are mixed with the rest in the normal processing. The sending should still be left to do for the normal processing, as it is not something critical for the fast interlock processing and can lead to errors if there are communication issues as the function is just called once every time an input changes (not cyclically).

Status table changes storing and sending can also be left to do to the normal processing, as it is just a visual information for the SCADA layer and does not need a precise time stamp.

All these changes were done for the implementation of the application for its testing. Some other issues were discovered and corrected in the application during the testing phase.

With this implementation, the events could be resent during the normal execution of the program if the corresponding UNICOS object was not called in between the interrupt and the TSPP processing in OB1. To avoid this issue, the current state of the event was copied to the old one after the TSPP fast interlock processing. The effect of this change can be seen in Fig. 2.

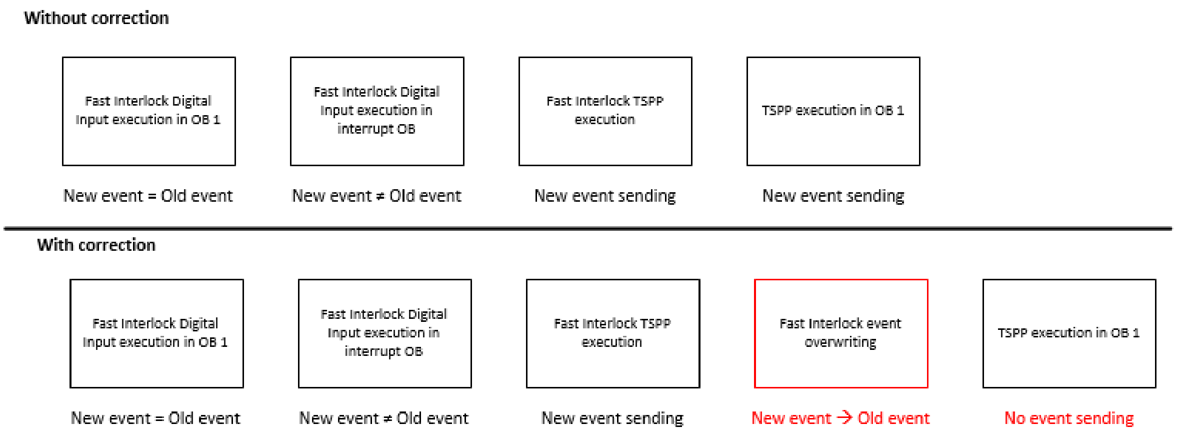

Without correction
Fast Interlock Digital Input execution in OB 1
Fast Interlock Digital Input execution in interrupt OB
Fast Interlock TSPP execution
TSPP execution in OB 1
New event = Old event
New event ≠ Old event
New event sending
New event sending
With correction
Fast Interlock Digital Input execution in OB 1
Fast Interlock Digital Input execution in interrupt OB
Fast Interlock TSPP execution
Fast Interlock event overwriting
TSPP execution in OB 1
New event = Old event
New event ≠ Old event
New event sending
New event → Old event
No event sending

Fig. 2. Event overwriting correction concerning a fast interlock digital input

For the cyclic interrupt, the events could be not sent if the physical input changed and the UNICOS object was executed in OB1 before the interrupt was triggered, as the execution of the same object inside the interrupt would overwrite the change before the TSPP was called. To avoid this issue, the TSPP was called before the rest of the interrupt execution if the physical input change was detected, even though this could mean a rare resend of an event of a fast interlock object that had changed the previous PLC cycle and had not been executed yet. The effect of this change can be seen in Fig. 3.

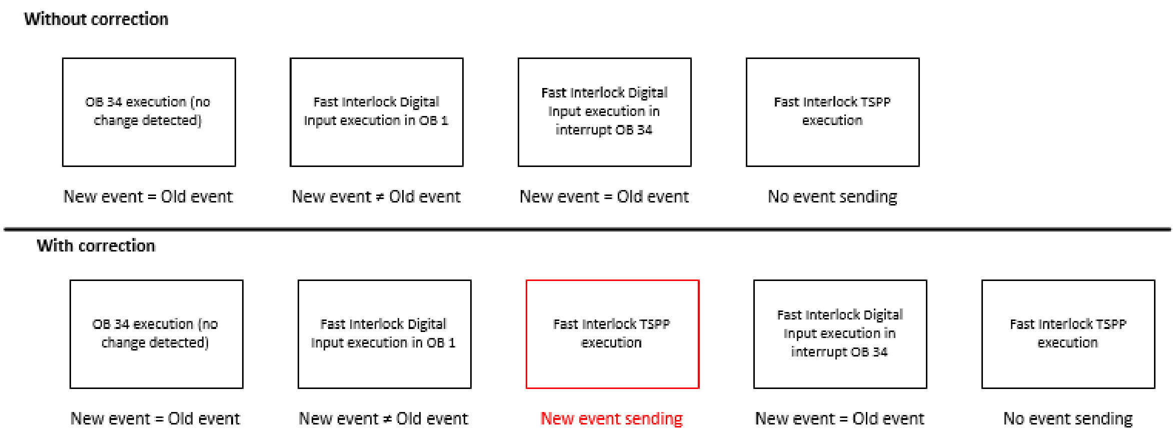

Without correction
OB 34 execution (no change detected)
Fast Interlock Digital Input execution in OB 1
Fast Interlock Digital Input execution in interrupt OB 34
Fast Interlock TSPP execution
New event = Old event
New event ≠ Old event
New event = Old event
No event sending
With correction
OB 34 execution (no change detected)
Fast Interlock Digital Input execution in OB 1
Fast Interlock TSPP execution
Fast Interlock Digital Input execution in interrupt OB 34
Fast Interlock TSPP execution
New event = Old event
New event ≠ Old event
New event sending
New event = Old event
No event sending

Fig. 3. TSPP call for OB 34 correction concerning a fast interlock digital input

2.1.3. Peripheral addressing

The peripheral addressing of both inputs and outputs permits the overwriting of bytes as the minimum unit of information. In the first approach, this was the method used, updating the full input and output bytes necessary. For further approaches, as to update just the necessary bits (only in the input, as the output did not change the rest of the bits), the method shown in Fig. 4 was used.

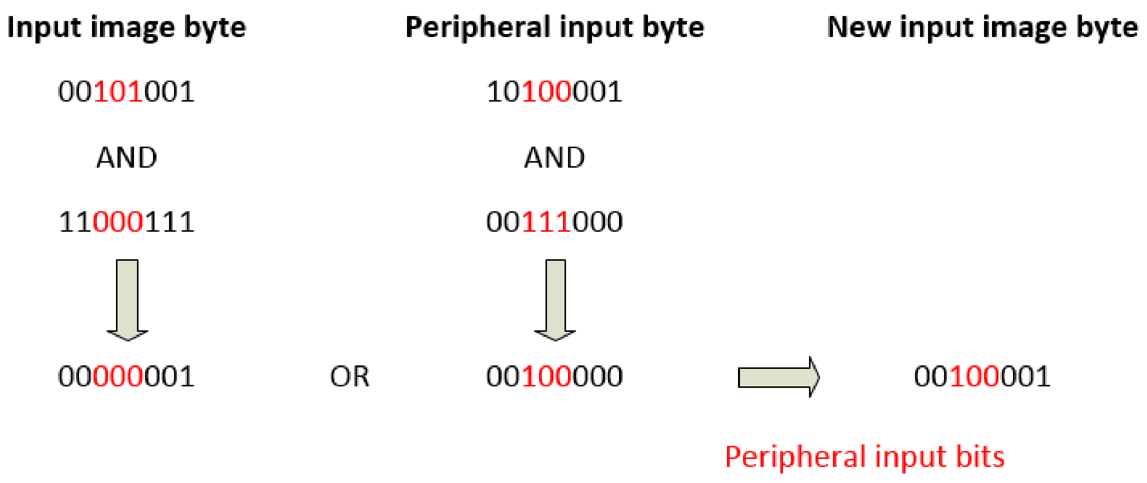


Fig. 4. Peripheral input byte modification in fast interlock program

The current input in the input image is stored in a variable, and then the fast interlock bits that it contains are overwritten with the ones obtained from the peripheral input bit read. This is done for every byte that contains fast interlock bits in the application (in the test application, just one).

2.1.4. Concurrency

The interrupts (either software or hardware) that occur need to share some resources with the main program OB (OB 1). As such, and considering interrupts can happen at any time, it appears the need of a synchronization on the access to those resources as to prevent unexpected misbehaviours of the program, similar to race conditions in typical computer programming with multithreading.

The first moment this problem appears in the current approach of the project is at the TSPP management inside the interrupt. It is necessary to store the events and status table changes as for the TSPP in the main program to send it to WinCC OA. There is not a simple solution as to coordinate both OB executions. As such, the search for some solutions is needed.

There is not a standard function or function block, or any instruction similar to a mutex or a semaphore in computer programming that allows the execution of some code avoiding other parts of the code the access to some protected resources. Nevertheless, there are some standard functions that allow the disabling of both synchronous and asynchronous interrupts that could be useful.

- SFC 39 (DIS_IRT) and SFC 40 (EN_IRT)

With these system functions [5] 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 [5] it is possible to disable (until the end of OB execution, or until enabling the interrupts back) and enable respectively the interrupts, such as hardware or cyclic interrupts. Unlike SFC 39 and SFC 40, interrupts triggered during the interrupt disabling will be queued until the interrupts are enabled again (they are just delayed).

As to not lose any information regarding interrupt occurrence, SFC 41 and SFC 42 seem to be more useful for the purpose of the function. Still though, these system functions disable (even though just temporarily) all interrupts, including not only hardware interrupts, but also others such as cyclic interrupts from the program and cycle time exceeded (for example, OB 35 and OB 32 which are already being used in the UNICOS framework). This should be taken into consideration when using them as to protect the access to shared resources in OB 1.

During the execution of the code inside a cyclic interrupt, the functions SFC 39 and SFC40 can be used as to avoid having some calls of the function queued when the execution of them is exactly the same and is not triggered by an input change.

The state of the program after a synchronization failure happened inside the TSPP is shown in Fig. 5 for WINCC OA [2] and in Fig. 6, Fig. 7 and Fig. 8 for the variables inside the S7 program. This synchronization failure happened when the status table treatment was still being implemented inside the fast interlock treatment.

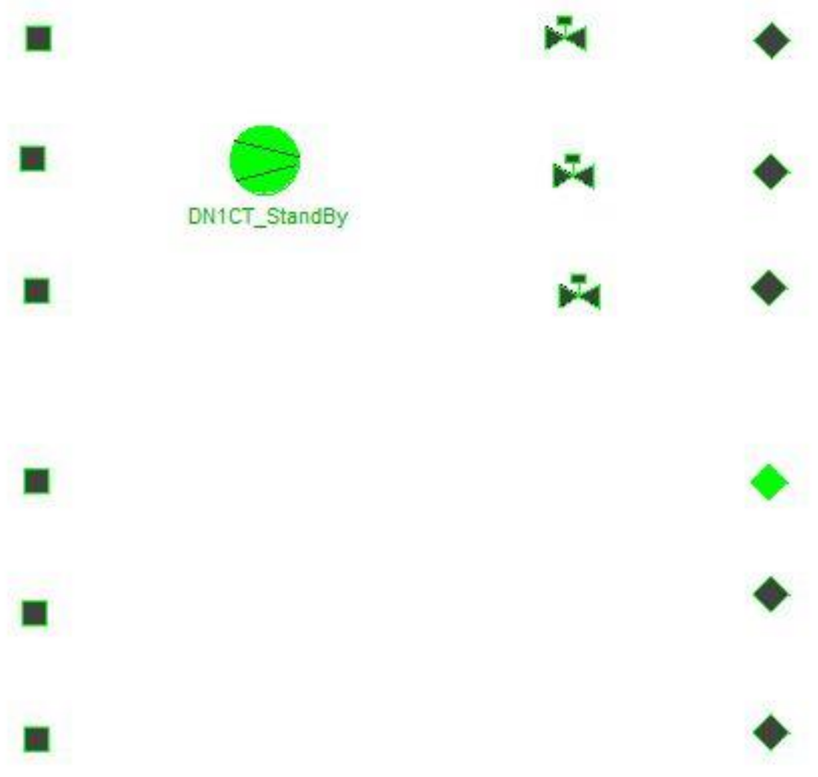
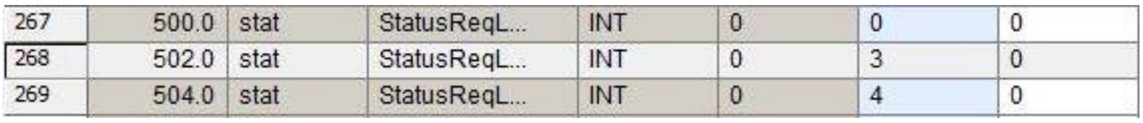


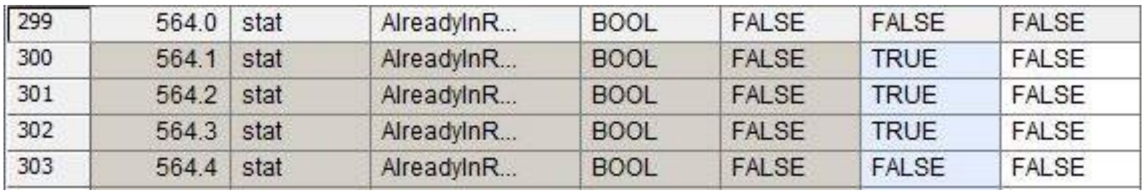
Fig. 5. State of WINCC OA after misbehaviour of the program due to lack of concurrency treatment. From left to right, objects are digital inputs, PCO, OnOffs and digital outputs

The status table 2 has been set to send inside OB40, and overwritten by OB1 inside the main program. As such, no more status tables are sent as the condition for sending status tables is checking the variable “StatusReqList[1]”, and sending it if it’s value is not 0. Still though, status table 2 is marked to be sent (“AlreadyInReqList[2]” is TRUE) and the number of status tables to be sent is 3 (number 2, which has disappeared, number 3 and number 4), although the number of requests is actually 0.



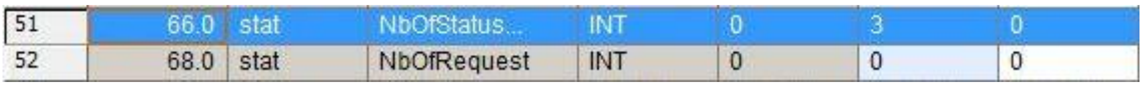
267	500.0	stat	StatusReqL...	INT	0	0	0
268	502.0	stat	StatusReqL...	INT	0	3	0
269	504.0	stat	StatusReqL...	INT	0	4	0

Fig. 6. State of "StatusReqList" variables 1 to 3 (top to bottom) after misbehaviour of the program due to lack of concurrency treatment. Current value is on blue, second last column



299	564.0	stat	AlreadyInR...	BOOL	FALSE	FALSE	FALSE
300	564.1	stat	AlreadyInR...	BOOL	FALSE	TRUE	FALSE
301	564.2	stat	AlreadyInR...	BOOL	FALSE	TRUE	FALSE
302	564.3	stat	AlreadyInR...	BOOL	FALSE	TRUE	FALSE
303	564.4	stat	AlreadyInR...	BOOL	FALSE	FALSE	FALSE

Fig. 7. State of "AlreadyInReqList" variables 1 to 5 after (top to bottom) misbehaviour of the program due to lack of concurrency treatment. Current value is on blue, second last column



51	66.0	stat	NbOfStatus...	INT	0	3	0
52	68.0	stat	NbOfRequest	INT	0	0	0

Fig. 8. State of "NbOfStatusReq" and "NbOfRequest" after misbehaviour of the program due to lack of concurrency treatment. Current value is on blue, second last column

2.1.5. Consistency

Another problem that needs to be treated is the inconsistency of the data used both during the normal PLC cycle and in the fast interlock processing. In general, the data during the fast interlock processing should be consistent during its whole execution, but the data used during the main program of the PLC can be inconsistent if it is not properly treated inside the fast interlock interrupts.

As such, the points of the fast interlock program where the main program could be affected must be detected and taken into consideration as to decide if the inconsistency issues can or not be a problem for the normal functionality of the program.

Some of the inconsistency issues detected appear in the peripheral input and output update. In the peripheral input, some of the normal digital inputs that are connected to the same byte as the fast interlock ones could change their value during the normal processing, which could not be convenient for the processing of the program. This issue can be solved by updating just the peripheral input bits of the fast interlock inputs and leaving the rest as they were before the peripheral update by doing some bit operations between the existent input byte in the input image and the updated input byte read by peripheral input update. Those operations are shown in Fig. 4.

For the peripheral output update, some of the normal output bits connected to the same output byte as the fast interlock ones could be updated in the output of the PLC before the normal output update at the end of the PLC cycle. This known issue is not considered as a critical one and is not corrected in the program.

Finally, the points of interaction between the fast interlock processing and the main one are a point of inconsistency if a fast interlock triggers in between two uses of the same fast interlock variable (for example, if a fast interlock digital alarm is used by two PCOs and the fast interlock event triggers in between the execution of them). This known issue is not either treated as it is not considered to be a critical one and the way of resolving it would imply delaying the PLC interrupts during a possible large time that would correspond to the execution of all the elements where the fast interlock objects are connected to, which could make it inviable to treat the fast interlock events on an affordable amount of time.

2.2. SCL files modifications

In most of the cases, as the processing of the objects is also needed to be done inside the normal processing (not just in the fast interlock treatment), the fast interlock functions have been called inside the normal processing of each device type (for example, the “FB_DI_FI” function block is called inside the “FC_DI” function), while having delayed the interrupts to avoid concurrency issues.

2.2.1. Files modified

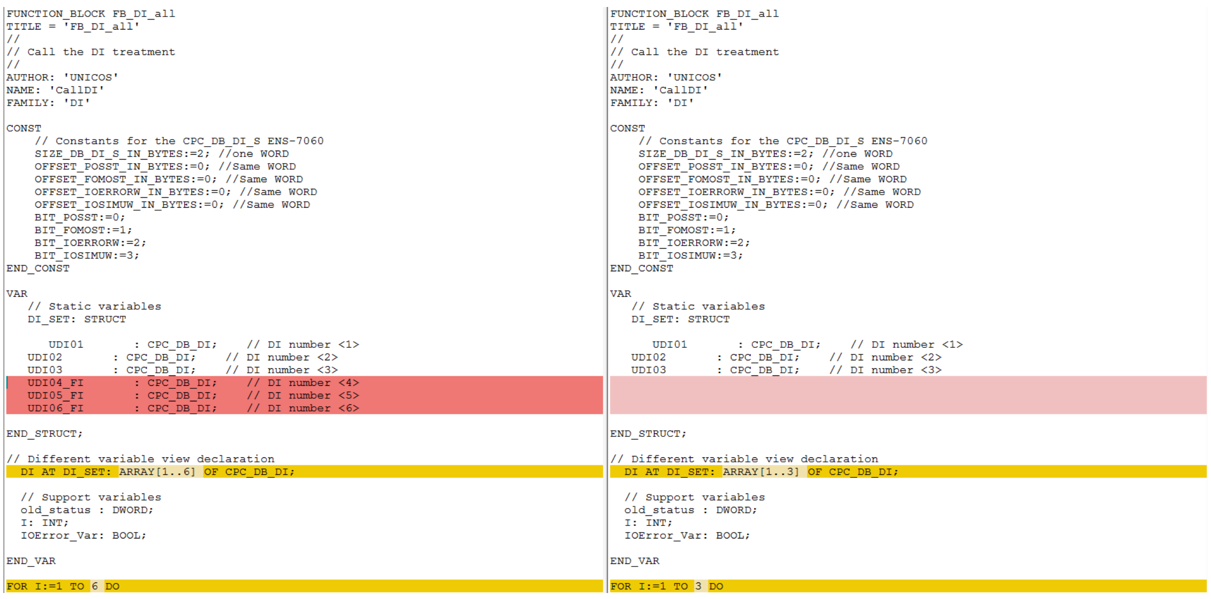
This chapter contains the list of the SCL files modified for each generation tool in UAB.

- Instance generator
    - 4_Compilation_OB
    - COMMUNICATION
    - CPC_TSPP_UNICOS
    - DA
    - DI
    - DO
    - ONOFF
- Logic generator
    - FC_PCO_Logic
    - OnOff04_DL (one file for each fast interlock OnOff. In the example application three files for “OnOff04_DL”, “OnOff05_DL” and “OnOff06_DL”)
    - PCO02_IL (one file for each PCO to which a fast interlock digital alarm is connected. In the example application just one file)

    - PCO02_BL (modified automatically by the templates)
- WinCC OA generator
    - Wincc_oa_db_file (not a SCL file. Modified for the proper DB number for the fast interlock objects)

2.2.2. Digital Input

The digital input file (“DI”) has been modified as to include the fast interlock digital inputs in different blocks than the original ones. Those blocks are the manual request (“DB_DI_ManRequest”), the event status register (“DB_Event_DI”), the binary status both current and old (“DB_bin_status_DI” and “DB_bin_status_DI_old”), and the function block, data block and function to call the digital input objects (“FB_DI_all”, “DB_DI_all” and “FC_DI”). The blocks of the fast interlock are the same changing the “DI” for a “DI_FI”. A sample of the difference for the same specifications file is shown in Fig. 9 for the “FB_DI_all” function block and in Fig. 10 for the new “FB_DI_FI” function block.



```
FUNCTION_BLOCK FB_DI_all
TITLE = 'FB_DI_all'
//
// Call the DI treatment
//
AUTHOR: 'UNICOS'
NAME: 'CallDI'
FAMILY: 'DI'

CONST
    // Constants for the CPC_DB_DI_S ENS-7060
    SIZE_DB_DI_S_IN_BYTES:=2; //one WORD
    OFFSET_POSST_IN_BYTES:=0; //Same WORD
    OFFSET_FOMOST_IN_BYTES:=0; //Same WORD
    OFFSET_IOERRORW_IN_BYTES:=0; //Same WORD
    OFFSET_IOSIMUW_IN_BYTES:=0; //Same WORD
    BIT_POSST:=0;
    BIT_FOMOST:=1;
    BIT_IOERRORW:=2;
    BIT_IOSIMUW:=3;
END_CONST

VAR
   // Static variables
   DI_SET: STRUCT

      UDI01       : CPC_DB_DI;    // DI number <1>
   UDI02       : CPC_DB_DI;    // DI number <2>
   UDI03       : CPC_DB_DI;    // DI number <3>
   UDI04_FI       : CPC_DB_DI;    // DI number <4>
   UDI05_FI       : CPC_DB_DI;    // DI number <5>
   UDI06_FI       : CPC_DB_DI;    // DI number <6>

END_STRUCT;

// Different variable view declaration
  DI AT DI_SET: ARRAY[1..6] OF CPC_DB_DI;

  // Support variables
  old_status : DWORD;
  I: INT;
  IOError_Var: BOOL;

END_VAR

FOR I:=1 TO 6 DO
```

```
FUNCTION_BLOCK FB_DI_all
TITLE = 'FB_DI_all'
//
// Call the DI treatment
//
AUTHOR: 'UNICOS'
NAME: 'CallDI'
FAMILY: 'DI'

CONST
    // Constants for the CPC_DB_DI_S ENS-7060
    SIZE_DB_DI_S_IN_BYTES:=2; //one WORD
    OFFSET_POSST_IN_BYTES:=0; //Same WORD
    OFFSET_FOMOST_IN_BYTES:=0; //Same WORD
    OFFSET_IOERRORW_IN_BYTES:=0; //Same WORD
    OFFSET_IOSIMUW_IN_BYTES:=0; //Same WORD
    BIT_POSST:=0;
    BIT_FOMOST:=1;
    BIT_IOERRORW:=2;
    BIT_IOSIMUW:=3;
END_CONST

VAR
   // Static variables
   DI_SET: STRUCT

      UDI01       : CPC_DB_DI;    // DI number <1>
   UDI02       : CPC_DB_DI;    // DI number <2>
   UDI03       : CPC_DB_DI;    // DI number <3>



END_STRUCT;

// Different variable view declaration
  DI AT DI_SET: ARRAY[1..3] OF CPC_DB_DI;

  // Support variables
  old_status : DWORD;
  I: INT;
  IOError_Var: BOOL;

END_VAR

FOR I:=1 TO 3 DO
```

Fig. 9. Difference between original (left) and new (right) implementation of the “FB_DI_all” function block with fast interlock objects

```
(******* DI EXEC ************************)
FUNCTION_BLOCK FB_DI_FI
TITLE = 'FB_DI_FI'
//
// Call the DI treatment
//
AUTHOR: 'UNICOS'
NAME: 'CallDI'
FAMILY: 'DI'

CONST
    // Constants for the CPC_DB_DI_S ENS-7060
    SIZE_DB_DI_S_IN_BYTES:=2; //one WORD
    OFFSET_POSST_IN_BYTES:=0; //Same WORD
    OFFSET_FOMOST_IN_BYTES:=0; //Same WORD
    OFFSET_IOERRORW_IN_BYTES:=0; //Same WORD
    OFFSET_IOSIMUW_IN_BYTES:=0; //Same WORD
    BIT_POSST:=0;
    BIT_FOMOST:=1;
    BIT_IOERRORW:=2;
    BIT_IOSIMUW:=3;
END_CONST

VAR
   // Static variables
   DI_SET: STRUCT

      UDI04_FI       : CPC_DB_DI;    // DI number <1>
   UDI05_FI       : CPC_DB_DI;    // DI number <2>
   UDI06_FI       : CPC_DB_DI;    // DI number <3>

END_STRUCT;

// Different variable view declaration
  DI AT DI_SET: ARRAY[1..3] OF CPC_DB_DI;

  // Support variables
  old_status : DWORD;
  I: INT;
  IOError_Var: BOOL;

END_VAR

FOR I:=1 TO 3 DO
```

Fig. 10. Extract of the new “FB_DI_FI” function block with fast interlock objects

2.2.3. Digital Output

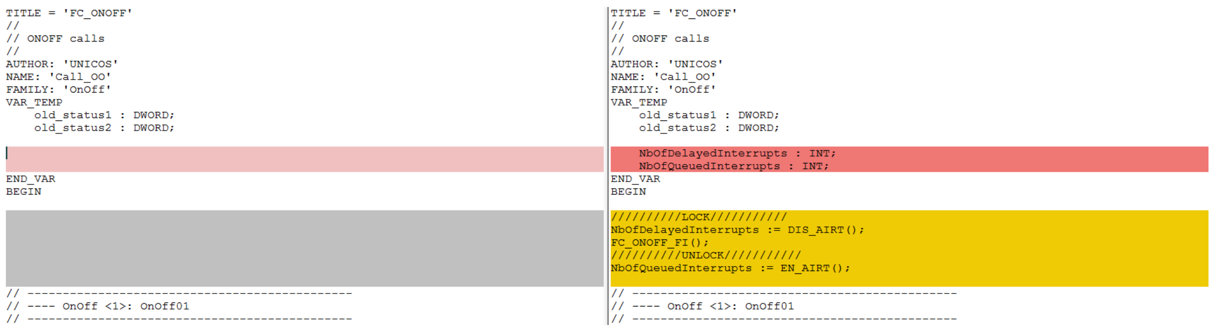
The digital output file (“DO”) has been modified similar to the digital input one with the appropriate names (“DO” instead of “DI”).

2.2.4. Digital Alarm

The digital alarm file (“DA”) has been modified similar to the digital input one with the appropriate names (“DA” instead of “DI”).

2.2.5. OnOff

The manual request, event status register and binary status both current and old contained in the OnOff file (“ONOFF”) have been modified the same way as other objects (digital input, digital output and digital alarm). The OnOff objects have their own data block for each instance though, so there is no need to modify the data block, and no function block is used. As such, only the function ("FC_ONOFF") has been modified as to include the OnOff objects in a different function call ("FC_ONOFF_FI"). A comparison between current UNICOS implementation and the fast interlock one is shown in Fig. 11 for the "FC_ONOFF" function and in Fig. 12 for the new "FC_ONOFF_FI" function for the fast interlock OnOffs.



```
TITLE = 'FC_ONOFF'
//
// ONOFF calls
//
AUTHOR: 'UNICOS'
NAME: 'Call_OO'
FAMILY: 'OnOff'
VAR_TEMP
    old_status1 : DWORD;
    old_status2 : DWORD;


END_VAR
BEGIN





// ---------------------------------------------
// ---- OnOff <1>: OnOff01
// ---------------------------------------------
```

```
TITLE = 'FC_ONOFF'
//
// ONOFF calls
//
AUTHOR: 'UNICOS'
NAME: 'Call_OO'
FAMILY: 'OnOff'
VAR_TEMP
    old_status1 : DWORD;
    old_status2 : DWORD;

    NbOfDelayedInterrupts : INT;
    NbOfQueuedInterrupts : INT;
END_VAR
BEGIN

//////////LOCK//////////
NbOfDelayedInterrupts := DIS_AIRT();
FC_ONOFF_FI();
/////////UNLOCK//////////
NbOfQueuedInterrupts := EN_AIRT();
// ---------------------------------------------
// ---- OnOff <1>: OnOff01
// ---------------------------------------------
```

Fig. 11. Difference between original (left) and new (right) implementation of the "FC_ONOFF" function with fast interlock objects

```
FUNCTION FC_ONOFF_FI : VOID
TITLE = 'FC_ONOFF_FI'
//
// ONOFF calls
//
AUTHOR: 'UNICOS'
NAME: 'Call_OO'
FAMILY: 'OnOff'
VAR_TEMP
    old_status1 : DWORD;
    old_status2 : DWORD;

END_VAR
BEGIN

// ---------------------------------------------
// ---- OnOff <1>: OnOff04
// ---------------------------------------------
old_status1 := DB_bin_status_ONOFF_FI.OnOff04.stsreg01;
old_status2 := DB_bin_status_ONOFF_FI.OnOff04.stsreg02;
old_status1 := ROR(IN:=old_status1, N:=16);
old_status2 := ROR(IN:=old_status2, N:=16);
OnOff04.ManReg01:=DB_ONOFF_FI_ManRequest.ONOFF_Requests[1].ManReg01;

OnOff04.HFOn := DB_DI_FI.DI_SET.UDI04_FI.PosSt;
```

Fig. 12. Extract of the new "FC_ONOFF_FI" function with fast interlock objects

2.2.6. Communication

The communication file ("COMMUNICATION") used as to set the data blocks necessary for the communication and the functions for checking the event changes and to call the TSPP has been modified as to include the new status data blocks and the new functions for checking the fast interlock events and calling the TSPP code for the fast interlocks. The new blocks added have been the

"DB_FIEvent" as to store the event changes, just like "DB_EventData" does for normal communication processing; and "FC_Event_FI" as to check the event changes and to call the TSPP processing of the fast interlock, like "FC_Event" does. A sample of that code is shown in Fig. 13.

```
DATA_BLOCK DB_FIEvent
TITLE = 'EventData'
//
// Contains pointers on evstsreg which have been changed
//
AUTHOR: 'ICE/PLC'
NAME: 'Comm'
FAMILY: 'UNICOS'
STRUCT

        Nb_Event     : INT;
        List_Event    :  ARRAY [1..1000] OF STRUCT
                                                    S7_ID       : BYTE;
                                                    DataType    : BYTE;
                                                    NbOfData    : INT;
                                                    DBNumber    : INT;
                                                    Address     : DWORD;
                                             END_STRUCT;
END_STRUCT;
BEGIN
END_DATA_BLOCK
```

```
(*FUNCTION WHICH GENERATE EVENT FOR evstsreg01 and CALL TS_EVENT_MANAGER*)
FUNCTION FC_Event_FI : VOID
TITLE = 'FC_Event_FI'
//
// FUNCTION WHICH GENERATE EVENT FOR evstsreg01 and CALL TS_EVENT_MANAGER
//
AUTHOR: 'ICE/PLC'
NAME: 'Comm'
FAMILY: 'UNICOS'
VAR
    NewEvent :          BOOL;
    i:                  INT;
    j:                  INT;
    k:                  INT;
    First_obj:          INT;
    ListEventSize:   INT;

    tempsts:         DWORD;
 END_VAR
BEGIN
```

Fig. 13. Extract of the new blocks included in the fast interlock processing

2.2.7. TSPP

The file that contains the TSPP processing ("CPC_TSPP_UNICOS") has been modified as to include the "TSPP_FI" function block which is used inside the fast interlock processing, as well as its "TSPP_FI_DB" data block. Its new content is shown in Fig. 14.

```
FUNCTION_BLOCK TSPP_FI

TITLE = 'TSPP_FI'
//
// Commentaire du bloc ...
//
VERSION : '4.3'  //optimization of the redundant implementation
AUTHOR  : 'UNICOS'
NAME    : 'COMM'
FAMILY  : 'COMMS'

CONST
(* ############################################################################
These parameters MUST be adjusted to fullfill the user requirements *)
MaxNbOfTSEvent      := 100; // ### Const1:To be set by the user
                            // Nb of event wich will trig the send to WinCCOA
SpareEventNumber    := 50;  // ### Const2:To be set by the user
                            // spare places to be able to continue to ../..
                            // record event when the send function is buzy
Word_in_one_TSEvent := 2;   // ### Const3:To be set by the user
                            // Number of data word in one Event
MaxStatusTable      := 11; // ### Const4:To be set by the user
                            // Max number of status tables (the length of one
                            // table is 200 bytes without the header)
MaxTableInOneSend := 11; // ### Const5:To be set by the user
                            //(MAX value S7-400:250  S7-300:100)
                            // Max nb of status tables in one TSPP message
                            // Must lower or equal to MaxStatusTable
Redundant_PLC := FALSE;      // Const6:To be set by the user
                            // Using a S7-400H PLC in Redundance mode
ListEventSize := 1000;      // ### Const7:To be set by the user
                            // Size of the Event List
(*###########################################################################*)
```

Fig. 14. Extract of the new “TSPP_FI” function block used for the fast interlock TSPP processing

The normal function block “TSPP_UNICOS_Manager” has also been modified as to delay the interrupts in the parts of the code where concurrency issues could appear.

2.2.8. OB Compilation

The file that compiles the different OBs (“4_Compilation_OB”), as well as some other functions, has been modified as to include the fast interlock processing. That processing is done differently in both hardware and cyclic interrupts, although the schedule is the same for both in most of its part. First, the peripheral inputs are read, then the digital input objects are executed. After that, the execution of the logic is done (as to set the inputs of the alarms), and after the alarms are executed, the logic is executed once again (to set the output of the alarms). After this process, the OnOff objects are executed. Then the digital outputs are executed and set to the periphery right after. Finally, the TSPP processing inside the fast interlocks is called.

Just the events are treated inside this TSPP processing.

- Hardware interrupts

The OB 40 has been included in the file with the fast interlock treatment. A function and a function block (with its data block) have also been implemented as to get and set the peripheral inputs and outputs of the fast interlock event. The execution of the OB 40 is shown in Fig. 15.

```
(*OB40******************************************************************************)
ORGANIZATION_BLOCK OB40
TITLE = 'Hardware Interrupt'
//
// Called after a hardware interrupt trigger
//
AUTHOR: 'ICE/PLC'
NAME: 'OB40'
FAMILY: 'UNICOS'
VAR_TEMP
    (*UNUSED VARIABLE*)
    UNUSED_VARIABLE : ARRAY [0..20] OF BYTE;

END_VAR
BEGIN
    (*Update inputs*)
    FB_FI_PIU.DB_FI_PIU();   // Peripheral input update

    (*Getting inputs*)
    FC_DI_FI();     // Digital inputs

    FC_FI_LOGIC();
    FC_DA_FI();    // DIGITAL ALARM objects

    (*Process Logic Control*)
    FC_FI_LOGIC();  // Process Control object logic

    (*UNICOS objects calculating*)
    FC_ONOFF_FI();     // OnOff objects

    (*Setting Outputs*)
    FC_DO_FI();     //Digital Outputs

    (*Update outputs*)
    FI_POU();    // Peripheral output update

    //Calling TS_Event_Manager for TSPP communication with the SCADA
    FC_Event_FI();

END_ORGANIZATION_BLOCK
```

Fig. 15. OB 40 execution code

- Cyclic interrupts

The OB 34 is included in the file with the fast interlock treatment. A function and a function block (with its data block) have also been implemented as to get and set the peripheral inputs and outputs of the fast interlock event. The different objects after the peripheral input update are only executed if there has been a change in the input signals, and the cyclic interrupts are disabled while the OB is executed in that case. The execution described is preceded by a TSPP call. The execution of this OB is the one shown in Fig. 16.

```
(*OB34*******************************************************************************)
ORGANIZATION_BLOCK OB34
TITLE = 'Cyclic Interrupt'
//
// Called every user-defined cyclic time
//
AUTHOR: 'ICE/PLC'
NAME: 'OB34'
FAMILY: 'UNICOS'
VAR_TEMP
    (*UNUSED VARIABLE*)
    UNUSED_VARIABLE : ARRAY [0..20] OF BYTE;
    Dis_error : INT;
    En_error : INT;
END_VAR
BEGIN
    (*Update inputs*)
    FB_FI_PIU.DB_FI_PIU();   // Peripheral input update
    IF (DB_FI_PIU.Inp_change) THEN
    //////////LOCK//////////
    Dis_error := DIS_IRT(MODE := B#16#02, OB_NR := 34);

    //Calling TS_Event_Manager for TSPP communication with the SCADA
    FC_Event_FI();

    (*Getting inputs*)
    FC_DI_FI();    // Digital inputs

    FC_FI_LOGIC();
    FC_DA_FI();   // DIGITAL ALARM objects

    (*Process Logic Control*)
    FC_FI_LOGIC();  // Process Control object logic

    (*UNICOS objects calculating*)
    FC_ONOFF_FI();    // OnOff objects

    (*Setting Outputs*)
    FC_DO_FI();    //Digital Outputs

    (*Update outputs*)
    FI_POU();   // Peripheral output update

    //Calling TS_Event_Manager for TSPP communication with the SCADA
    FC_Event_FI();
    //////////UNLOCK//////////
    En_error := EN_IRT(MODE := B#16#02, OB_NR := 34);
    END_IF;
END_ORGANIZATION_BLOCK
```

Fig. 16. OB 34 execution code

2.2.9. Logic

The logic is called from the FC_PCO_LOGIC function (in the “FC_PCO_Logic” file) in a normal UNICOS application. For the fast interlock, the file where this function is defined has been modified as to include another function (“FC_FI_LOGIC”) which executes the fast interlock logic, which is the dependent logic of the OnOffs selected as fast interlock.

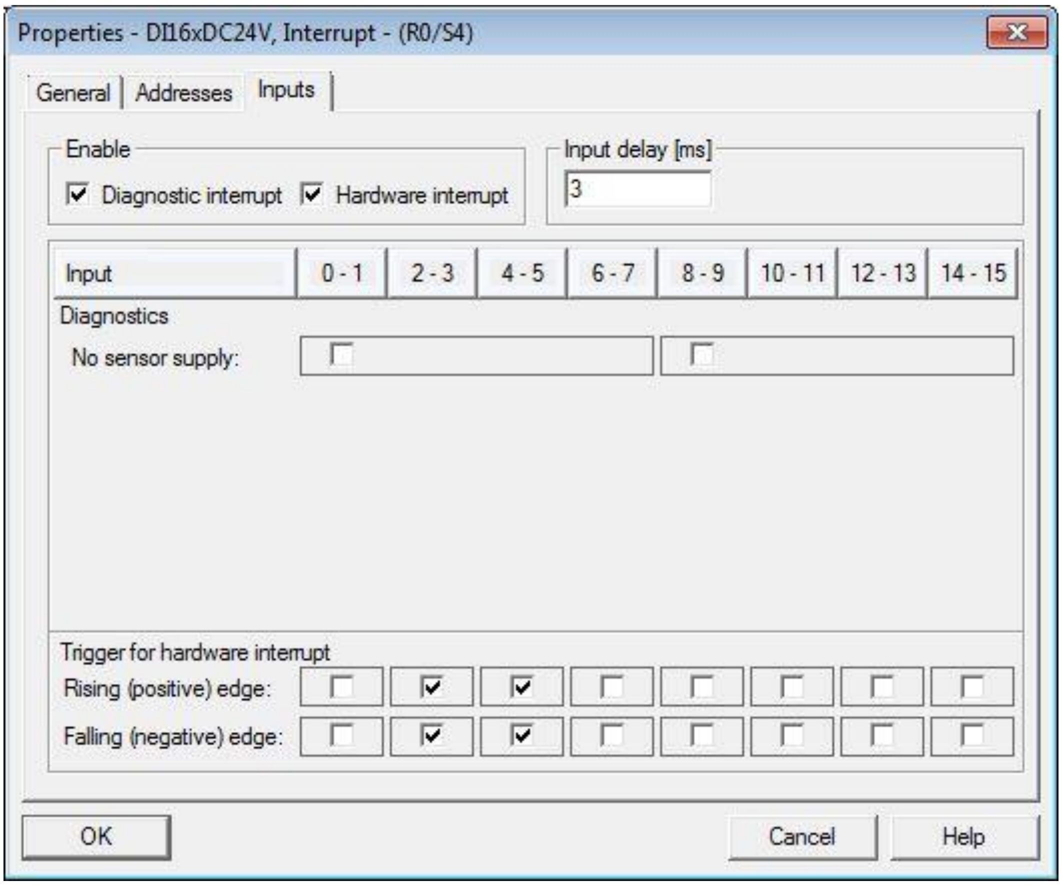
The function is shown in Fig. 17.

```
FUNCTION FC_FI_LOGIC : VOID
TITLE = 'FC_FI_LOGIC'
//
// Call FI DL sections
//
AUTHOR: 'ICE/PLC'
NAME: 'Logic'
FAMILY: 'UNICOS'
BEGIN
// Calling the DL of the FI devices with no master
// Calling the DL of the Dependent Devices for PCO01
// Calling the DL of the Dependent Devices for PCO02
// Calling the DL of the Dependent Devices for NO_Master
OnOff04_DL();
OnOff05_DL();
OnOff06_DL();
END_FUNCTION
```

Fig. 17. Function to execute the fast interlock logic contained in the OnOff dependent logic

2.3. Hardware interrupt

For hardware interrupts handling, the OB 40 is called. This OB is defined including the elements described in section 2.1.1. The digital input module must be configured as to support hardware interrupts in both rising and falling edges for the fast interlock bit inputs (2-3 and 4-5, as they are configured in groups of 2). The diagnostic interrupts are also enabled as to check the input module errors in the PLC diagnostic buffer. The input delay is set to 3 ms by default. The configuration of the digital input module is the one shown in Fig. 18.


Properties - DI16xDC24V, Interrupt - (R0/S4)

General | Addresses | Inputs

Enable: [✓] Diagnostic interrupt [✓] Hardware interrupt

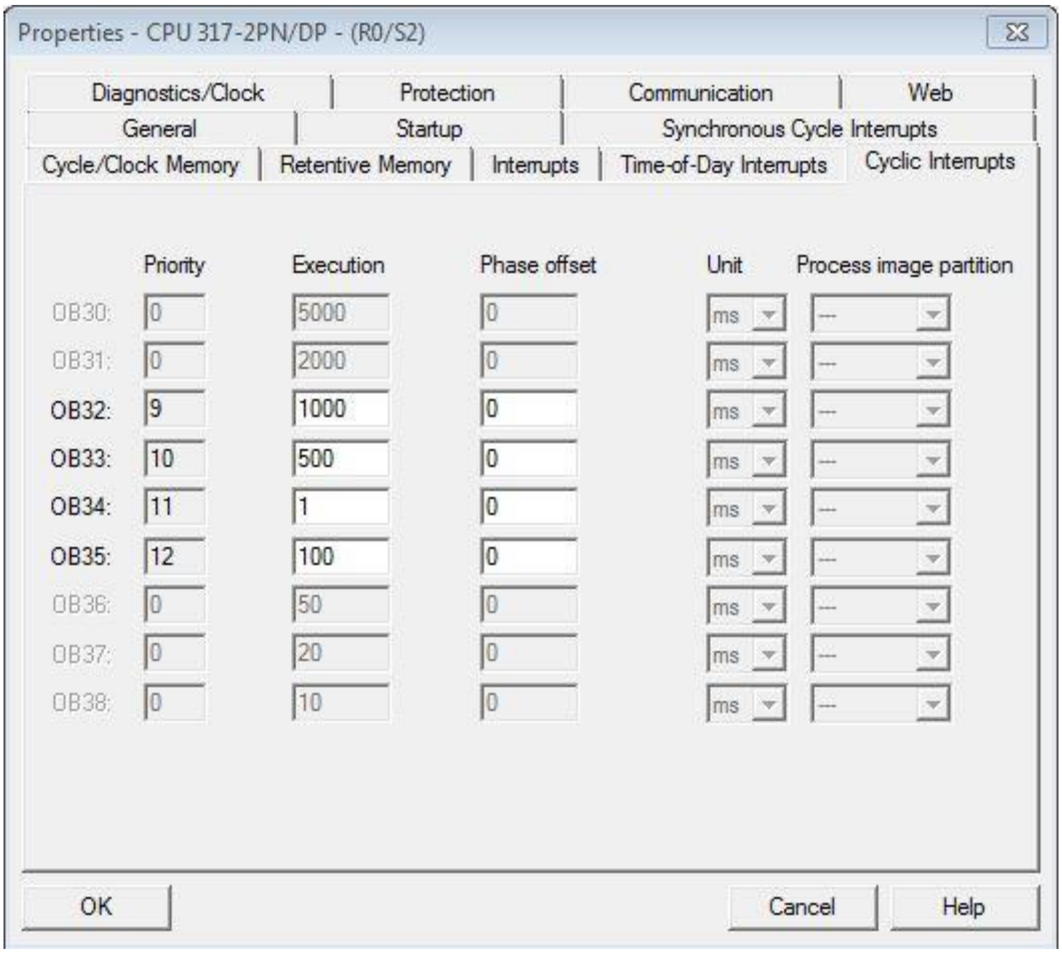
Input delay [ms]: 3

Input	0 - 1	2 - 3	4 - 5	6 - 7	8 - 9	10 - 11	12 - 13	14 - 15
Diagnostics								
No sensor supply:	☐				☐			
Trigger for hardware interrupt								
Rising (positive) edge:	☐	✓	✓	☐	☐	☐	☐	☐
Falling (negative) edge:	☐	✓	✓	☐	☐	☐	☐	☐

Fig. 18. Digital Input module configuration for hardware interrupts

2.4. Cyclic interrupt

For cyclic interrupts handling, the OB 34 is called (as OB 35 is already in use in the framework). This OB is defined including the elements described in section 2.1.1. As soon as the Inputs are updated, they are compared to their previous state as to decide if the rest of the OB processing is necessary (as it would be a waste of CPU time to process them without changes having appeared in the inputs). The CPU is configured to support cyclic interrupts of OB 34 every 1 ms (shortest time possible). The input delay in the digital input module is set to 3 ms by default. The configuration of the CPU is the one shown in Fig. 19.


Properties - CPU 317-2PN/DP - (R0/S2)

Diagnostics/Clock | Protection | Communication | Web
General | Startup | Synchronous Cycle Interrupts
Cycle/Clock Memory | Retentive Memory | Interrupts | Time-of-Day Interrupts | Cyclic Interrupts

	Priority	Execution	Phase offset	Unit	Process image partition
OB30:	0	5000	0	ms	---
OB31:	0	2000	0	ms	---
OB32:	9	1000	0	ms	---
OB33:	10	500	0	ms	---
OB34:	11	1	0	ms	---
OB35:	12	100	0	ms	---
OB36:	0	50	0	ms	---
OB37:	0	20	0	ms	---
OB38:	0	10	0	ms	---

Fig. 19. CPU configuration for cyclic interrupt OB 34

3. Results

For acquiring the response time, measures have been made with the use of the oscilloscope cursors capability, as shown in Fig. 20. In general, response times have been tested from the rising edge of the input to the rising edge of the output, although falling edges would have also been valid for the measures, as the hardware interrupts were configured for both edges.

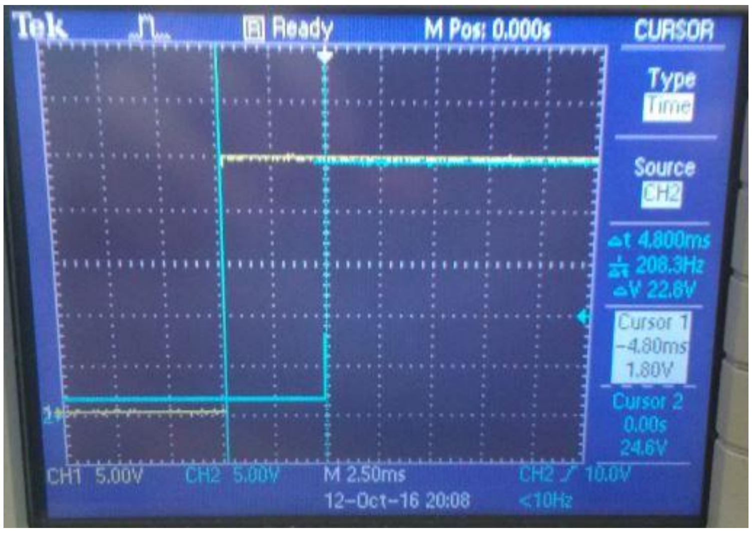
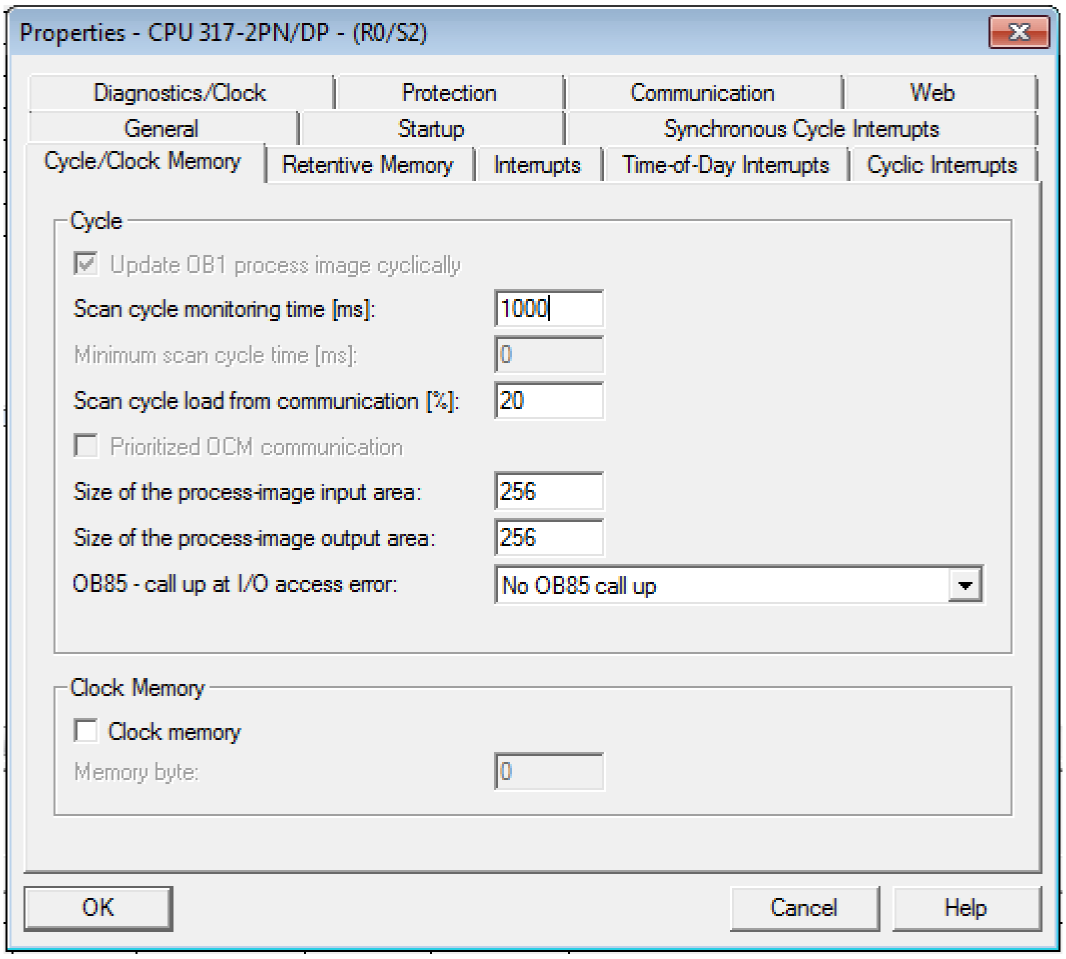


Fig. 20. Oscilloscope cursors used for measurement

Different measures have been made for both the normal and the fast interlock response times with different PLC cycle times [6]. As to enlarge the PLC cycle time, a loop of different sizes has been added each time as to reach different PLC cycle time values.

The PLC maximum cycle time permitted has been configured in the Hardware configuration of the CPU inside the Step 7 software, as shown in Fig. 21. This time has been set to 1000 ms as to avoid any issue concerning long PLC cycle times.


Properties - CPU 317-2PN/DP - (R0/S2)

Cycle

[✓] Update OB1 process image cyclically

Scan cycle monitoring time [ms]: 1000

Minimum scan cycle time [ms]: 0

Scan cycle load from communication [%]: 20

[ ] Prioritized OCM communication

Size of the process-image input area: 256

Size of the process-image output area: 256

OB85 - call up at I/O access error: No OB85 call up

Clock Memory

[ ] Clock memory

Memory byte: 0

Fig. 21. Maximum cycle time configuration

Results for the normal response times using hardware interrupts are shown in Fig. 22. Values are present in the expected theoretical range.

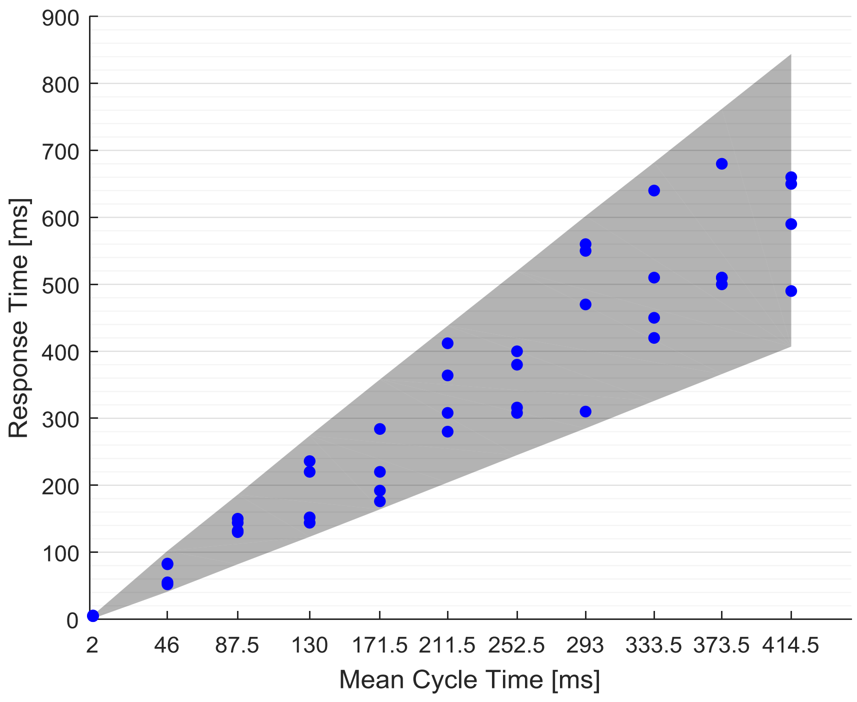


Fig. 22. Response time for the normal input/output using hardware interrupts (theoretical range in grey)

Results for the fast interlock response times using hardware interrupts are shown in Fig. 23. It has been considered a small discrepancy since the input delay had been set to 3 ms and some values were measured below that amount.

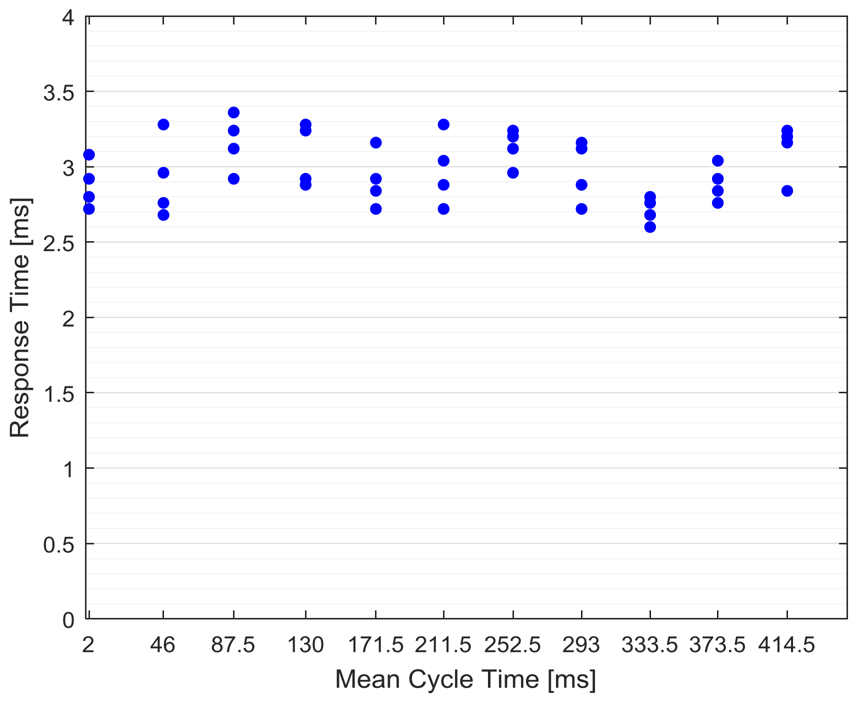


Fig. 23. Response time for the fast interlock input/output using hardware interrupts

In Fig. 24, response times for the fast interlocks using just digital input, digital output objects and processing (as well as TSPP managing), as well as with OnOff objects and with OnOff, digital alarms and PCO (interlocks and instantiation) using hardware interrupts are shown.

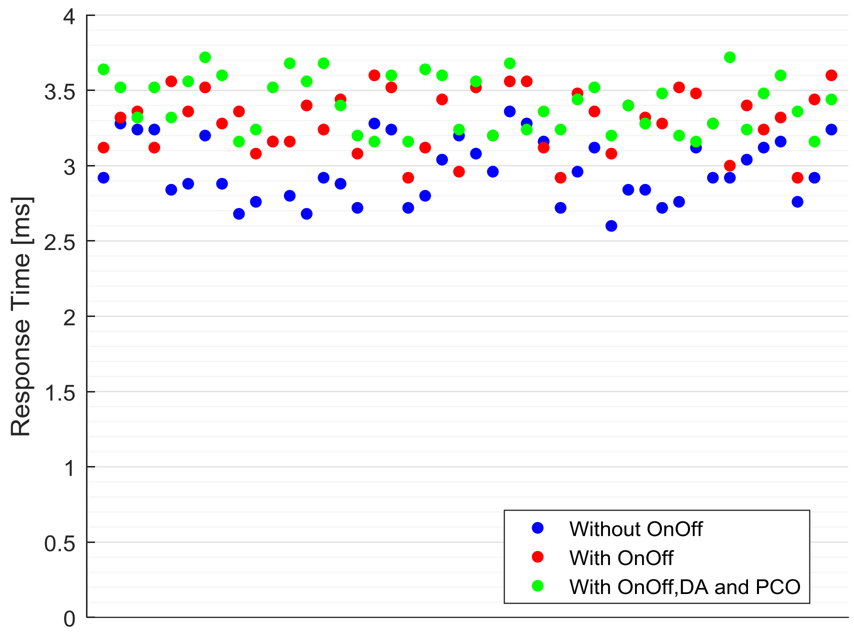


Fig. 24. Response time for the fast interlock input/output using different UNICOS objects and hardware interrupts

Same measures for cyclic interrupts are shown in Fig. 25 and Fig. 26. A comparison of hardware interrupt response times and cyclic interrupt response times (both without the OnOff objects) is shown in Fig. 27.

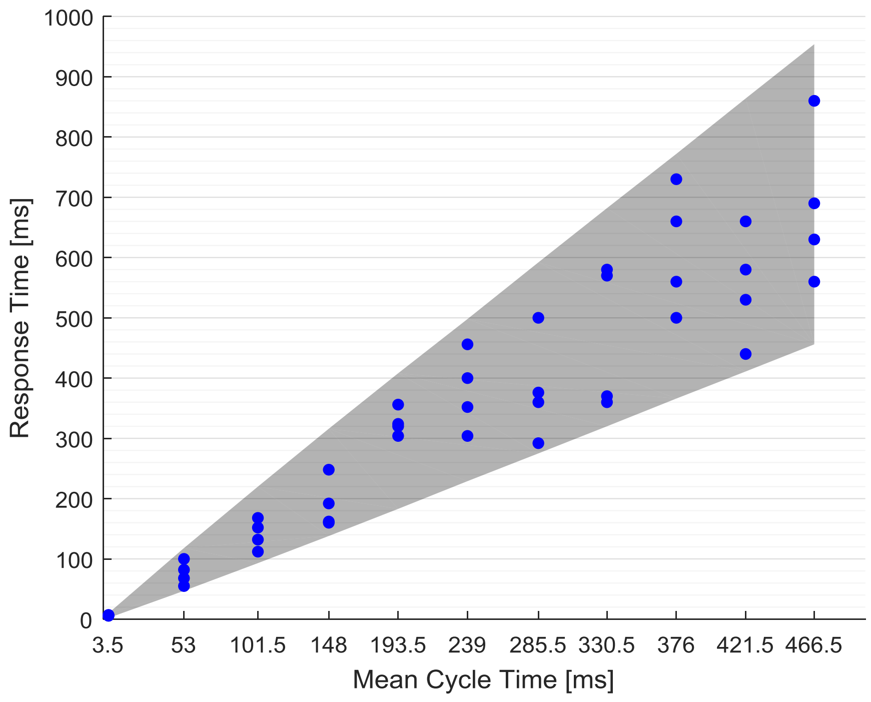


Fig. 25. Response time for the normal input/output using cyclic interrupts (theoretical range in grey)

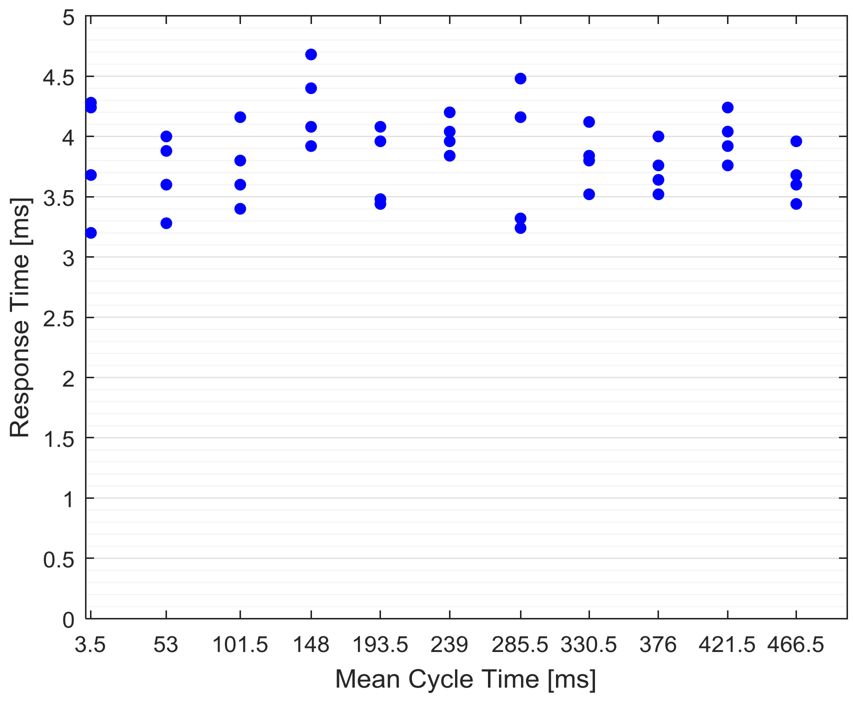


Fig. 26. Response time for the fast interlock input/output using cyclic interrupts

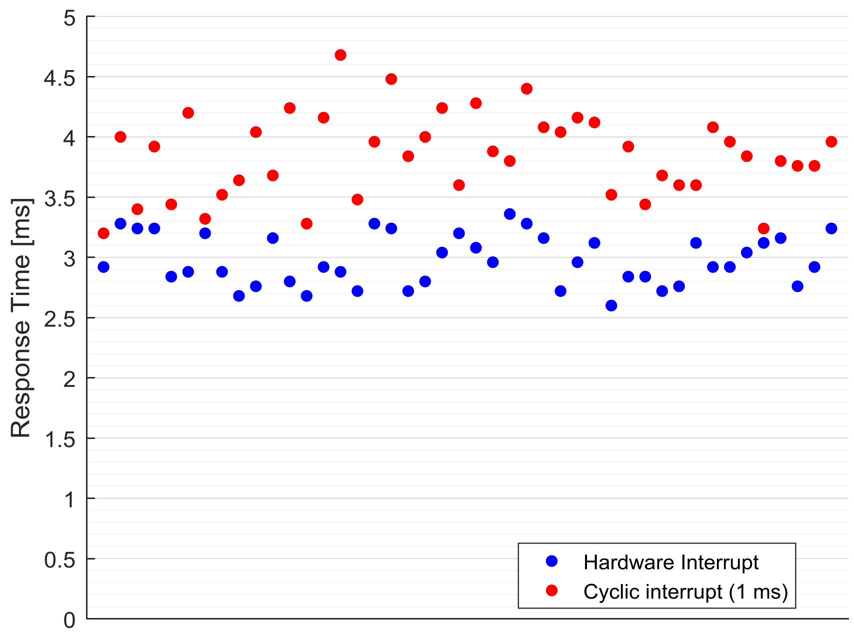
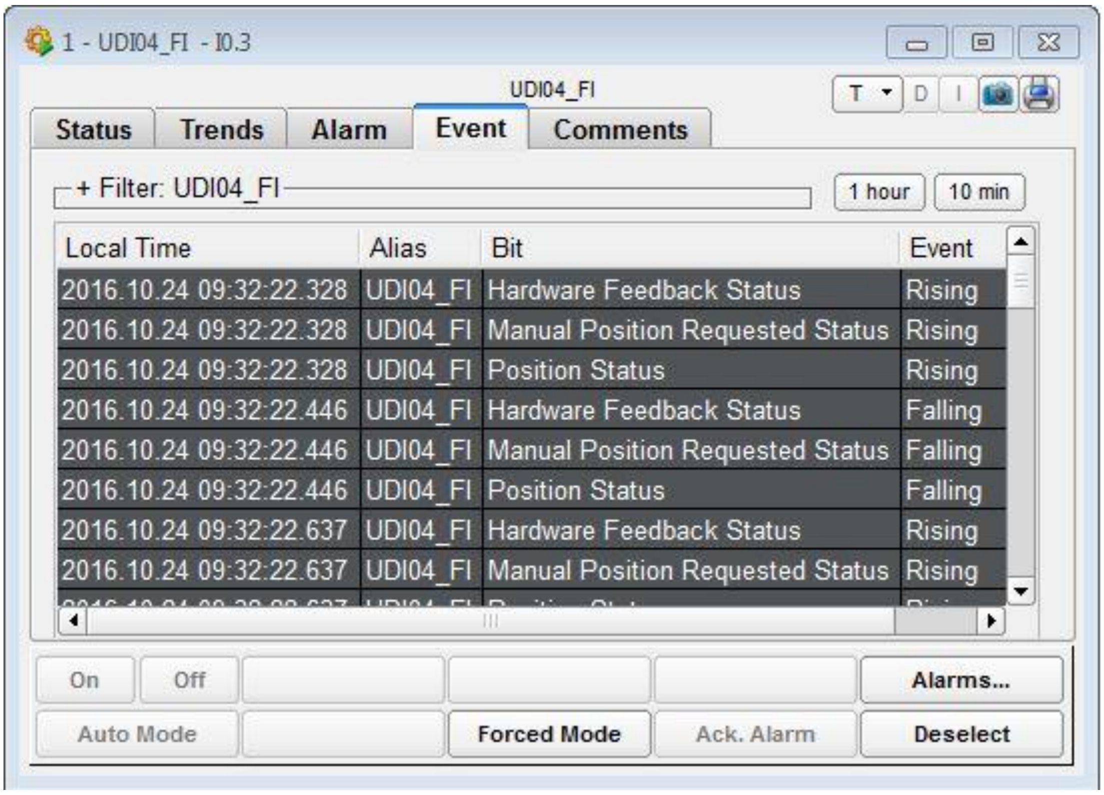


Fig. 27. Response time for the fast interlock input/output

The proper reception of TSPP messages has been made using WINCC OA [2] in the computer where the Step 7 program was created. The reception of changes in the status tables has just been checked by sight in the objects inside the application configured, whereas events have been checked inside the faceplate of each object, as shown in Fig. 28 for a fast interlock digital input.



Local Time	Alias	Bit	Event
2016.10.24 09:32:22.328	UDI04_FI	Hardware Feedback Status	Rising
2016.10.24 09:32:22.328	UDI04_FI	Manual Position Requested Status	Rising
2016.10.24 09:32:22.328	UDI04_FI	Position Status	Rising
2016.10.24 09:32:22.446	UDI04_FI	Hardware Feedback Status	Falling
2016.10.24 09:32:22.446	UDI04_FI	Manual Position Requested Status	Falling
2016.10.24 09:32:22.446	UDI04_FI	Position Status	Falling
2016.10.24 09:32:22.637	UDI04_FI	Hardware Feedback Status	Rising
2016.10.24 09:32:22.637	UDI04_FI	Manual Position Requested Status	Rising

Fig. 28. Events received inside digital input 4 (fast interlock)

It has been detected that sometimes events reception is done unordered in WINCC OA objects faceplates (more specifically during fast input state changes, especially with long PLC cycle times) although time stamping is properly set. This fact is not considered a big issue, and is supposed to be a SCADA layer problem. As such, it has been ignored for the realization of the current project.

3.1. Decision

With the conclusions extracted from the results obtained for the different possibilities, the final decision has been to support both hardware and software (cyclic) interrupts for the fast interlock feature, by selecting the desired type in the specifications file.

The UNICOS objects that will be used inside the fast interlock processing will be the digital inputs, digital alarms, OnOffs and digital outputs. These objects will have a new column in the specifications file as to be selected for the fast interlock processing.

4. Acronyms

CERN	European Organization for Nuclear Research
UNICOS	Unified Industrial Control Systems
UAB	UNICOS Application Builder
CPC	Continuous Process Control
BE-ICS-PCS	Beams department, Industrial Controls and Safety group, Process Control Systems Section
FI	Fast Interlock
TSPP	Time Stamp Push Protocol
PLC	Programmable Logic Controller
ST	Structured Text
SCL	Structured Control Language
OB	Organization Block
FB	Function Block
FC	Function
SFB	Standard Function Block
SFC	Standard Function
PII	Peripheral Image of Inputs
PIO/PIQ	Peripheral Image of Outputs
IEC	International Electrotechnical Commission
SCADA	Supervisory Control And Data Acquisition
WINCC OA	WinCC Open Architecture
DI	Digital Input
DA	Digital Alarm
DO	Digital Output
PCO	Process Control Object

5. Documents of the project

The current project has been elaborated in multiple documents that describe a certain part of the project.

1. Report: General description of the project. Objectives and conditions for its test. Conclusion from the realization of the project and future works.
2. Planning and budget: Schedule of the different tasks that compound the project and price of the resources used.
3. Step 7 programmer manual: Modifications to the code of the UNICOS applications to support the fast interlock capability. Results obtained from these modifications.
4. Templates programmer manual: Modifications to the code of the templates and of the plugin used to generate the SCL files used in the PLC.
5. User manual: Steps to create a fast interlock UNICOS application.
6. Templates code: Modified template files inside the resources folder of an application and of the UAB plugin.
7. Datasheets: Datasheets of the devices used to research and test the solution for the fast interlocks issue.

Attachments.

1. Attachment 1: Fast interlock application example.

6. Bibliography

[1] S. Izquierdo Rosas and M. Zimny, “Creation of a Siemens S7 UNICOS-CPC 6 Application,” 16 October 2015. [Online]. Available: https://edms.cern.ch/ui/file/1228441/1.8.0/Procedure_S7-UCPC_Application.pdf. [Accessed 15 December 2016].

[2] S. Izquierdo Rosas, A. Merezhin and M. Bes, “Creation of a WINCC OA 3.11SP1 - CPC 6 Application,” 28 October 2015. [Online]. Available: https://edms.cern.ch/ui/file/1228441/1.9.1/Procedure_WinCCOA-UCPC_Application.pdf. [Accessed 15 December 2016].

[3] M. Boccioli, “Creation and configuration of a PVSS (WINCC OA) UNICOS application,” 21 September 2012. [Online]. Available: https://unicos.web.cern.ch/sites/unicos.web.cern.ch/files/page/unicos-wincc-oa-documentation/winccoa-unicos-application-setup.pdf. [Accessed 23 January 2017].

[4] J. Brahy, “TSPP UNICOS MANAGER,” 14 May 2005. [Online]. Available: https://svn.cern.ch/reps/en-ice-svn/trunk/tools/UNICOS/CPC/TSPP/DriverS7/Doc/TSPP_description.pdf. [Accessed 15 December 2016].

[5] “PLC dev,” [Online]. Available: http://www.plcdev.com/s7_library_functions. [Accessed 11 November 2016].

[6] Siemens, “How do you process the data from the temporary variables of an organization block in order to determine the cycle time of OB1, for example?,” 12 February 2007. [Online]. Available: https://support.industry.siemens.com/cs/document/245243/how-do-you-process-the-data-from-the-temporary-variables-of-an-organization-block-in-order-to-determine-the-cycle-time-of-ob1-for-example-?dti=0&lc=en-WW. [Accessed 11 November 2016].