

SICK **RFH5xx Function Block**

SICK RFH5xx IO-Link function block
for Mitsubishi PLCs
(GX-Works 3)



Version history

Block Version	Date	Remark
V1.0	11.07.2023	Initial version

Table of content

1 About this document.....	3
1.1 Function of this document	3
1.2 Target group	3
2 General information	4
3 Hardware Configuration.....	5
3.1 Supported PLCs.....	5
3.2 RJ71EIP91 Configuration.....	5
4 Function block.....	6
4.1 Block specifications.....	6
4.2 Process data handover to the FB	7
4.3 Operation of the function block.....	8
4.4 Data type description	8
4.4.1 AntennaField	8
4.4.2 ReadUID	9
4.4.3 ReadUMem	9
4.4.4 WriteUMem	9
4.5 Adjust maximum data length	11
Adjustment at the function block interface:	11
4.6 Behavior when error occurs	11
5 Parameter.....	12
6 Error description	14
7 Example	15
7.1.1 Process data exchange and function block call up.....	15

1 About this document

Please read this chapter carefully before you start working with this technical information and the FB_SICK_RFH5xx_IOL function block.

1.1 Function of this document

This technical information describes how to use the FB_SICK_RFH5xx_IOL function block. It is used for guiding technical personnel working for the machine manufacturer / operator in project planning and commissioning.

1.2 Target group

This technical information is aimed for specialists, such as technicians and engineers.

2 General information

The function block “FB_SICK_RFH5xx_IOL” simplifies the use of RFH5xx RFID interrogators on Mitsubishi R04CPU PLCs. The device has to be embedded into the IO-Link surrounding of the PLC-Controller.

The function block enables reading and writing tag data as well as controlling the RFHxx device via the cyclic IO-Link process data channel.

Functionalities:

- Read UID
- Write user memory up to 512 bytes
- Read user memory up to 512 bytes
- Switch RF-Field power on/off
- Tag present indication
- Antenna state
- Information about RSSI value

Figure 1 shows the concept behind the RFH5xx IO-Link PLC integration.

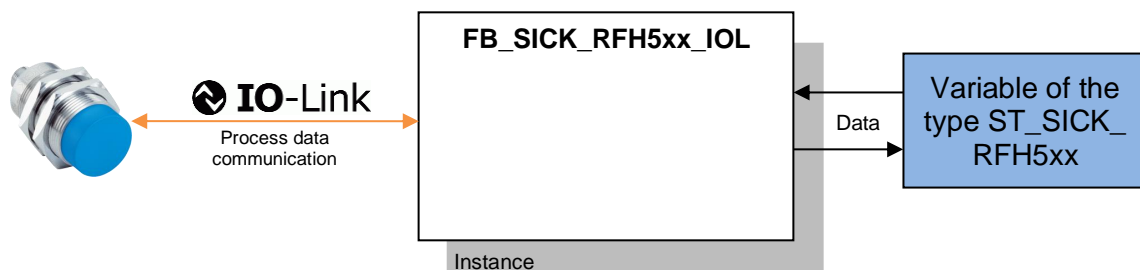


Figure 1: Concept behind the RFH5xx IO-Link function block

3 Hardware Configuration

3.1 Supported PLCs

The function block has only been tested with a Mitsubishi R04 CPU in conjunction with a RJ71EIP91 Ethernet/IP I/F unit and GX-Works 3.

**Please note!**

The function block is IO-Link Master independent and can be used for all available Masters.

3.2 RJ71EIP91 Configuration

Before the function block can be used, an IO-Link Master device must be configured in the hardware configuration. The IO-Link port used for communication with the RFH5xx must support a process data length of 32byte In/Out.

Figure 2 shows an example projecting of a SICK SIG350 IO-Link master (Ethernet/IP). The RFH5xx IO-Link device is connected to the first master port.

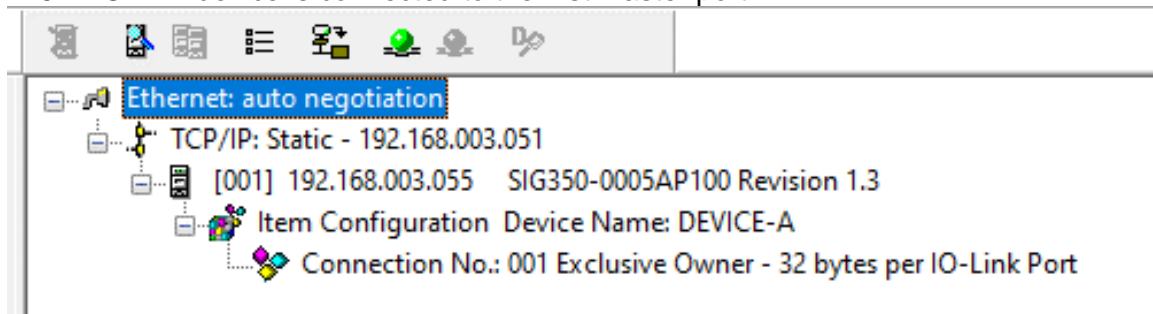


Figure 2: Example hardware configuration

4 Function block

This function block (FB) simplifies the usage of a SICK RFH5xx RFID interrogator in combination with a Mitsubishi R04CPU PLC. The FB uses only the IO-Link process data communication channel for reading or writing RFID transponder data.

The function block works asynchronously, that is, processing requires several function block calls. Therefore, it is necessary that the function block is called cyclically in the user program.



Figure 3: FB_SICK_RFH5xx_IOL function block

4.1 Block specifications

Block name:	FB_SICK_RFH5xx_IOL
Version:	1.0
Used PLC data types:	ST_SICK_RFH5xx ST_SICK_RFH5xx_Error ST_SICK_RFH5xx_AntennaField ST_SICK_RFH5xx_ReadUMem ST_SICK_RFH5xx_WriteUMem ST_SICK_RFH5xx_QueueElement ST_SICK_RFH5xx_ProcessData
Function block call up:	Cyclically
Optimized block access:	Yes
Used flags:	No
Language:	Structured Language (SCL)
Developed with:	GX-Works 3

4.2 Process data handover to the FB

The function block can read or write the user memory of a RFID transponder using the IO-Link process data channel. The mapping of the IO-Link data to PLC variables/registers depends on the IO-Link master used. To support all possible IO-Link Masters, it is necessary to exchange the process data individually.

GX-Works 3 provides the module function blocks “M+RJ71EIP91_Class1GetInputData_00A” and “M+RJ71EIP91_Class1SetOutputData_00A” to exchange process data consistently with a fieldbus device (IO-Link Master). In the next step, the I/O data must be assigned to the function blocks for further processing. The FB expects an array of Words with at least 16 elements for the I/O data. The first 16 words must contain the I/O data of the RFH.

Figure 4 shows an example to use the “M+RJ71EIP91_Class1GetInputData_00A” and the “M+RJ71EIP91_Class1SetOutputData_00A” together with the SICK RFH5xx function block.

```

(*===== Start Ethernet/IP Communication =====*)
EIP91_1.bSet_CommunicationStartupRequest := EIP91_1.bSts_ModuleReady AND EIP91_1.bSts_CommunicationReady;

(*===== Read Process Data IN =====*)
M_RJ71EIP91_Class1GetInputData_00A_1(i_bEN:= NOT bErr AND NOT bOk ,
                                     i_stModule:= EIP91_1 ,
                                     i_uConnectionNo:= 1 ,
                                     o_bOK=> bOk ,
                                     o_bErr_r => bErr ,
                                     o_uErrId=> nErrId ,
                                     o_uStatusId=> nStatusId ,
                                     o_uInputData=> arrData[0]);

(*===== Map Process Data IN =====*)
BMOV(TRUE,arrData[3], 16, TempData.ProcessDataMapping.PDInput[0]);

(*===== RFH =====*)
fbRFH5xx(TOut:=T#5s,
         BlockSize:=4,
         Data:=TempData);

(*===== Map Process Data OUT =====*)
BMOV(TRUE, TempData.ProcessDataMapping.PDOutput[0],16,arrOutputData[3]);

(*===== Write Process Data OUT =====*)
M_RJ71EIP91_Class1SetOutputData_00A_1(i_bEN:= NOT bErr_Out AND NOT bOk_Out ,
                                       i_stModule:= EIP91_1 ,
                                       i_uConnectionNo:= 1 ,
                                       i_uOutputData:= arrOutputData[0],
                                       o_bOK=> bOk_Out ,
                                       o_bErr_r => bErr_Out ,
                                       o_uErrId=> nErrId_Out ,
                                       o_uStatusId=> nStatusId_Out );

```

Figure 4: Process data exchange example

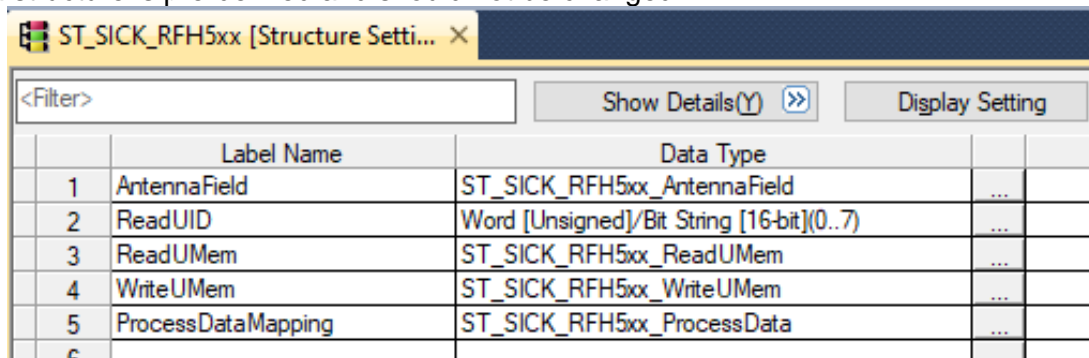
4.3 Operation of the function block

Each block action ("ReadUMem", "WriteUMem" etc.) can be parameterized via the data type "ST_SICK_RFH5xx" (Data). In order to execute a function block action, the desired action has to be selected first. It is also possible to select more than one action. In order to execute the selected action, the parameter "Req" has to be triggered with a positive edge (signal change from a logical zero to one). As long as no valid device answer has to be received, this is signaled via the parameter "Busy".

If the function block signals "Done = TRUE" at the output parameter, the action has been done successfully. If, for this action (e.g. "ReadUMem") data has been requested from the device, it will be copied into the respective data area ("Data").

4.4 Data type description

The data type "ST_SICK_RFH5xx" contains input and output parameters for all supported function block actions. The function block used an instance (variable) of this data type. The data structure is pre-defined and should not be changed.



	Label Name	Data Type
1	AntennaField	ST_SICK_RFH5xx_AntennaField
2	ReadUID	Word [Unsigned]/Bit String [16-bit](0..7)
3	ReadUMem	ST_SICK_RFH5xx_ReadUMem
4	WriteUMem	ST_SICK_RFH5xx_WriteUMem
5	ProcessDataMapping	ST_SICK_RFH5xx_ProcessData

Figure 5: ST_SICK_RFH5xx data type

4.4.1 AntennaField

This function can be used to switch the RF-Field of the Antenna on or off.

Parameter	Declaration	Data type	Description
Power	Input	Bit	Antenna power on/off True = On False = Off

Table 1: Parameter of the AntennaField function

4.4.2 ReadUID

When the ReadUID function is executed, the following structure was filled with the transponder identifier (UID).

Parameter	Declaration	Data type	Description
UID	Output	Word [Unsigned] /Bit String [16-bit](0..7)	Transponder UID

Table 2: Parameter of the ReadUID function

4.4.3 ReadUMem

Here you can define which area of the RFID tag should be read out.

Parameter	Declaration	Data type	Description
StartAddress	Input	Word [Unsigned]/Bit String [16-bit]	Block number at which the reading should be started. <u>Valid range:</u> [0..255]
NumOfBlocks	Input	Word [Unsigned]/Bit String [16-bit]	Number of blocks that should be read. The valid range depends on the predefined block size (FB input parameter). <u>Valid range:</u> (BlockSize x NumOfBlocks) <= 512
DataLength	Output	Word [Unsigned]/Bit String [16-bit]	Byte length of the data that was read out
Data	Output	Word [Unsigned]/Bit String [16-bit](0..511)	Data that was read out.

Table 3: Parameter of the ReadUMem function

4.4.4 WriteUMem

Here you can define which area of the RFID tag should be written.

Parameter	Declaration	Data type	Description
StartAddress	Input	Word [Unsigned]/Bit String [16-bit]	Block number at which the writing should be started. <u>Valid range:</u> [0..255]
NumOfBlocks	Input	Word [Unsigned]/Bit String [16-bit]	Number of blocks that should be written. The valid range depends on the predefined block size (FB input parameter). <u>Valid range:</u> (BlockSize x NumOfBlocks) <= 512

RFH5xx IO-Link FB (GX-Works 3)

Technical Information



Parameter	Declaration	Data type	Description
Data	Input	Word [Un- signed]/Bit String [16- bit](0..511)	Data that should be written.

Table 4: Parameter of the WriteUMem function

4.5 Adjust maximum data length

By default, this function block can read or write a maximum number of 512 bytes of user memory. For applications that require more data, the function block can be adapted accordingly.

Adjustment at the function block interface:

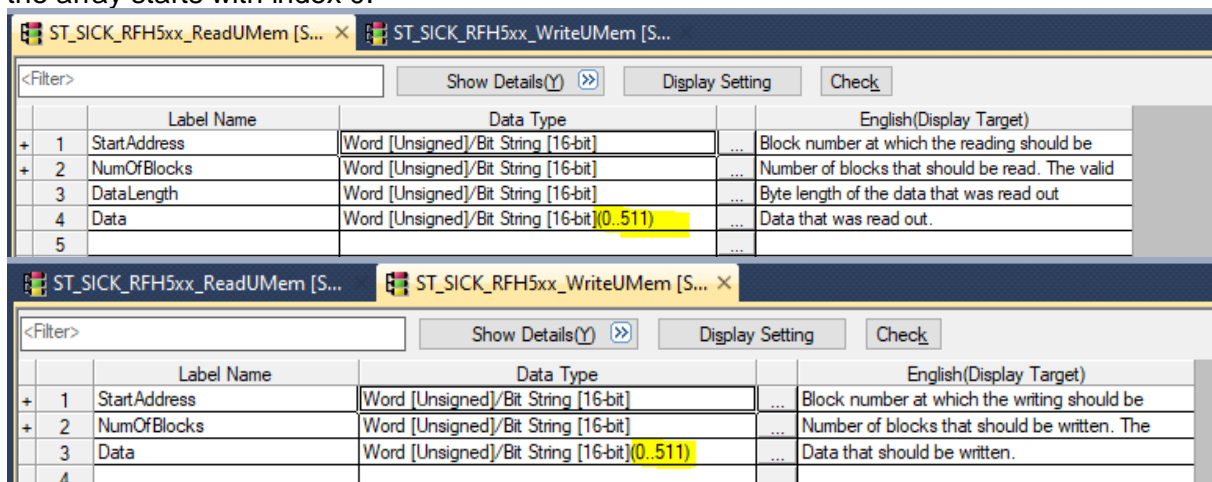
The following constants define how many data (in bytes) can be read or written at maximum. Adjust the "default values" accordingly. Make sure that the value is divisible by 4.

32	MAX_SIZE_READ	Word [Signed]	VAR_CONSTANT	512
33	MAX_SIZE_WRITE	Word [Unsigned]/Bit String [16-bit]	VAR_CONSTANT	512

Figure 6: Adjustment at the function block interface

Adaptation in the data structure:

For the data allocation the UMem "Data" arrays must be adapted accordingly. Make sure that the array starts with index 0.



	Label Name	Data Type	English(Display Target)
1	StartAddress	Word [Unsigned]/Bit String [16-bit]	Block number at which the reading should be
2	NumOfBlocks	Word [Unsigned]/Bit String [16-bit]	Number of blocks that should be read. The valid
3	DataLength	Word [Unsigned]/Bit String [16-bit]	Byte length of the data that was read out
4	Data	Word [Unsigned]/Bit String [16-bit](0..511)	Data that was read out.

	Label Name	Data Type	English(Display Target)
1	StartAddress	Word [Unsigned]/Bit String [16-bit]	Block number at which the writing should be
2	NumOfBlocks	Word [Unsigned]/Bit String [16-bit]	Number of blocks that should be written. The
3	Data	Word [Unsigned]/Bit String [16-bit](0..511)	Data that should be written.

Figure 7: Adaptation in the data structure

4.6 Behavior when error occurs

If there is a wrong input value of the function block, an error bit ("Error") is set and an error code ("Errorcode") will be given out. In this case, there is no further processing. The diagnosis parameter ("Error" and "Errorcode") of the routine maintains their value until a new request has been started.

5 Parameter

Parameter	Declaration	Data type	Description
EN	Input	Bit	Enable input (only in the FBD view)
TOut	Input	Time	Time after a timeout error occurs.
BlockSize	Input	Word [Un-signed]/Bit String [16-bit]	Block size of the transponder you want to use in the application. <u>Valid range:</u> [4,8]
Req	Input	Bit	A rising edge executes the selected actions.
ReadUID	Input	Bit	Action: Reading the UID of the transponder in the RF-Field.
WriteUMem	Input	Bit	Action: Writing data blocks into the user memory of the transponder. Please use the corresponding data structure to define which blocks of the user data should be written.
ReadUMem	Input	Bit	Action: Reading data blocks from the user memory of the transponder. Please use the corresponding data structure to define which blocks of the user data should be read.
AntennaField	Input	Bit	Action: Switching the RF-Field of the Antenna on or off. Please use the corresponding data structure to define whether the RF field should be switched on or off.
Data	In/Out	ST_SICK_RFH5xx	Contains input and output parameters for all supported function block actions.
TagPresent	Output	Bit	Indicates if there is a tag in the RF field of the RFH5xx. This flag is updated cyclically.
AntennaState	Output	Bit	Indicates the state of the antenna power. This flag is updated cyclically.
RSSI	Output	Word [Un-signed]/Bit String [16-bit]	RSSI signal level coming from the transponder. This value is updated cyclically
Alarm	Output	Bit(0..1)	The device offers the possibility to configure and output two alarms. The alarms are updated cyclically.
Done	Output	Bit	Indicates that the selected function block action has been performed without errors.
Busy	Output	Bit	Request in process FALSE: Request is terminated TRUE: Request is being processed

RFH5xx IO-Link FB (GX-Works 3)

Technical Information

Parameter	Declaration	Data type	Description
Error	Output	Bit	Error occurred. FALSE: No error TRUE: Error detected
Errorcode	Output	Word [Un-signed]/Bit String [16-bit]	Error information (see error code description)
ENO	Output	Bit	Enable output (only in the FBD view)

Table 5: Function block parameters

6 Error description

The parameter "Errorcode" contains the following error information:

- Block specific error code
- Extended error code
- Device error code

Block Errorcode	Description																										
16#0000	No error																										
16#0001	Timeout error occurs. The processing of the actions takes longer than the time set at the "TOut" parameter.																										
16#0002	No FB action selected. A request (Req) was executed without selecting an action.																										
16#0003	The defined block size is not supported by the function block. Valid values: [4, 8]																										
16#0004	It's not possible to read or write more than 512 byte of the user memory. Please adjust the number of blocks to be read or written.																										
16#0005	No transponder present in the RF-Field of RFH5xx.																										
16#0007	The start address of the requested command does not match with the response.																										
16#0008 – 16#000F	Reserved																										
16#0010	Device error detected <table><tr><th>Error Code</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td>CommandNotSupported</td><td rowspan="8">Error code values replied by the transponder to the RWM interrogation. Depend of ISO15693 command set supported by the different transponder IC of the market. These are error code values defined by the IOS15693 standard.</td></tr><tr><td>2</td><td>FormatError</td></tr><tr><td>3</td><td>OptionNotSupported</td></tr><tr><td>5</td><td>CommandProblem</td></tr><tr><td>6</td><td>CommTagError</td></tr><tr><td>15</td><td>TagError</td></tr><tr><td>16</td><td>NoMemoryBlock</td></tr><tr><td>18</td><td>BlockProtected</td></tr><tr><td>30</td><td>TAGCommError</td><td>Indicates a transponder communication error (e.g. more than 1 transponder detected or transponder reply not understood)</td></tr><tr><td>255</td><td>AppGeneralError</td><td>General Error</td></tr></table>	Error Code	Name	Description	1	CommandNotSupported	Error code values replied by the transponder to the RWM interrogation. Depend of ISO15693 command set supported by the different transponder IC of the market. These are error code values defined by the IOS15693 standard.	2	FormatError	3	OptionNotSupported	5	CommandProblem	6	CommTagError	15	TagError	16	NoMemoryBlock	18	BlockProtected	30	TAGCommError	Indicates a transponder communication error (e.g. more than 1 transponder detected or transponder reply not understood)	255	AppGeneralError	General Error
Error Code	Name	Description																									
1	CommandNotSupported	Error code values replied by the transponder to the RWM interrogation. Depend of ISO15693 command set supported by the different transponder IC of the market. These are error code values defined by the IOS15693 standard.																									
2	FormatError																										
3	OptionNotSupported																										
5	CommandProblem																										
6	CommTagError																										
15	TagError																										
16	NoMemoryBlock																										
18	BlockProtected																										
30	TAGCommError	Indicates a transponder communication error (e.g. more than 1 transponder detected or transponder reply not understood)																									
255	AppGeneralError	General Error																									

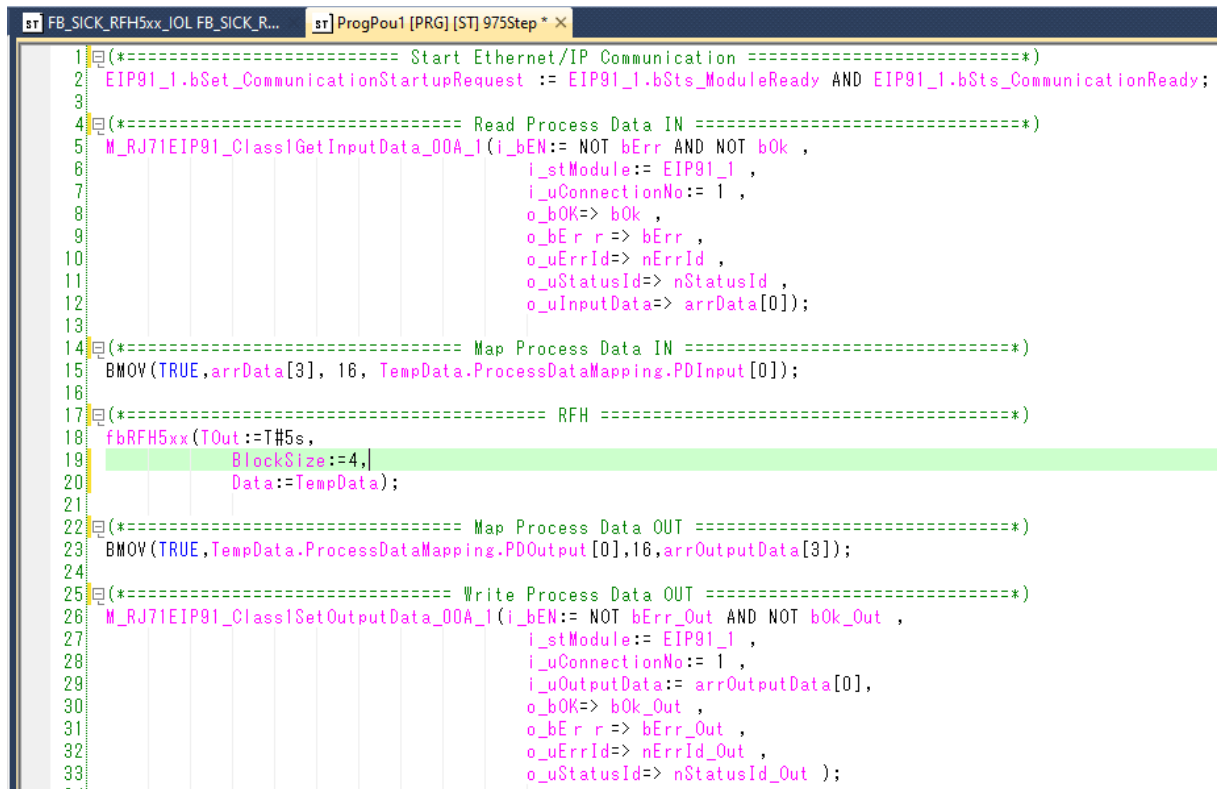
Table 6: Function block error codes

7 Example

The example routine repeats the actions ReadUID, WriteUMem and ReadUMem until the specified target iteration is reached. The routine can be started by specifying the value "ProgPou1/wTargetIteration" and setting the Bit "ProgPou1/bStartTest".

7.1.1 Process data exchange and function block call up

Reading and writing of the process data.



```

1  (*===== Start Ethernet/IP Communication =====*)
2  EIP91_1.bSet_CommunicationStartupRequest := EIP91_1.bSts_ModuleReady AND EIP91_1.bSts_CommunicationReady;
3
4  (*===== Read Process Data IN =====*)
5  M_RJ71EIP91_Class1GetInputData_00A_1(i_bEN:= NOT bErr AND NOT bOk ,
6      i_stModule:= EIP91_1 ,
7      i_uConnectionNo:= 1 ,
8      o_bOk=> bOk ,
9      o_bErr=> bErr ,
10     o_uErrId=> nErrId ,
11     o_uStatusId=> nStatusId ,
12     o_uInputData=> arrData[0]);
13
14 (*===== Map Process Data IN =====*)
15 BMOV(TRUE,arrData[3], 16, TempData.ProcessDataMapping.PDInput[0]);
16
17 (*===== RFH =====*)
18 fbRFH5xx(TOut:=T#5s,
19     BlockSize:=4,
20     Data:=TempData);
21
22 (*===== Map Process Data OUT =====*)
23 BMOV(TRUE,TempData.ProcessDataMapping.PDOutput[0],16,arrOutputData[3]);
24
25 (*===== Write Process Data OUT =====*)
26 M_RJ71EIP91_Class1SetOutputData_00A_1(i_bEN:= NOT bErr_Out AND NOT bOk_Out ,
27     i_stModule:= EIP91_1 ,
28     i_uConnectionNo:= 1 ,
29     i_uOutputData:= arrOutputData[0],
30     o_bOk=> bOk_Out ,
31     o_bErr=> bErr_Out ,
32     o_uErrId=> nErrId_Out ,
33     o_uStatusId=> nStatusId_Out );

```

Figure 8: Process data assignment

```

41 | (*===== Test routine =====*)
42 | CASE wTestStep OF
43 | // Step 0: Initialize Test
44 | 0:
45 |     IF bStartTest THEN
46 |         wTestStep := 1;
47 |         wCurrentIteration := 0;
48 |         bStartTest := FALSE;
49 |         wSuccessCounter := 0;
50 |         wErrorCounter := 0;
51 |     END_IF;
52 |
53 | // Step 1: Check if Target Iteration is reached, if yes stop. Else start another iteration.
54 | 1:
55 |     IF wCurrentIteration < wTargetIteration THEN
56 |         wTestStep := 2;
57 |         fbRFH5xx.Req := FALSE;
58 |         RETURN;
59 |     ELSE
60 |         wTestStep := 0;
61 |     END_IF;
62 |
63 | // Step 2: Read UID - Start Request
64 | 2:
65 |     fbRFH5xx.ReadUID := TRUE;
66 |     fbRFH5xx.Req := TRUE;
67 |     IF fbRFH5xx.Busy THEN
68 |         wTestStep := 3;
69 |         fbRFH5xx.ReadUID := FALSE;
70 |     END_IF;
71 |
72 | // Step 3: Read UID - Wait for completion (increment success/error counters accordingly)
73 | 3:
74 |     IF fbRFH5xx.Done THEN
75 |         wSuccessCounter := wSuccessCounter + 1;
76 |         wCurrentIteration := wCurrentIteration + 1;
77 |         fbRFH5xx.Req := FALSE;
78 |         wTestStep := 4;
79 |         RETURN;
80 |     ELSEIF fbRFH5xx.Error THEN
81 |         wErrorCounter := wErrorCounter + 1;
82 |         wCurrentIteration := wCurrentIteration + 1;
83 |         fbRFH5xx.Req := FALSE;
84 |         wTestStep := 4;
85 |         RETURN;
86 |     END_IF;
87 |
88 | // Step 4: WriteUMem - Create Random Data
89 | 4:
90 |
91 |     RND(TRUE, wRandom[0]);
92 |     wRandom[0] := wRandom[0] AND 16#00FF;
93 |
94 |     RND(TRUE, wRandom[1]);
95 |     wRandom[1] := wRandom[1] AND 16#00FF;
96 |
97 |     RND(TRUE, wRandom[2]);
98 |     wRandom[2] := wRandom[2] AND 16#00FF;
99 |
100 |     RND(TRUE, wRandom[3]);
101 |     wRandom[3] := wRandom[3] AND 16#00FF;
102 |     wTestStep := 5;
103 |
104 | // Step 5: WriteUMem - Start Request
105 | 5:
106 |     TempData.WriteUMem.Data[0] := wRandom[0];
107 |     TempData.WriteUMem.Data[1] := wRandom[1];
108 |     TempData.WriteUMem.Data[2] := wRandom[2];
109 |     TempData.WriteUMem.Data[3] := wRandom[3];
110 |
111 |     TempData.WriteUMem.StartAddress := 0;
112 |     TempData.WriteUMem.NumOfBlocks := 1;
113 |     fbRFH5xx.WriteUMem := TRUE;
114 |     fbRFH5xx.Req := TRUE;
115 |     IF fbRFH5xx.Busy THEN
116 |         wTestStep := 6;
117 |         fbRFH5xx.WriteUMem := FALSE;
118 |     END_IF;
119 |
120 | // Step 6: WriteUMem - Wait for completion (increment success/error counters accordingly)
121 | 6:
122 |     IF fbRFH5xx.Done THEN
123 |         wSuccessCounter := wSuccessCounter + 1;
124 |         wCurrentIteration := wCurrentIteration + 1;
125 |         fbRFH5xx.Req := FALSE;
126 |         wTestStep := 7;
127 |         RETURN;
128 |     ELSEIF fbRFH5xx.Error THEN
129 |         wErrorCounter := wErrorCounter + 1;
130 |         wCurrentIteration := wCurrentIteration + 1;
131 |         fbRFH5xx.Req := FALSE;
132 |         wTestStep := 7;
133 |         RETURN;
134 |     END_IF;
135 |
136 |

```

Figure 9: Example routine Part 1


```

136
137 // Step 7: ReadUMem - Start Request
138 7:
139     TempData.ReadUMem.StartAddress := 0;
140     TempData.ReadUMem.NumOfBlocks := 1;
141     fbRFH5xx.ReadUMem := TRUE;
142     fbRFH5xx.Req := TRUE;
143     IF fbRFH5xx.Busy THEN
144         wTestStep := 8;
145         fbRFH5xx.ReadUMem := FALSE;
146
147     END_IF;
148
149 // Step 8: ReadUMem - Wait for completion (increment success/error counters accordingly)
150 // Additionally check if data matches the written data of Step 5
151 8:
152     IF fbRFH5xx.Done AND TempData.ReadUMem.DataLength = 4 THEN
153         IF TempData.ReadUMem.Data[0] = wRandom[0] AND
154             TempData.ReadUMem.Data[1] = wRandom[1] AND
155             TempData.ReadUMem.Data[2] = wRandom[2] AND
156             TempData.ReadUMem.Data[3] = wRandom[3] THEN
157             wSuccessCounter := wSuccessCounter + 1;
158             wCurrentIteration := wCurrentIteration + 1;
159             fbRFH5xx.Req := FALSE;
160             wTestStep := 1;
161             RETURN;
162         ELSE
163             wErrorCounter := wErrorCounter + 1;
164             wCurrentIteration := wCurrentIteration + 1;
165             fbRFH5xx.Req := FALSE;
166             wTestStep := 1;
167             RETURN;
168         END_IF;
169     ELSIF fbRFH5xx.Error THEN
170         wErrorCounter := wErrorCounter + 1;
171         wCurrentIteration := wCurrentIteration + 1;
172         fbRFH5xx.Req := FALSE;
173         wTestStep := 1;
174         RETURN;
175     END_IF;
176
177 END_CASE;
178

```

Figure 10: Example routine Part 2