```
 7 FUNCTION FC1240 : VOID
 8
 9 AUTHOR  : 'CV'
10 VERSION : '1.0'
11
12 VAR_TEMP
13   SET_CLK_ReturnValue : INT;
14 END_VAR
15
16 BEGIN
17
18   IF ( DB240.Commands.SetSystemDate_and_Time = TRUE ) THEN
19     SET_CLK_ReturnValue := SFC0(DB140.ConfigParameters_extend.Now // IN: DATE_AND_TIME
20                                 );
21     DB240.Commands.SetSystemDate_and_Time := FALSE;
22   END_IF;
23
24 END_FUNCTION
25
```

# Summary of Comments on FC1000 -- PIXEL_20080314\SIMATI

## Page: 64

Number: 1 Author: jbk Subject: Sticky Note Date: 4/24/2008 10:24:43 PM

seems this return value goes unused. Can the setting of the clock fail?

```
10 FUNCTION FC1400 : VOID
11
12 AUTHOR  : 'CV'
13 VERSION : '1.0'
14
15 VAR_INPUT
16   currState_isOk : BOOL;
17   Alarm_Delay : TIME;
18   Acknowledge_Command : BOOL;
19 END_VAR
20
21 VAR_IN_OUT
22   lastTimeStateOk : TIME_OF_DAY;
23   Alarm_isActive : BOOL;
24   Alarm_beenActive : BOOL;
25 END_VAR
26
27 VAR_TEMP
28   timeNow : TIME_OF_DAY;
29   durationStateNotOk : TIME;
30 END_VAR
31
32 BEGIN
33
34   timeNow := DB10.todNow;
35
36 //--- set lastTimeStateOk to current PLC system Time,
37 //    provided Module/Channel is currently not in Alarm condition
38   IF ( currState_isOk = TRUE ) THEN
39     lastTimeStateOk := timeNow;
40   END_IF;
41
42 //--- compute Duration for which Module/Channel has been in Alarm condition
43   omputeDurStateNotOk(timeNow := timeNow                        // IN: TIME_OF_DAY
44                      ,lastTimeStateOk := lastTimeStateOk        // IN: TIME_OF_DAY
45                      ,durationStateNotOk := durationStateNotOk  // IN: TIME
46                      ,Time_Error := DB13.TimeError               // IN: BOOL
47                       );
48
49 //--- raise Alarm if Module/Channel has been in Alarm condition for Duration > Alarm Delay,
50 //    enter Alarm in history
51   IF ( durationStateNotOk > Alarm_Delay ) THEN
52     Alarm_isActive := TRUE;
53     Alarm_beenActive := TRUE;
54   ELSE
55     Alarm_isActive := FALSE;
56   END_IF;
57
58 //--- clear Alarm history if requested by Alarm "ACKNOWLEDGE" command issued by Operator
59   IF ( (Acknowledge_Command = TRUE) AND (Alarm_isActive = FALSE) ) THEN
60     Alarm_beenActive := FALSE;
61   END_IF;
62
63 END_FUNCTION
64
```

1

Called by FC402 2

3

4 5

6

beenActive - turned on when alarm condition present and only turned off when operator acknowledges the alarm AND it is still in alarm condition? 7

# Page: 74

Number: 1 Author: jbk Subject: Sticky Note Date: 4/24/2008 3:15:31 PM

why is time_of_day the right thing to use?
why not date_and_time?

milliseconds into the current day seems to be more trouble than it is worth.

Number: 2 Author: jbk Subject: Text Box Date: 4/23/2008 4:30:00 PM

Called by FC402

Number: 3 Author: jbk Subject: Sticky Note Date: 4/24/2008 3:29:18 PM

the combination of in-out arguments and special cases in this function and the one it calls makes me nervous. I think it is more difficult to understand than the logic should be.

Is there a better way to express this whole thing? How about:

```
IN: time of last state change
IN: state
IN: delay
OUT: in alarm condition

if state = bad then
 if (now - last_state_change) > delay then
   return bad
 else return good
 endif
endif

return good
```

Number: 4 Author: jbk Subject: Highlight Date: 4/23/2008 4:15:44 PM

Number: 5 Author: jbk Subject: Sticky Note Date: 4/23/2008 4:32:48 PM

The last two parameters are OUTs

Number: 6 Author: jbk Subject: Sticky Note Date: 4/23/2008 4:38:44 PM

```
isActive = durationStateNotOk > Alarm_Delay
if isActive { beenActive = true }
```

Number: 7 Author: jbk Subject: Text Box Date: 4/24/2008 3:33:01 PM

beenActive - turned on when alarm condition present and only turned off when operator acknowledges the alarm AND it is still in alarm condition?

```
 6 FUNCTION FC1401 : VOID
 7
 8 AUTHOR  : 'CV'
 9 VERSION : '1.0'
10
11 VAR_INPUT
12   timeNow : TIME_OF_DAY;
13   lastTimeStateOk : TIME_OF_DAY;
14 END_VAR
15
16 VAR_OUTPUT
17   durationStateNotOk : TIME;
18   Time_Error : BOOL;
19 END_VAR
20
21 BEGIN
22
23 //--- compute Duration in Time during which the Module/Channel has been in an Alarm condition
24   durationStateNotOk := timeNow - lastTimeStateOk;
25   Time_Error := FALSE;
26
27 //--- take care of negative Duration
28 //    arising when e.g. lastTimeStateOk@yesterday, 11pm and
29 //                                timeNow@today, 1am
30   IF ( durationStateNotOk < TIME#0d0h0m ) THEN
31     durationStateNotOk := durationStateNotOk + TIME#1d0h0m;
32   END_IF;
33
34 //--- in case timeNow - lastTimeStateOk differs by more than one day,
35 //    set an Error flag indicating that there is a problem with the PLC's system time
36   IF ( durationStateNotOk < TIME#0d0h0m ) THEN
37     Time_Error := TRUE;
38   END_IF;
39
40 END_FUNCTION
41
```

# Page: 76

Number: 1 Author: jbk Subject: Sticky Note Date: 4/24/2008 11:20:58 AM

Just a note: you can use the DATE_AND_TIME format and use the function FC34_SB_DT_DT to get a TIME difference out.

```
 7 FUNCTION FC402 : VOID
 8
 9 AUTHOR  : 'CV'
10 VERSION : '1.0'
11
12 VAR_TEMP
13   iDigitalInput : INT;
14   numDigInput_processed : INT;
15   numDigInput_inAlarm : INT;
16   numDigInput_CfgPar_set : INT;
17   numDigInput_CfgPar_Err : INT;
18   digInputAdress_Qindex : INT;
19   digInputAdress_Bindex : INT;
20   DigitalInput_isOk : BOOL;
21   moduleId : INT;
22   channelId : INT;
23   unmaskedChannelDiagnosticsBits : BYTE;
24   FILL_ReturnValue : INT;
25   CRC16_CheckSum_DB402 : WORD;
26 END_VAR
27
28 BEGIN
29
30 //--- count total number of Digital Input Channels processed,
31 //    number of Channels currently in Alarm,
32 //    number of Channels for which erroneous have been uploaded into DB 602 by PVSS
33 //    and number of Channels for which valid Configuration Parameters have been uploaded and h
   ave been successfully transferred to DB 102,
34 //    containing the Settings actually used by PLC
35   numDigInput_processed := 0;
36   numDigInput_inAlarm := 0;
37   numDigInput_CfgPar_set := 0;
38   numDigInput_CfgPar_Err := 0;
39
40 //--- process all Digital Input Channels
41   FOR iDigitalInput := 1 TO DB1.NumDigitalInputs DO
42
43 //--- check if Channel has been correctly configured
44     IF ( DB402.DigitalInput[iDigitalInput].ConfigParameterStatus.ConfigParameters_set = TRUE )
    THEN
45
46 //--- read Status of Digital Input
47 //    from Process Image
48       DB402.DigitalInput[iDigitalInput].Readings.InputId := iDigitalInput;
49
50       digInputAdress_Qindex := BYTE_TO_INT(DB102.DigitalInput[iDigitalInput].ConfigParameters_
   base.Input_AddressQindex);
51       digInputAdress_Bindex := BYTE_TO_INT(DB102.DigitalInput[iDigitalInput].ConfigParameters_
   base.Input_AddressBindex);
52       DB402.DigitalInput[iDigitalInput].Readings.Input_Status := I [digInputAdress_Qindex,
53                                                                   digInputAdress_Bindex];
54

55 //--- check if current Digital Input Status corresponds to an Alarm condition
56       DigitalInput_isOk := TRUE;
57       IF ( DB402.DigitalInput[iDigitalInput].Readings.Input_Status <> DB102.DigitalInput[iDigi
   talInput].ConfigParameters_base.Input_StatusWhenOk ) THEN
58         DigitalInput_isOk := FALSE;
59       END_IF;
60
61 //--- copy Channel diagnostics information from DB59,
62 //    apply Mask and check for Alarm condition
63       moduleId := DB102.DigitalInput[iDigitalInput].ConfigParameters_base.assocModule_Id;
64       channelId := BYTE_TO_INT(DB102.DigitalInput[iDigitalInput].ConfigParameters_base.assocMo
   dule_channelId);
65       DB402.DigitalInput[iDigitalInput].Readings.ChannelDiagn_isAvailable := DB59.Module_Statu
   s[moduleId].Diagnostic_isAvailable;
66       DB402.DigitalInput[iDigitalInput].Readings.ChannelDiagn_readError := DB59.Module_Status[
   moduleId].RD_REC_readError;
67       IF ( (DB59.Module_Status[moduleId].Diagnostic_isAvailable = TRUE) AND
68            (DB59.Module_Status[moduleId].RD_REC_readError = FALSE) ) THEN
69         DB402.DigitalInput[iDigitalInput].Readings.ChannelDiagn_lastTimeUpd := DB59.Module_Sta
   tus[moduleId].Diagnostic_lastTimeUpd;
```

# Page: 121

Number: 1 Author: jbk Subject: Sticky Note Date: 4/23/2008 3:23:44 PM

is the outcome of the byte to int conversion always expected to succeed?

If you want the value, why not call a function with iDigitalInput and have it return the book value?

Why is this conversion done anyway since it is configuration information?

Number: 2 Author: jbk Subject: Sticky Note Date: 4/23/2008 3:15:04 PM

this is read of the digital input value.

Number: 3 Author: jbk Subject: Sticky Note Date: 4/23/2008 3:27:12 PM

Why is this not just DigitalInput_isOk = NOT(Input_Status XOR Input_StatusWhenOk) ?

```
70          DB402.DigitalInput[iDigitalInput].Readings.Channel_DiagnosticsBits := DB59.Module_Stat
    us[moduleId].Channel[channelId];
71
72          unmaskedChannelDiagnosticsBits := (DB402.DigitalInput[iDigitalInput].Readings.Channel_
    DiagnosticsBits AND
73                                            DB102.DigitalInput[iDigitalInput].ConfigParameters_ex
    tend.Channel_DiagnAlarmMask);
74          IF ( unmaskedChannelDiagnosticsBits <> B#16#0 ) THEN
75            DigitalInput_isOk := FALSE;
76          END_IF;
77 //--- do not care if diagnostic information is available for Channel
78 //     in case Mask bits are all zero
79        ELSIF ( DB102.DigitalInput[iDigitalInput].ConfigParameters_extend.Channel_DiagnAlarmMask
     <> 2#0000_0000 ) THEN
80          DigitalInput_isOk := FALSE;
81        END_IF;
82
83 //--- raise Alarm if Digital Input Status has been in Alarm condition for Time > Delay
84 //     or Channel diagnostics indicate Alarm condition
85        computeAlarmStatus(currState_isOk := DigitalInput_isOk
          FC1400                          // IN: BOOL
86                           ,Alarm_Delay := DB102.DigitalInput[iDigitalInput].ConfigParameters_ext
    end.Alarm_Delay // IN: TIME
87                           ,Acknowledge_Command := DB202.DigitalInput[iDigitalInput].Commands.Ala
    rm_Acknowledge   // IN: BOOL
88                           ,lastTimeStateOk := DB402.DigitalInput[iDigitalInput].Readings.Alarm_l
    astTimeInputOk   // IN_OUT: TIME_OF_DAY
89                           ,Alarm_isActive := DB402.DigitalInput[iDigitalInput].Readings.Alarm_is
    Active           // IN_OUT: BOOL
90                           ,Alarm_beenActive := DB402.DigitalInput[iDigitalInput].Readings.Alarm_
    beenActive       // IN_OUT: BOOL
91                            );
92
93        IF ( DB402.DigitalInput[iDigitalInput].Readings.Alarm_isActive ) THEN
94          numDigInput_inAlarm := numDigInput_inAlarm + 1;
95        END_IF;
96
97        numDigInput_CfgPar_set := numDigInput_CfgPar_set + 1;
98      ELSE
99 //--- in case no valid Configuration Parameters have been set,
100 //     clear all Readings in DB 402, except for Configuration Parameter Status and Error flags
101        FILL_ returnValue := SFC21(BVAL := DB4.zero                                  // IN : ANY
102                                  ,BLK  := DB402.DigitalInput[iDigitalInput].Readings // OUT : AN
    Y
103                                   );
104
105        numDigInput_CfgPar_Err := numDigInput_CfgPar_Err + 1;
106      END_IF;
107
108      numDigInput_processed := numDigInput_processed + 1;
109
110 //--- reset Alarm "ACKNOWLEDGE" command
111      DB202.DigitalInput[iDigitalInput].Commands.Alarm_Acknowledge := FALSE;
112    END_FOR;
113
114 //--- store Digital Input Channel status information in DB 402
115    DB402.NumDigInput_processed := numDigInput_processed;
116    DB402.NumDigInput_inAlarm := numDigInput_inAlarm;
117    DB402.NumDigInput_CfgPar_set := numDigInput_CfgPar_set;
118    DB402.NumDigInput_CfgPar_Err := numDigInput_CfgPar_Err;
119    DB402.anyDigInput_CfgPar_Err := (numDigInput_CfgPar_Err > 0);
120    DB402.allDigInput_CfgPar_set := (numDigInput_CfgPar_set = numDigInput_processed);
121
122 //--- compute CRC16 check-sum for "Readings" DB 402
123    CRC16_CheckSum_DB402 := computeCRC16CheckSum(Sptr := DB402);
124    DB402.Header.CRC16 := CRC16_CheckSum_DB402;
125
126 END_FUNCTION
127
```

# Page: 122

Number: 1 Author: jbk Subject: Highlight Date: 4/23/2008 3:16:07 PM

Number: 2 Author: jbk Subject: Text Box Date: 4/23/2008 3:17:45 PM

FC1400

Number: 3 Author: jbk Subject: Sticky Note Date: 4/24/2008 11:36:46 AM

counts how many bits were manipulated in the loop.

Number: 4 Author: jbk Subject: Sticky Note Date: 4/24/2008 11:38:44 AM

Is this really the right place for this clearing?
Why not do this at configuration validation time once and don't touch it afterwards?

Number: 5 Author: jbk Subject: Sticky Note Date: 4/24/2008 12:11:09 PM

Note for jbk - this works because the acknowledgement is not asynchronous i.e. flag is set in an earlier step.

So the Ack from the operator only persists for one cycle? even if the reset does not occur because the alarm persists?

Why should the reader function (here) be resetting the operator ACK in DB202? shouldn't this be done elsewhere? What are the rules for data ownership (modification versus read-only)?

Number: 6 Author: jbk Subject: Sticky Note Date: 4/24/2008 12:15:37 PM

why is this necessary and why is it done here?

again, what is the rule about what code modifies what blocks? I'm not sure I understand need for a header in the readout area for digital I/O.

```
1 // FUNCTION FC 404
2 //
3 // Read Process values of Relay Digital Output Channels;
4 // store all Readings in DB 404.
5 //
6
```

1

The logic I see:
for each channel
 check if configuration is OK
 if so then
  get the channel index
  read the data
  set alarm and interlocked state
  adjust the interlock value
  muck with beenInterlocked depending value of operator ack

  count the valid and total channels
  clear the ack from the operator
  copy all the counts from local variables to the data block

  calculate a checksum for the header

Number: 1 Author: jbk Subject: Text Box Date: 4/24/2008 2:08:30 PM

The logic I see:
for each channel
 check if configuration is OK
 if so then
  get the channel index
  read the data
  set alarm and interlocked state
  adjust the interlock value
  muck with beenInterlocked depending value of operator ack

  count the valid and total channels
  clear the ack from the operator
  copy all the counts from local variables to the data block

  calculate a checksum for the header

```
 8 FUNCTION FC410 : VOID
 9
10 AUTHOR  : 'CV'
11 VERSION : '1.0'
12
13 VAR_TEMP
14   iTempSensor : INT;
15   numTempSens_processed : INT;
16   numTempSens_inAlarm : INT;
17   numTempSens_CfgPar_set : INT;
18   numTempSens_CfgPar_Err : INT;
19   rawTemperature : INT;
20   calibratedTemperature : REAL;
21   TempSensor_isOk : BOOL;
22   moduleId : INT;
23   channelId : INT;
24   unmaskedChannelDiagnosticsBits : BYTE;
25   FILL_ReturnValue : INT;
26   CRC16_CheckSum_DB410 : WORD;
27 END_VAR
28
29 BEGIN
30
31 //--- count total number of RTD Temperature Sensors processed,
32 //    number of Sensors currently in Alarm,
33 //    number of Sensors for which erroneous have been uploaded into DB 610 by PVSS
34 //    and number of Sensors for which valid Configuration Parameters have been uploaded and ha
   ve been successfully transferred to DB 110,
35 //    containing the Settings actually used by PLC
36   numTempSens_processed := 0;
37   numTempSens_inAlarm := 0;
38   numTempSens_CfgPar_set := 0;
39   numTempSens_CfgPar_Err := 0;
40
41 //--- process all RTD Temperature Sensors
42   FOR iTempSensor := 1 TO DB1.NumTemperatureSensors DO
43
44 //--- check if Channel has been correctly configured
45     IF ( DB410.TempSensor[iTempSensor].ConfigParameterStatus.ConfigParameters_set = TRUE ) THE
   N
46
47 //--- read "raw" Temperature (in S7-internal units)
48 //    from Process Image
49       DB410.TempSensor[iTempSensor].Readings.SensorId := iTempSensor;
50       DB410.TempSensor[iTempSensor].Readings.Sensor_rawTemperature := PIW[DB110.TempSensor[iTe
   mpSensor].ConfigParameters_base.Sensor_Address];
51
52 //--- check if "raw" Temperature is within nominal Measurement Range
53 //    (0...27648 for RTD Temperature Sensors)
54       rawTemperature := WORD_TO_INT(DB410.TempSensor[iTempSensor].Readings.Sensor_rawTemperatu
   re);
55       IF ( 1rawTemperature < -12000) OR (rawTemperature > +13000) ) THEN          2
56         DB410.TempSensor[iTempSensor].Readings.Sensor_outsideMeasRange := TRUE;
57       ELSE
58         DB410.TempSensor[iTempSensor].Readings.Sensor_outsideMeasRange := FALSE;
59       END_IF;
60                                  3
61       4alibratedTemperature := DB110.TempSensor[iTempSensor].ConfigParameters_extend.Sensor_ca
   librationSlope * rawTemperature
62                                + DB110.TempSensor[iTempSensor].ConfigParameters_extend.Sensor_c
   alibrationOffset;
63       DB410.TempSensor[iTempSensor].Readings.Sensor_calibrTemperature := calibratedTemperature
   ;
64


65 //--- check if calibrated Temperature is currently within Alarm range
66       TempSensor_isOk := TRUE;
67       IF ( ((calibratedTemperature < DB110.TempSensor[iTempSensor].ConfigParameters_extend.Ala
   rm_limitLow) AND
68             DB110.TempSensor[iTempSensor].ConfigParameters_extend.Alarm_enabledLow = TRUE) OR
69            ((calibratedTemperature > DB110.TempSensor[iTempSensor].ConfigParameters_extend.Ala
   rm_limitHigh) AND
```

# Page: 127

Number: 1 Author: jbk Subject: Highlight Date: 4/24/2008 2:19:15 PM

Number: 2 Author: jbk Subject: Sticky Note Date: 4/24/2008 3:08:26 PM

why are these the the nominal range?

can they be anything else?
what condition does this trigger?

why not just assign the value of the expression?

Number: 3 Author: jbk Subject: Sticky Note Date: 4/24/2008 2:23:50 PM

why are temporary variables used for calculation and then copied into blocks?
Is this required?

Number: 4 Author: jbk Subject: Highlight Date: 4/24/2008 2:24:01 PM

```
70              DB110.TempSensor[iTempSensor].ConfigParameters_extend.Alarm_enabledHigh = TRUE) )
    THEN
71          2empSensor_isOk := FALSE;   1
72        END_IF;
73
74  //--- copy Channel diagnostics information from DB59,
75  //     apply Mask and check for Alarm condition
76        moduleId := DB110.TempSensor[iTempSensor].ConfigParameters_base.assocModule_Id;
77        channelId := BYTE_TO_INT(DB110.TempSensor[iTempSensor].ConfigParameters_base.assocModule
    _channelId);
78        DB410.TempSensor[iTempSensor].Readings.ChannelDiagn_isAvailable := DB59.Module_Status[mo
    duleId].Diagnostic_isAvailable;
79        DB410.TempSensor[iTempSensor].Readings.ChannelDiagn_readError := DB59.Module_Status[modu
    leId].RD_REC_readError;
80        IF ( (DB59.Module_Status[moduleId].Diagnostic_isAvailable = TRUE) AND
81             (DB59.Module_Status[moduleId].RD_REC_readError = FALSE) ) THEN
82          DB410.TempSensor[iTempSensor].Readings.ChannelDiagn_lastTimeUpd := DB59.Module_Status[
    moduleId].Diagnostic_lastTimeUpd;
83          DB410.TempSensor[iTempSensor].Readings.Channel_DiagnosticsBits := DB59.Module_Status[m
    oduleId].Channel[channelId];
84
85          unmaskedChannelDiagnosticsBits := (DB410.TempSensor[iTempSensor].Readings.Channel_Diag
    nosticsBits AND
86                                             DB110.TempSensor[iTempSensor].ConfigParameters_exte
    nd.Channel_DiagnAlarmMask);
87          IF ( unmaskedChannelDiagnosticsBits <> B#16#0 ) THEN
88            TempSensor_isOk := FALSE;
89          END_IF;
90  //--- do not care if diagnostic information is available for Channel
91  //     in case Mask bits are all zero
92        ELSIF ( DB110.TempSensor[iTempSensor].ConfigParameters_extend.Channel_DiagnAlarmMask <>
    2#0000_0000 ) THEN
93          TempSensor_isOk := FALSE;
94        END_IF;
95
96                                                                                  3
97  //--- raise Alarm if calibrated Temperature has been within Alarm range for Time > Delay
98  //     or Channel diagnostics indicate Alarm condition
99        computeAlarmStatus(currState_isOk := TempSensor_isOk
                 // IN: BOOL
100                          ,Alarm_Delay := DB110.TempSensor[iTempSensor].ConfigParameters_extend.
    Alarm_Delay // IN: TIME
101                          ,Acknowledge_Command := DB210.TempSensor[iTempSensor].Commands.Alarm_A
    cknowledge  // IN: BOOL
102                          ,lastTimeStateOk := DB410.TempSensor[iTempSensor].Readings.Alarm_lastT
    imeSensorOk // IN_OUT: TIME_OF_DAY
103                          ,Alarm_isActive := DB410.TempSensor[iTempSensor].Readings.Alarm_isActi
    ve          // IN_OUT: BOOL
104                          ,Alarm_beenActive := DB410.TempSensor[iTempSensor].Readings.Alarm_been
    Active      // IN_OUT: BOOL
105                           );
106
107       IF ( DB410.TempSensor[iTempSensor].Readings.Alarm_isActive ) THEN
108         numTempSens_inAlarm := numTempSens_inAlarm + 1;
109       END_IF;
110
111       numTempSens_CfgPar_set := numTempSens_CfgPar_set + 1;
112     ELSE
113 //--- in case no valid Configuration Parameters have been set,
114 //     clear all Readings in DB 410, except for Configuration Parameter Status and Error flags
115       FILL_ReturnValue := SFC21(BVAL := DB4.zero                                  // IN : ANY
116                                ,BLK  := DB410.TempSensor[iTempSensor].Readings // OUT : ANY
117                                 );
118
119       numTempSens_CfgPar_Err := numTempSens_CfgPar_Err + 1;
120     END_IF;
121
122     numTempSens_processed := numTempSens_processed + 1;
123
124 //--- reset Alarm "ACKNOWLEDGE" command
125     DB210.TempSensor[iTempSensor].Commands.Alarm_Acknowledge := FALSE;
126   END_FOR;
127
```

# Page: 128

Number: 1 Author: jbk Subject: Sticky Note Date: 4/24/2008 2:33:46 PM

I think the logic for this variable can be simplified and made easier to read. A channel is not OK unless it is in range. What is here is a declaration of an OK channel and several checks to see if it is not.

Number: 2 Author: jbk Subject: Highlight Date: 4/24/2008 2:30:08 PM

Number: 3 Author: jbk Subject: Sticky Note Date: 4/24/2008 3:10:10 PM

what about reading outside range?

```
 8 FUNCTION FC411 : VOID
 9
10 AUTHOR  : 'CV'
11 VERSION : '1.0'
12
13 VAR_TEMP
14   iHumidSensor : INT;
15   numHumidSens_processed : INT;
16   numHumidSens_inAlarm : INT;
17   numHumidSens_CfgPar_set : INT;
18   numHumidSens_CfgPar_Err : INT;
19   rawHumidity : INT;
20   calibratedHumidity : REAL;
21   refTemperature : REAL;
22   refTempSensor_isOk : BOOL;
23   refTemperatureSensor_Id : INT;
24   relHumidity_isValid : BOOL;
25   computedDewPoint : REAL;
26   DewPoint_isValid : BOOL;
27   HumidSensor_isOk : BOOL;
28   Channel_isOk : BOOL;
29   moduleId : INT;
30   channelId : INT;
31   unmaskedChannelDiagnosticsBits : BYTE;
32   FILL_ReturnValue : INT;
33   CRC16_CheckSum_DB411 : WORD;
34 END_VAR
35
36 BEGIN
37
38 //--- count total number of HMX Humidity Sensors processed,
39 //    number of Sensors currently in Alarm,
40 //    number of Sensors for which erroneous have been uploaded into DB 611 by PVSS
41 //    and number of Sensors for which valid Configuration Parameters have been uploaded and ha
   ve been successfully transferred to DB 111,
42 //    containing the Settings actually used by PLC
43   numHumidSens_processed := 0;
44   numHumidSens_inAlarm := 0;
45   numHumidSens_CfgPar_set := 0;
46   numHumidSens_CfgPar_Err := 0;
47
48 //--- process all HMX Humidity Sensors
49   FOR iHumidSensor := 1 TO DB1.NumHumiditySensors DO
50
51 //--- check if Channel has been correctly configured
52     IF ( DB411.HumidSensor[iHumidSensor].ConfigParameterStatus.ConfigParameters_set = TRUE ) T
   HEN
53
54 //--- read "raw" Humidity (in S7-internal units)
55 //    from Process Image
56       DB411.HumidSensor[iHumidSensor].Readings.SensorId := iHumidSensor;
57       DB411.HumidSensor[iHumidSensor].Readings.Sensor_rawHumidity := PIW[DB111.HumidSensor[iHu
   midSensor].ConfigParameters_base.Sensor_Address];
58
59 //--- check if "raw" Humidity is within nominal Measurement Range
60 //    (-27648...+27648 for Analog Input Channels measuring Voltages)
61       rawHumidity := WORD_TO_INT(DB411.HumidSensor[iHumidSensor].Readings.Sensor_rawHumidity);

62       IF ( (rawHumidity < -27648) OR (rawHumidity > +27648) ) THEN
63         DB411.HumidSensor[iHumidSensor].Readings.Sensor_outsideMeasRange := TRUE;
64       ELSE
65         DB411.HumidSensor[iHumidSensor].Readings.Sensor_outsideMeasRange := FALSE;
66       END_IF;
67
68 //--- compute relative Humidity and Dew-Point;
69 //    in order to compute the relative Humidity and Dew-Point, a RTD Temperature Sensor is req
   uired to define a reference Temperature;
70 //    require that the RTD Temperature Sensor defined as reference has been correctly configur
   ed
71 //    and its current Process value is within the Measurement range of RTD Modules
72       refTemperatureSensor_Id := DB111.HumidSensor[iHumidSensor].ConfigParameters_base.refTemp
   eratureSensor_Id;
73       IF ( (DB410.TempSensor[refTemperatureSensor_Id].ConfigParameterStatus.ConfigParameters_s
```

# Page: 131

Number: 1 Author: jbk Subject: Highlight Date: 4/24/2008 4:25:19 PM

Number: 2 Author: jbk Subject: Sticky Note Date: 4/24/2008 4:25:44 PM

See temperature reading code

```
et = TRUE) AND
            (DB410.TempSensor[refTemperatureSensor_Id].Readings.Sensor_outsideMeasRange = FALSE
) ) THEN
        refTemperature := DB410.TempSensor[refTemperatureSensor_Id].Readings.Sensor_calibrTemp
erature;
        refTempSensor_isOk := TRUE;

        computeRelHumidity(rawHumidity := rawHumidity
                                  // IN: WORD
                          ,calibratedAirTemperature := refTemperature
                                  // IN: REAL
                          ,Sensor_calibSlopeA := DB111.HumidSensor[iHumidSensor].ConfigParamet
ers_extend.Sensor_calibSlopeA   // IN: REAL
                          ,Sensor_calibSlopeB := DB111.HumidSensor[iHumidSensor].ConfigParamet
ers_extend.Sensor_calibSlopeB   // IN: REAL
                          ,Sensor_calibOffsetA := DB111.HumidSensor[iHumidSensor].ConfigParame
ters_extend.Sensor_calibOffsetA // IN: REAL
                          ,Sensor_calibOffsetB := DB111.HumidSensor[iHumidSensor].ConfigParame
ters_extend.Sensor_calibOffsetB // IN: REAL
                          ,condBoard_Slope := DB111.HumidSensor[iHumidSensor].ConfigParameters
_extend.condBoard_calibSlope    // IN: REAL
                          ,condBoard_Offset := DB111.HumidSensor[iHumidSensor].ConfigParameter
s_extend.condBoard_calibOffset  // IN: REAL
                          ,condBoard_Vbias := DB111.HumidSensor[iHumidSensor].ConfigParameters
_extend.condBoard_calibVref     // IN: REAL
                          ,calibratedRelativeHumidity := calibratedHumidity
                                  // OUT: REAL
                          ,relHumidity_isValid := relHumidity_isValid
                                  // OUT: BOOL
                           );

        DB411.HumidSensor[iHumidSensor].Readings.Sensor_calibrHumidity := calibratedHumidity;

        computeDewPoint(calibratedRelativeHumidity := calibratedHumidity // IN: REAL
                       ,calibratedAirTemperature := refTemperature       // IN: REAL
                       ,computedDewPoint := computedDewPoint            // OUT: REAL
                       ,DewPoint_isValid := DewPoint_isValid            // OUT: BOOL
                        );

//--- check if computed Dew-Point Value is flagged as being valid;
//    otherwise, set Dew-Point Value to absolute zero Temperature
//    (which will trigger Dew-Point based Interlock)
        IF ( DewPoint_isValid ) THEN
          DB411.HumidSensor[iHumidSensor].Readings.computedDewPoint := computedDewPoint;
        ELSE
          DB411.HumidSensor[iHumidSensor].Readings.computedDewPoint := -273.15;
        END_IF;
        DB411.HumidSensor[iHumidSensor].Readings.referenceTemperature := refTemperature;

      ELSE
        refTempSensor_isOk := FALSE;
      END_IF;



//--- check if calibrated Humidity is currently within Alarm range
      HumidSensor_isOk := TRUE;
      IF ( ((calibratedHumidity < DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Alar
m_limitLowHumidity) AND
            DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Alarm_enabledLowHumid = TR
UE) OR
            ((calibratedHumidity > DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Alar
m_limitHighHumidity) AND
            DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Alarm_enabledHighHumid = T
RUE) ) THEN
        HumidSensor_isOk := FALSE;
      END_IF;

//--- check if computed Dew-Point is currently within Alarm range
      IF ( (refTempSensor_isOk = TRUE) AND
           (DewPoint_isValid = TRUE) AND
           (computedDewPoint > (refTemperature - DB111.HumidSensor[iHumidSensor].ConfigParamet
```

# Page: 132

Number: 1 Author: jbk Subject: Sticky Note Date: 4/24/2008 4:34:15 PM

I'm not sure I like the check here for a valid temperature reading. Two things need to be checked to get relative humidity.

Number: 2 Author: jbk Subject: Sticky Note Date: 4/24/2008 4:35:45 PM

many of the functions use this pattern of setting an OUT parameter describing if the value is valid. There seems to be an easier way using the EN/ENO flag in the function and checking for OK in an if statement after the execution.

Number: 3 Author: jbk Subject: Highlight Date: 4/24/2008 4:36:36 PM

```
ers_extend.Alarm_marginDewPoint)) AND
            (DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Alarm_enabledDewPoint = TRUE) ) THEN
          HumidSensor_isOk := FALSE;
        END_IF;

//--- copy Channel diagnostics information from DB59,
//    apply Mask and check for Alarm condition
        moduleId := DB111.HumidSensor[iHumidSensor].ConfigParameters_base.assocModule_Id;
        channelId := BYTE_TO_INT(DB111.HumidSensor[iHumidSensor].ConfigParameters_base.assocModule_channelId);
        DB411.HumidSensor[iHumidSensor].Readings.ChannelDiagn_isAvailable := DB59.Module_Status[moduleId].Diagnostic_isAvailable;
        DB411.HumidSensor[iHumidSensor].Readings.ChannelDiagn_readError := DB59.Module_Status[moduleId].RD_REC_readError;
        IF ( (DB59.Module_Status[moduleId].Diagnostic_isAvailable = TRUE) AND
             (DB59.Module_Status[moduleId].RD_REC_readError = FALSE) ) THEN
          DB411.HumidSensor[iHumidSensor].Readings.ChannelDiagn_lastTimeUpd := DB59.Module_Status[moduleId].Diagnostic_lastTimeUpd;
          DB411.HumidSensor[iHumidSensor].Readings.Channel_DiagnosticsBits := DB59.Module_Status[moduleId].Channel[channelId];

          unmaskedChannelDiagnosticsBits := (DB411.HumidSensor[iHumidSensor].Readings.Channel_DiagnosticsBits AND
                                             DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Channel_DiagnAlarmMask);
          IF ( unmaskedChannelDiagnosticsBits <> B#16#0 ) THEN
            HumidSensor_isOk := FALSE;
          END_IF;
//--- do not care if diagnostic information is available for Channel
//    in case Mask bits are all zero
        ELSIF ( DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Channel_DiagnAlarmMask <> 2#0000_0000 ) THEN
          HumidSensor_isOk := FALSE;
        END_IF;

//--- raise Alarm if calibrated Temperature has been within Alarm range for Time > Delay,
//    reference Temperature information is unavailable,
//    Dew-Point computation failed
//    or Channel diagnostics indicate Alarm condition
        Channel_isOk := (HumidSensor_isOk AND refTempSensor_isOk AND DewPoint_isValid);
        computeAlarmStatus(currState_isOk := Channel_isOk
                  // IN: BOOL
                         ,Alarm_Delay := DB111.HumidSensor[iHumidSensor].ConfigParameters_extend.Alarm_Delay // IN: TIME
                         ,Acknowledge_Command := DB211.HumidSensor[iHumidSensor].Commands.Alarm_Acknowledge  // IN: BOOL
                         ,lastTimeStateOk := DB411.HumidSensor[iHumidSensor].Readings.Alarm_lastTimeSensorOk // IN_OUT: TIME_OF_DAY
                         ,Alarm_isActive := DB411.HumidSensor[iHumidSensor].Readings.Alarm_isActive          // IN_OUT: BOOL
                         ,Alarm_beenActive := DB411.HumidSensor[iHumidSensor].Readings.Alarm_beenActive      // IN_OUT: BOOL
                          );

        IF ( DB411.HumidSensor[iHumidSensor].Readings.Alarm_isActive ) THEN
          numHumidSens_inAlarm := numHumidSens_inAlarm + 1;
        END_IF;

        numHumidSens_CfgPar_set := numHumidSens_CfgPar_set + 1;
      ELSE
//--- in case no valid Configuration Parameters have been set,
//    clear all Readings in DB 411, except for Configuration Parameter Status and Error flags

        FILL_ReturnValue := SFC21(BVAL := DB4.zero                                    // IN : ANY
                                 ,BLK  := DB411.HumidSensor[iHumidSensor].Readings // OUT : ANY
                                  );

        numHumidSens_CfgPar_Err := numHumidSens_CfgPar_Err + 1;

      END_IF;

      numHumidSens_processed := numHumidSens_processed + 1;
```

# Page: 133

Number: 1 Author: jbk Subject: Sticky Note Date: 4/24/2008 4:44:15 PM

isn't total processed = _set + _Err ?

```
180
181 //--- reset Alarm "ACKNOWLEDGE" command
182     DB211.HumidSensor[iHumidSensor].Commands.Alarm_Acknowledge := FALSE;
183   END_FOR;
184
185 //--- store HMX Humidity Sensor status information in DB 411
186   DB411.NumHumidSens_processed := numHumidSens_processed;
187   DB411.NumHumidSens_inAlarm := numHumidSens_inAlarm;
188   DB411.NumHumidSens_CfgPar_set := numHumidSens_CfgPar_set;
189   DB411.NumHumidSens_CfgPar_Err := numHumidSens_CfgPar_Err;
190   DB411.anyHumidSens_CfgPar_Err := (numHumidSens_CfgPar_Err > 0);
191   DB411.allHumidSens_CfgPar_set := (numHumidSens_CfgPar_set = numHumidSens_processed);
192
193 //--- compute CRC16 check-sum for "Readings" DB 411
194   CRC16_CheckSum_DB411 := computeCRC16CheckSum(Sptr := DB411);
195   DB411.Header.CRC16 := CRC16_CheckSum_DB411;
196
197 END_FUNCTION
198
```

1

The if-then-else structure these functions is somewhat harder to follow than it needs to be. One way to improve this condition is to make channel processing functions and just have these function drive over the list of channels.

It looks like the names are common amongst the reading functions, so there might be more opportunities to make common handling functions.

# Page: 134

Number: 1 Author: jbk Subject: Text Box Date: 4/24/2008 4:49:22 PM

The if-then-else structure these functions is somewhat harder to follow than it needs to be. One way to improve this condition is to make channel processing functions and just have these function drive over the list of channels.

It looks like the names are common amongst the reading functions, so there might be more opportunities to make common handling functions.