Project properties

```
Database version    : 3/5/2009 05:25:26
PLC related version : 3/5/2009 05:25:26
Global DFB Path     : C:\CONCEP~3\DFB\
Secure Application  : No
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section Announce_Alarms *)

(* First version: W. Brooks, Dec 8, 1997 *)

(* Modifications:
      None so far *)

(* This section organizes the information sent
to the annunciator boxes. Each individual bit which
turns on a light is packed together into a single
16-bit word which is sent to the TIO modules in the
annunciator boxes. *)

IF do_announce_alarms THEN

  (* First check alarm acknowledge button. Then announce
  the alarm. *)
  (* As a parenthetical note, if there is no alarm, both the alarm_acknowledged_lamp
  and the acknowledge_alarm_lamp are OFF. If there is an alarm of any type, EITHER one
  OR the other are on, but never both on or both off. *)
  IF adc4[7] > 500 THEN   (* Alarm acknowledge button is currently pushed in. *)
    alarm_acknowledged_lamp := 1 ;
    acknowledge_alarm_lamp := 0 ;
    audio_alarm := 0 ;
  END_IF  ;

  (* The following is for simulating the code response in the
  Concept plc simulator *)
  IF simulate=1 THEN
    IF simulate_alarm_acknowledge=1 THEN
      alarm_acknowledged_lamp := 1 ;
      acknowledge_alarm_lamp := 0 ;
      audio_alarm := 0 ;
    END_IF  ;
  END_IF  ;

  (* The following word is written to the upper TIO
  module in the annunciator boxes. Actually writing the
  word is handled by the 'peer cop' option in the
  configurator. *)
  annunciator_output_1 := (*65535;*)
      bit_to_word( sniffer_lamp,
                   crate_monitoring_lamp,
                   level0_forwardcarriage_lamp,
                   level1_forwardcarriage_lamp,
                   level2_forwardcarriage_lamp,
                   level3_forwardcarriage_lamp,
                   level0_south_sidecarriage_lamp,
                   level1_south_sidecarriage_lamp,
                   level2_south_sidecarriage_lamp,
                   level3_south_sidecarriage_lamp,
                   level0_north_sidecarriage_lamp,
                   level1_north_sidecarriage_lamp,
                   level1_south_spaceframe_lamp,
                   level0_north_spaceframe_lamp,
                   level1_north_spaceframe_lamp,
```

> page 3

```
                level2_north_spaceframe_lamp ) ;

  (* The following word is written to the lower TIO
  module in the annunciator boxes. *)
  annunciator_output_2 :=     (*65535; *)
     bit_to_word( spare_lamp3,
                spare_lamp2,
                spare_lamp1,
                audio_alarm,
                trouble_lamp,
                acknowledge_alarm_lamp,
                alarm_acknowledged_lamp,
                no_alarms_lamp,
                warning_lamp,
                alarm_lamp,
                forwardcarriage_VESDA_lamp,
                spaceframe_VESDA_lamp,
                sidecarriage_VESDA_lamp,
                return_air_duct_VESDA_lamp,
                pie_tower_VESDA_lamp,
                fire_wire_lamp ) ;


  (* In the following line, activate pager output number 2 if
  the trouble light is lit. *)
    dialer_relay_2 := 0 ;
  (*IF trouble_lamp = 1 THEN
    dialer_relay_2 := 1 ;
  END_IF ;
 Mod Dec 13 2001 W. Brooks. Now have multipaging option *)

  (* In the next line, turn off the trouble lamp
  indicator bit. If there is a persistent trouble condition,
  this bit will be reset to 1 before it reaches this section. The
  ultimate effect is that the trouble lamp is not latched on,
  but is only persistently on if there is a persistent trouble
  condition. *)
  trouble_lamp := 0 ;


  IF dialer THEN
    dialer_relay_1 := 1 ;
  ELSE
    dialer_relay_1 := 0 ;
  END_IF  ;

END_IF  ;


»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section Evaluate_Alarm_Status *)

(* First version: W. Brooks, Dec 4, 1997 *)

(* Modifications:
      Added audio alarm for warning (yellow) as well
      as alarm (red) W. Brooks, 29 Jan 1998
      Added SNIFFER, 17 Sept 1999 W. Brooks *)

(* This section decides whether the alarm status
is No_Alarms, Warning, or Alarm, based on combinations
of input information. The variables ending with "_lamp"
are Boolean indications of whether the lamp is lit or
not. *)

(* The philosophy here is that the alarm level lamps
show the PRESENT STATE of the system; the 'alarm
acknowledge' lamp and the audio alarm show the status
of the acknowledgement of an alarm. *)

IF do_evaluate_alarm_status THEN

  (* Identify the maximum alarm level from
  each system: *)

  itest := 0 ;

  FOR VESDA_location := 1 TO 4 DO
    IF VESDA_alarm_status[VESDA_location] > itest THEN
      itest := VESDA_alarm_status[VESDA_location] ;
      alarm_id := int_to_uint(VESDA_location) ;
    END_IF ;
  END_FOR ;

  FOR firewire_location := 1 TO 8 DO
    IF firewire_alarm_status_1[firewire_location] > itest THEN (* Adc #1 *)
      itest := firewire_alarm_status_1[firewire_location] ;
      alarm_id := 10 ;
    END_IF ;
    IF firewire_alarm_status_2[firewire_location] > itest THEN (* Adc #2 *)
      itest := firewire_alarm_status_2[firewire_location] ;
      alarm_id := 10 ;
    END_IF ;
    IF firewire_alarm_status_3[firewire_location] > itest THEN (* Adc #3 *)
      itest := firewire_alarm_status_3[firewire_location] ;
      alarm_id := 10 ;
    END_IF ;
  END_FOR ;

  FOR SNIFFER_location := 1 TO number_of_active_valves DO
    IF SNIFFER_alarm_status[SNIFFER_location] > itest THEN
      itest := SNIFFER_alarm_status[SNIFFER_location] ;
      alarm_id := int_to_uint(SNIFFER_location) + 10 ;
    END_IF ;
  END_FOR ;
```

> page 5

> page 4

```
(* Assign the alarm level based on the above: *)

(* Don't dial pager unless there is a current alarm or warning *)
dialer                   :=   0 ;

IF itest = 0 THEN              (* Light the green lamp *)
  no_alarms_lamp          :=  1 ;
  warning_lamp            :=  0 ;
  alarm_lamp              :=  0 ;
  alarm_acknowledged_lamp  := 0 ;
  audio_alarm             :=  0 ;
  warning_lamp_timer       :=  0 ;
  alarm_lamp_timer         :=  0 ;
  dialer                  :=  0 ;
  broadcast_alarm_id       :=  0 ;

ELSIF  itest = 1 THEN             (* Light the yellow lamp *)

  (* Delay announcing the alarm in order to ignore transient alarms *)

  IF (warning_lamp_timer = 0 ) THEN  (* First time condition has been noted *)
     (* Capture the present time into the timer variable *)
     warning_lamp_timer  := timer_register ;
     (* Other than setting the above flag, ignore the warning condition *)

  ELSE  (* Condition has already been flagged, check how long since it was flagged *)

     (* The following scheme only works for time delays of
     less than 65535/100 seconds, or 10.92 minutes. *)
     test_dint := uint_to_dint(timer_register) - uint_to_dint(warning_lamp_timer) ;
     IF test_dint < 0 THEN   (* timing register rolled over, => compensate *)
       test_dint := test_dint + 65535 ;
       time_difference_1 := dint_to_uint(test_dint) ;
     ELSE
       time_difference_1 := dint_to_uint(test_dint) ;
     END_IF  ;

     (* The following number specifies the allowed time delay in milliseconds. *)
     IF time_difference_1 > announce_delay THEN

        (* We've delayed long enough, now act on the warning: *)
        no_alarms_lamp          :=  0 ;
        warning_lamp            :=  1 ;
        alarm_lamp              :=  0 ;
        dialer                  :=  1 ;
        broadcast_alarm_id       :=  alarm_id; (* posts alarm_id to visible register *)
        IF acknowledge_alarm_lamp = 0 AND alarm_acknowledged_lamp = 0 THEN   (* First time alarm has been noted *)
           audio_alarm           :=  1 ;     (* Turn on audio alarm for yellow alarm *)
           acknowledge_alarm_lamp := 1 ;     (* Turn on alarm acknowledge request *)
        END_IF  ;
        IF acknowledge_alarm_lamp = 1 AND alarm_acknowledged_lamp = 0 THEN   (* Alarm not yet acknowledged *)
          audio_alarm           :=  1 ;     (* keep audio alarm for yellow alarm on - this case may not be needed *)
        END_IF  ;
        IF acknowledge_alarm_lamp = 0 AND alarm_acknowledged_lamp = 1 THEN   (* Alarm has been acknowledged *)
          audio_alarm           :=  0 ;     (* Turn off audio alarm *)
        END_IF  ;

     END_IF  ;
```

> page 6

```
   END_IF ;

 ELSIF itest = 2 THEN              (* Light the red lamp *)

   (* Delay announcing the alarm in order to ignore transient alarms *)

   IF (alarm_lamp_timer = 0 ) THEN (* First time condition has been noted *)
      (* Capture the present time into the timer variable *)
      alarm_lamp_timer  := timer_register ;
      (* Other than setting the above flag, ignore the warning condition *)

   ELSE (* Condition has already been flagged, check how long since it was flagged *)

      (* In the following, timer_register is incremented every
      10 ms by the PLC. Therefore 100 such counts is a one-second
      timer. The following scheme only works for time delays of
      less than 65535/100 seconds, or 10.92 minutes. *)
      time_difference_1 := timer_register - alarm_lamp_timer ;
      IF time_difference_1 < 0 THEN  (* timing register rolled over, => compensate *)
        time_difference_1 := time_difference_1 + 65535 ;
      END_IF ;

      (* The following number specifies the allowed time delay in milliseconds. *)
      IF time_difference_1 > announce_delay THEN

         (* We've delayed long enough, now act on the alarm: *)
         no_alarms_lamp          :=  0 ;
         warning_lamp            :=  0 ;
         alarm_lamp              :=  1 ;
         dialer                  :=  1 ;
         broadcast_alarm_id      :=  alarm_id; (* posts alarm_id to visible register *)
         IF acknowledge_alarm_lamp = 0 AND alarm_acknowledged_lamp = 0 THEN  (* First time alarm has been noted *)
           audio_alarm           :=  1 ;     (* Turn on audio alarm for red alarm *)
           acknowledge_alarm_lamp := 1 ;     (* Turn on alarm acknowledge request *)
         END_IF ;
         IF acknowledge_alarm_lamp = 1 AND alarm_acknowledged_lamp = 0 THEN  (* Alarm not yet acknowledged *)
            audio_alarm          :=  1 ;     (* keep audio alarm for red alarm on - this case may not be needed *)
         END_IF ;
         IF acknowledge_alarm_lamp = 0 AND alarm_acknowledged_lamp = 1 THEN  (* Alarm has been acknowledged *)
            audio_alarm          :=  0 ;     (* Turn off audio alarm *)
         END_IF ;

      END_IF ;

   END_IF ;

 END_IF ;

END_IF ;
»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section Evaluate_System_Status *)

(* First version: W. Brooks, Dec 8, 1997 *)

(* Modifications:
      Add SNIFFER Sept 99 W. Brooks *)


(* This section decides which system has initiated
any alarms. If no alarm has been flagged, this section
is not executed. *)

IF do_evaluate_alarm_status THEN

  IF no_alarms_lamp = 0 THEN
  (* An alarm was identified in the previous section,
  therefore proceed further to identify WHICH system
  originated the alarm. *)

  (* First, do the VESDA systems. These combine system
  information with location information, since there are
  four separate systems covering four locations. *)

    IF VESDA_alarm_status[1] > 0 THEN
      forwardcarriage_VESDA_lamp := 1 ;
    END_IF ;

    IF VESDA_alarm_status[2] > 0 THEN
      sidecarriage_VESDA_lamp := 1 ;
    END_IF ;

    IF VESDA_alarm_status[3] > 0 THEN
      return_air_duct_VESDA_lamp := 1 ;
    END_IF ;

    IF VESDA_alarm_status[4] > 0 THEN
      spaceframe_VESDA_lamp := 1 ;
    END_IF ;

  (* Next, do the fire wire. *)

    itest := 0 ;
    FOR firewire_location := 1 TO 8 DO
      IF firewire_alarm_status_1[firewire_location] > itest THEN (* Adc #1 *)
        itest := firewire_alarm_status_1[firewire_location] ;
      END_IF ;
      IF firewire_alarm_status_2[firewire_location] > itest THEN (* Adc #2 *)
        itest := firewire_alarm_status_2[firewire_location] ;
      END_IF ;
      IF firewire_alarm_status_3[firewire_location] > itest THEN (* Adc #3 *)
        itest := firewire_alarm_status_3[firewire_location] ;
      END_IF ;
    END_FOR ;
    IF itest > 0 THEN
      fire_wire_lamp := 1 ; (* Turn on fire wire lamp *)
    END_IF ;
```

> page 8

> page 7

```
(* Next, do SNIFFER *)

itest := 0 ;
FOR SNIFFER_location := 1 TO number_of_active_valves DO
  IF SNIFFER_alarm_status [SNIFFER_location] > 0 THEN
      itest := 1 ;
  END_IF ;
END_FOR ;
IF itest > 0 THEN
  sniffer_lamp := 1 ;
END_IF ;

ELSIF acknowledge_alarm_lamp=0 THEN (* This condition latches unacknowledged alarms *)
(* Presently there are no active alarms; make sure all
locations' lamps are turned off, unless there was previously an alarm which was
not acknowledged. Include all possible locations here, regardless of whether they
are connected to a lamp at a particular time. *)

spaceframe_VESDA_lamp         :=  0 ;
forwardcarriage_VESDA_lamp    :=  0 ;
sidecarriage_VESDA_lamp       :=  0 ;
return_air_duct_VESDA_lamp    :=  0 ;

level0_north_spaceframe_lamp  := 0 ;
level1_north_spaceframe_lamp  := 0 ;
level2_north_spaceframe_lamp  := 0 ;
level3_north_spaceframe_lamp  := 0 ;

level0_south_spaceframe_lamp  := 0 ;
level1_south_spaceframe_lamp  := 0 ;
level2_south_spaceframe_lamp  := 0 ;
level3_south_spaceframe_lamp  := 0 ;

level0_south_sidecarriage_lamp := 0 ;
level1_south_sidecarriage_lamp := 0 ;
level2_south_sidecarriage_lamp := 0 ;
level3_south_sidecarriage_lamp := 0 ;

level0_north_sidecarriage_lamp := 0 ;
level1_north_sidecarriage_lamp := 0 ;
level2_north_sidecarriage_lamp := 0 ;
level3_north_sidecarriage_lamp := 0 ;

level0_forwardcarriage_lamp   := 0 ;
level1_forwardcarriage_lamp   := 0 ;
level2_forwardcarriage_lamp   := 0 ;
level3_forwardcarriage_lamp   := 0 ;

level0_pie_tower_lamp         :=  0 ;
level1_pie_tower_lamp         :=  0 ;
level2_pie_tower_lamp         :=  0 ;
level3_pie_tower_lamp         :=  0 ;

sniffer_lamp                 :=   0 ;
crate_monitoring_lamp         :=  0 ;
fire_wire_lamp               :=   0 ;
```

> page 9

```
  spare_lamp1                 :=   0 ;
  spare_lamp2                 :=   0 ;
  spare_lamp3                 :=   0 ;

  END_IF ;

END_IF ;


»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section Fire_Wire *)

(* First version: W. Brooks, Dec 4, 1997 *)

(* Modifications:
     Wrote the code for this section, W. Brooks Feb 4 1998

(* This section checks to see if any of the "fire wire"
(linear heat sensor) channels are reading a bad value.
If the currents exceed the upper
limit, it sets the appropriate value for the alarm status.
If the currents are below the lower limit, it sets the
trouble lamp on ('broken wire'). *)

IF do_fire_wire THEN

  FOR firewire_location := 1 TO 8 DO  (* Look at all active locations *)

    (* ADC number 1 *)
    IF active_firewire_1[firewire_location] = 1 THEN        (* check that this wire is active *)
      IF adc1[firewire_location] > alarm_limit_firewire_1[firewire_location] THEN
        firewire_alarm_status_1[firewire_location] := 2 ;   (* alarm level *)
      ELSIF adc1[firewire_location] <= lower_limit_firewire_1[firewire_location] THEN
        trouble_lamp := 1 ;
        alarm_id := 31;                            (* trouble*)
      ELSE
        firewire_alarm_status_1[firewire_location] := 0 ;   (* no alarm *)
      END_IF ;
    END_IF ;

    (* ADC number 2 *)
    IF active_firewire_2[firewire_location] = 1 THEN        (* check that this wire is active *)
      IF adc2[firewire_location] > alarm_limit_firewire_2[firewire_location] THEN
        firewire_alarm_status_2[firewire_location] := 2 ;   (* alarm level *)
      ELSIF adc2[firewire_location] <= lower_limit_firewire_2[firewire_location] THEN
        trouble_lamp := 1 ;
        alarm_id := 32;                              (* trouble*)
      ELSE
        firewire_alarm_status_2[firewire_location] := 0 ;   (* no alarm *)
      END_IF ;
    END_IF ;

    (* ADC number 3 *)
    IF active_firewire_3[firewire_location] = 1 THEN        (* check that this wire is active *)
      IF adc3[firewire_location] > alarm_limit_firewire_3[firewire_location] THEN
        firewire_alarm_status_3[firewire_location] := 2 ;   (* alarm level *)
      ELSIF adc3[firewire_location] <= lower_limit_firewire_3[firewire_location] THEN
        trouble_lamp := 1 ;
        alarm_id := 33;                              (* trouble*)
      ELSE
        firewire_alarm_status_3[firewire_location] := 0 ;   (* no alarm *)
      END_IF ;
    END_IF ;

  END_FOR ;
```

> page 11

```
  (* The following is for simulating the code response in the
  Concept plc simulator *)
  IF simulate=1 THEN
    firewire_alarm_status_3[4] := simulate_firewire_status ;
  END_IF ;

END_IF ;

»»»»»»»»»»»» Structured Text End ««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section Gas_Concentrations *)

(* First version: W. Brooks, Dec 4, 1997 *)

(* Modifications: fill in first-pass logic, Sept 98 WB. *)

(* This section checks to see if the concentrations exceed the lower
limit, the warning upper limit, or the alarm upper limit, and
sets the appropriate value for the alarm status. *)

(* In the following it is assumed that the alarm limits for the
'alarm' level are greater than those for the 'warning' level. *)

IF do_gas_concentrations THEN      (* Toggle on or off executions of this section. *)

  IF initialize_SNIFFER = 0 AND liquid_nitrogen_is_filling = 0 THEN    (* Don't check for alarms during initialization or LN2 fill *)

    (* Could use abs() below, but I couldn't get it to work *)

    (* Let's try this instead. -- Doug Curry *)
(*  IF CH_concentration < 0.0 THEN
       CH_concentration := CH_concentration * -1.0;
    END_IF; *)

   test := CH_warning_limit_SNIFFER[sampling_valve]-CH_concentration ;

    diagnostic1 := CH_scale_factor * CH_warning_limit_SNIFFER[sampling_valve] ;

       IF CH_concentration > CH_scale_factor * CH_warning_limit_SNIFFER[sampling_valve] THEN
         CH_alarmed := 1 ;
         alarm_level_history := CH_concentration ;
         alarm_zone_history := sampling_valve ;
       END_IF ;

    diagnostic2 := CO_scale_factor * 4.0 * CO_concentration_error ;
       IF CO_concentration > CO_scale_factor * 4.0 * CO_concentration_error THEN
         CO_alarmed := 1 ;
         alarm_level_history := CO_concentration ;
         alarm_levelerror_history := CO_concentration_error ;
         alarm_zone_history := sampling_valve ;
       END_IF ;

    IF CH_concentration > CH_scale_factor * CH_warning_limit_SNIFFER[sampling_valve] OR
       CO_concentration > CO_scale_factor * 4.0 * CO_concentration_error THEN

          SNIFFER_alarm_status[sampling_valve] := 1 ;   (* warning level *)


    ELSE

          SNIFFER_alarm_status[sampling_valve] := 0 ;   (* no_alarm level *)

    END_IF ;

    IF    CH_concentration > CH_scale_factor * CH_alarm_limit_SNIFFER[sampling_valve] OR
          CO_concentration > CO_scale_factor * 5.0 * CO_concentration_error THEN
```

> page 13

```
        SNIFFER_alarm_status[sampling_valve] := 2 ;   (* alarm level *)

    END_IF ;

    IF CH_alarm_limit_SNIFFER[sampling_valve] < CH_warning_limit_SNIFFER[sampling_valve] THEN
      trouble_lamp := 1; (* The above algorithm relies on this condition *)
      alarm_id := 30;
    END_IF ;

  END_IF ;

END_IF ;


»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section Identify_Alarm_Locations *)

(* First version: W. Brooks, Dec 4, 1997 *)

(* Modifications:
      W. Brooks, 7 Feb 1997, fill in the actual code
      for the first time.
      W. Brooks, 17 Sept 1999, add SNIFFER *)

(* This section identifies the location of any alarm,
within the constraints imposed by the alarm annunciator
boxes. *)

IF do_identify_alarm_locations THEN

  (* VESDA's are already done in section Evaluate_System_Status *)

  (* Do firewire: *)

  (* -------------- adc # 1 -------------- *)

  IF adc1[1]>alarm_limit_firewire_1[1] THEN (* DC #5 *)
    level1_south_spaceframe_lamp := 1 ;
  END_IF ;

  IF adc1[2]>alarm_limit_firewire_1[2] THEN (* DC #6 *)
    level1_south_spaceframe_lamp := 1 ;
  END_IF ;

  IF adc1[3]>alarm_limit_firewire_1[3] THEN (* not used *)
    test := 1.0 ; (* do nothing *)
  END_IF ;

  IF adc1[4]>alarm_limit_firewire_1[4] THEN (* space frame level 3 north *)
    level3_north_spaceframe_lamp := 1 ;
  END_IF ;

  IF adc1[5]>alarm_limit_firewire_1[5] THEN (* space frame level 1 *)
    level1_south_spaceframe_lamp := 1 ;
    level1_north_spaceframe_lamp := 1 ;
  END_IF ;

  IF adc1[6]>alarm_limit_firewire_1[6] THEN (* space frame level 3 south *)
    level3_south_spaceframe_lamp := 1 ;
  END_IF ;

  IF adc1[7]>alarm_limit_firewire_1[7] THEN (* south space frame vertical run *)
    level1_south_spaceframe_lamp := 1 ;
    level2_south_spaceframe_lamp := 1 ;
    level3_south_spaceframe_lamp := 1 ;
  END_IF ;

  IF adc1[8]>alarm_limit_firewire_1[8] THEN (* north space frame vertical run *)
    level1_north_spaceframe_lamp := 1 ;
    level2_north_spaceframe_lamp := 1 ;
    level3_north_spaceframe_lamp := 1 ;
```

> page 15

> page 14

```
END_IF ;

(* -------------- adc # 2 -------------- *)

IF adc2[1]>alarm_limit_firewire_2[1] THEN (* not used *)
  test := 1.0 ;
END_IF ;

IF adc2[2]>alarm_limit_firewire_2[2] THEN (* not used *)
  test := 1.0 ;
END_IF ;

IF adc2[3]>alarm_limit_firewire_2[3] THEN (* not used *)
  test := 1.0 ;
END_IF ;

IF adc2[4]>alarm_limit_firewire_2[4] THEN (* space frame level 2 north *)
  level2_north_spaceframe_lamp := 1 ;
END_IF ;

IF adc2[5]>alarm_limit_firewire_2[5] THEN (* not used *)
  test := 1.0 ;
END_IF ;

IF adc2[6]>alarm_limit_firewire_2[6] THEN (* not used *)
  test := 1.0 ;
END_IF ;

IF adc2[7]>alarm_limit_firewire_2[7] THEN (* DC #1 *)
  level1_south_spaceframe_lamp := 1 ;
END_IF ;

IF adc2[8]>alarm_limit_firewire_2[8] THEN (* not used *)
  test := 1.0 ;
END_IF ;

(* -------------- adc # 3 -------------- *)

IF adc3[1]>alarm_limit_firewire_3[1] THEN (* not used *)
  test := 1.0 ;
END_IF ;

IF adc3[2]>alarm_limit_firewire_3[2] THEN (* not used *)
  test := 1.0 ;
END_IF ;

IF adc3[3]>alarm_limit_firewire_3[3] THEN (* forward carriage level 2 *)
  level2_forwardcarriage_lamp := 1 ;
END_IF ;

IF adc3[4]>alarm_limit_firewire_3[4] THEN (* forward carriage level 1 *)
  level1_forwardcarriage_lamp := 1 ;
END_IF ;

IF adc3[5]>alarm_limit_firewire_3[5] THEN (* forward carriage level 3 *)
  level3_forwardcarriage_lamp := 1 ;
END_IF ;
```

> page 16

> page 15

```
IF adc3[6]>alarm_limit_firewire_3[6] THEN (* forward carriage north vertical run *)
  level1_forwardcarriage_lamp := 1 ;
  level2_forwardcarriage_lamp := 1 ;
  level3_forwardcarriage_lamp := 1 ;
END_IF ;

IF adc3[7]>alarm_limit_firewire_3[7] THEN (* forward carriage south vertical run *)
  level1_forwardcarriage_lamp := 1 ;
  level2_forwardcarriage_lamp := 1 ;
  level3_forwardcarriage_lamp := 1 ;
END_IF ;

IF adc3[8]>alarm_limit_firewire_3[8] THEN (* not used *)
      test := 1.0 ;
END_IF ;

(* End of fire wire *)

(* Do VESDA: *)

(* THIS SECTION IS NOT COMPLETE - Sept 99 *)

IF SNIFFER_alarm_status[1] > 0 THEN
  level3_north_spaceframe_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[2] > 0 THEN
  level3_south_spaceframe_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[3] > 0 THEN
  level2_north_spaceframe_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[4] > 0 THEN
  level2_south_spaceframe_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[5] > 0 THEN
  level1_north_spaceframe_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[6] > 0 THEN
  level1_south_spaceframe_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[7] > 0 THEN
  level0_south_spaceframe_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[8] > 0 THEN
  level1_south_sidecarriage_lamp := 1;
END_IF ;

IF SNIFFER_alarm_status[9] > 0 THEN
  level1_north_sidecarriage_lamp := 1;
END_IF ;
```

> page 17

```
  IF SNIFFER_alarm_status[10] > 0 THEN
    level2_forwardcarriage_lamp := 1;
  END_IF ;

  IF SNIFFER_alarm_status[11] > 0 THEN
    level1_forwardcarriage_lamp := 1;
  END_IF ;

  IF SNIFFER_alarm_status[12] > 0 THEN
    level0_north_spaceframe_lamp := 1;
  END_IF ;

END_IF  ;

»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* First version: W. Brooks, 4 Nov 1999

This section sets the initial values of all
variables. This function was formerly performed
by using the initial values in the variable
declarations menu. However, several times the
program somehow 'forgot' these inital values,
causing alarms. This section hardwires those
values. The forgetting seems to happen only
to numerical arrays. *)

(* Changed VESDA warning and alarm limits to be compatible
with new high-resolution ADC. W. Brooks, Jan 02 *)

IF do_initialize_variables THEN

  do_initialize_variables := 0 ;

  active_firewire_1[1] := 0 ;
  active_firewire_1[2] := 0 ;
  active_firewire_1[3] := 0 ;
  active_firewire_1[4] := 1 ;
  active_firewire_1[5] := 1 ;
  active_firewire_1[6] := 1 ;
  active_firewire_1[7] := 1 ;
  active_firewire_1[8] := 1 ;

  active_firewire_2[1] := 0 ;
  active_firewire_2[2] := 0 ;
  active_firewire_2[3] := 0 ;
  active_firewire_2[4] := 1 ;
  active_firewire_2[5] := 0 ;
  active_firewire_2[6] := 0 ;
  active_firewire_2[7] := 0 ;
  active_firewire_2[8] := 0 ;

  active_firewire_3[1] := 0 ;
  active_firewire_3[2] := 0 ;
  active_firewire_3[3] := 1 ;
  active_firewire_3[4] := 1 ;
  active_firewire_3[5] := 1 ;
  active_firewire_3[6] := 1 ;
  active_firewire_3[7] := 1 ;
  active_firewire_3[8] := 0 ;


  alarm_limit_firewire_1[1] := 0 ;
  alarm_limit_firewire_1[2] := 0 ;
  alarm_limit_firewire_1[3] := 0 ;
  alarm_limit_firewire_1[4] := 3000 ;
  alarm_limit_firewire_1[5] := 2500 ;
  alarm_limit_firewire_1[6] := 2500 ;
  alarm_limit_firewire_1[7] := 2500 ;
  alarm_limit_firewire_1[8] := 3000 ;

  alarm_limit_firewire_2[1] := 0 ;
```

> page 19

> page 18

```
 alarm_limit_firewire_2[2] := 0 ;
 alarm_limit_firewire_2[3] := 0 ;
 alarm_limit_firewire_2[4] := 2500 ;
 alarm_limit_firewire_2[5] := 0 ;
 alarm_limit_firewire_2[6] := 0 ;
 alarm_limit_firewire_2[7] := 3000 ;
 alarm_limit_firewire_2[8] := 0 ;

 alarm_limit_firewire_3[1] := 0 ;
 alarm_limit_firewire_3[2] := 0 ;
 alarm_limit_firewire_3[3] := 2000 ;
 alarm_limit_firewire_3[4] := 3000 ;
 alarm_limit_firewire_3[5] := 2000 ;
 alarm_limit_firewire_3[6] := 2500 ;
 alarm_limit_firewire_3[7] := 2500 ;
 alarm_limit_firewire_3[8] := 0 ;

 lower_limit_firewire_1[1] := 0 ;
 lower_limit_firewire_1[2] := 0 ;
 lower_limit_firewire_1[3] := 0 ;
 lower_limit_firewire_1[4] := 10 ;
 lower_limit_firewire_1[5] := 10 ;
 lower_limit_firewire_1[6] := 10 ;
 lower_limit_firewire_1[7] := 10 ;
 lower_limit_firewire_1[8] := 10 ;

 lower_limit_firewire_2[1] := 0 ;
 lower_limit_firewire_2[2] := 0 ;
 lower_limit_firewire_2[3] := 0 ;
 lower_limit_firewire_2[4] := 10 ;
 lower_limit_firewire_2[5] := 0 ;
 lower_limit_firewire_2[6] := 0 ;
 lower_limit_firewire_2[7] := 10 ;
 lower_limit_firewire_2[8] := 0 ;

 lower_limit_firewire_3[1] := 0 ;
 lower_limit_firewire_3[2] := 0 ;
 lower_limit_firewire_3[3] := 10 ;
 lower_limit_firewire_3[4] := 10 ;
 lower_limit_firewire_3[5] := 10 ;
 lower_limit_firewire_3[6] := 10 ;
 lower_limit_firewire_3[7] := 10 ;
 lower_limit_firewire_3[8] := 0 ;

 warning_limit_VESDA[1] := 37568 ;
 warning_limit_VESDA[2] := 37568 ;
 warning_limit_VESDA[3] := 37568 ;
 warning_limit_VESDA[4] := 37568 ;

 alarm_limit_VESDA[1] := 39168 ;
 alarm_limit_VESDA[2] := 39168 ;
 alarm_limit_VESDA[3] := 39168 ;
 alarm_limit_VESDA[4] := 39168 ;

(* Set CH warning limit to 1.00 from nom. 0.15 on South Clamshell, s.c. 11/11/08*)
(* Set CH warning limit back to nom. 0.15 on South Clamshell, s.c. 2/5/09*)

 CH_warning_limit_SNIFFER[1]  :=  0.15 ;
```

> page 20

```
 CH_warning_limit_SNIFFER[2]  :=  0.15 ;
 CH_warning_limit_SNIFFER[3]  :=  0.15 ;
 CH_warning_limit_SNIFFER[4]  :=  0.15 ;
 CH_warning_limit_SNIFFER[5]  :=  0.15 ;
 CH_warning_limit_SNIFFER[6]  :=  0.15 ;
 CH_warning_limit_SNIFFER[7]  :=  0.15 ;
 CH_warning_limit_SNIFFER[8]  :=  0.15 ;
 CH_warning_limit_SNIFFER[9]  :=  0.15 ;
 CH_warning_limit_SNIFFER[10]  :=  0.15 ;
 CH_warning_limit_SNIFFER[11]  :=  0.15 ;
 CH_warning_limit_SNIFFER[12]  :=  0.15 ;
 CH_warning_limit_SNIFFER[13]  :=  0.15 ;
 CH_warning_limit_SNIFFER[14]  :=  0.15 ;
 CH_warning_limit_SNIFFER[15]  :=  0.15 ;
 CH_warning_limit_SNIFFER[16]  :=  0.15 ;
 CH_warning_limit_SNIFFER[17]  :=  0.15 ;
 CH_warning_limit_SNIFFER[18]  :=  0.15 ;
 CH_warning_limit_SNIFFER[19]  :=  0.15 ;
 CH_warning_limit_SNIFFER[20]  :=  0.15 ;

 (*Set CH alarm limit to 1.50 from 0.2 on South Clamshell, 11/12/08*)
 (*Set CH alarm limit back to 0.2 on South Clamshell, 2/5/09 s.c.*)

 CH_alarm_limit_SNIFFER[1]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[2]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[3]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[4]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[5]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[6]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[7]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[8]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[9]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[10]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[11]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[12]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[13]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[14]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[15]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[16]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[17]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[18]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[19]  :=  0.2 ;
 CH_alarm_limit_SNIFFER[20]  :=  0.2 ;

 solenoid_valves_1[8] := 1;
 solenoid_valves_2[9] := 1;

END_IF ;


»»»»»»»»»»»» Structured Text End ««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* This section increments a counter each time one minute has passed *)

(* The following is a workaround for an address conflict with the timer register *)
(* It assumes a constant sweep time of 30 ms. I should fix this at some point. *)

  IF timer_register < 65506 THEN
    timer_register := timer_register + 30;
  ELSE
    timer_register := 0;
  END_IF ;

        IF minute_counter_timer = 0 THEN (* initialize *)
            minute_counter_timer := timer_register ;
            (* note, this method can cause rare 1-minute skips *)
        END_IF ;

       (* In the following, timer_register is incremented every
       10 ms by the PLC. Therefore 100 such counts is a one-second
       timer. The following scheme only works for time delays of
       less than 65535/100 seconds, or 10.92 minutes. *)
       test_dint := uint_to_dint(timer_register) - uint_to_dint(minute_counter_timer) ;
       IF test_dint < 0 THEN  (* timing register rolled over, => compensate *)
         test_dint := test_dint + 65535 ;
         time_difference_1 := dint_to_uint(test_dint) ;
       ELSE
         time_difference_1 := dint_to_uint(test_dint) ;
       END_IF  ;

       (* The following number specifies the allowed time delay in 1 ms units. *)
       IF time_difference_1 > 60000 THEN
         minute_counter_timer := timer_register ;
         minutes := minutes + 1 ;
         IF minutes = 65535 THEN (* roll over *)
           minutes := 0 ;
         END_IF  ;
       END_IF  ;


»»»»»»»»»»»»» Structured Text End ««««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* The purpose of this section is to simplify the process of setting all
thresholds to either a low value or a high value, through the action of
a single 4x register. The purpose is to permit a person not familiar with
the details of the program to change from low to high threshold, e.g., when
the hall goes to restricted access and truck access, etc. is expected to
occur.

  A second function of this routine is to limit the amount of time the
thresholds are set high. I.e., while someone may set them high for a particular
set of operations, in case they forget to set them back down, they should
automatically go back down after some time, e.g., 8 hours.

First version, April 2000, W. Brooks

Modify to set high thresholds to 'infinity.' Note there is still
some protection available via firewire. August 2001, W. Brooks

Increase the timeout to 10 hours from 8 hours for automatically going from
high threshold to low threshold, on request of tech staff. W. Brooks, 31 Oct 03.

7/18/06, s.c. This version is used for test purposes only. Sniffer will remain
in high threshold condition even after the 10 hr. timeout limit but VESDA will
always stay on, even when threshold is set high.

7/9/07, s.c. Reset to working condition. Modifying this program because the sniffer
seems to work better with this one rather than the older NT versions. Don't understand
why, though.

*)

IF do_set_thresholds THEN

  IF set_thresholds_high = 1 THEN

    (* Check how long thresholds have been set high *)
    IF (threshold_timer = 0 ) THEN  (* First time condition has been noted *)
       (* Capture the present time into the timer variable *)
       threshold_timer  :=  minutes; (* should be "minutes" *)

    ELSE  (* Condition has already been flagged, check how long since it was flagged *)

       (* In the following, minutes is incremented once per minute. *)
       test_dint := uint_to_dint(minutes) - uint_to_dint(threshold_timer) ;
       IF test_dint < 0 THEN   (* timing register rolled over, => compensate *)
         test_dint := test_dint + 65535 ;
         time_difference_1 := dint_to_uint(test_dint) ;
       ELSE
         time_difference_1 := dint_to_uint(test_dint) ;
       END_IF  ;

       (* If threshold has been high too long, reset to low threshold *)
       (* WB: change to 10 hours 10/31/03 - IF time_difference_1 > 480 THEN (* 480 is 8 hours *)
       IF time_difference_1 > 600 THEN  (* 480 is 8 hours *)
         set_thresholds_high := 0 ; (* should be 0 *)
       END_IF  ;
    END_IF  ;
```

> page 23

> page 22

```
    CO_scale_factor := 1000.0 ; (* turn off *)
    CH_scale_factor := 1000.0 ; (* turn off *)
    VESDA_scale_factor := 1 ; (* VESDA stays on now, s.c. 7/18/06 *)

  ELSE

    (* set scale factors to nominal *)
    (* CO & CH_scale factor s set to remain in high threshold, s.c. 7/18/06
    Reset to CO and CH_scale factors to 1, s.c. 7/9/07 *)
    CO_scale_factor := 1.0 ; (* should be 1 *)
    CH_scale_factor := 1.0 ; (* should be 1 *)
    VESDA_scale_factor := 1 ;

    threshold_timer := 0 ; (* reset timing counter *)

  END_IF ;

(* to turn sniffer alarm off, but permit it to cycle, and leave VESDA alarm on: *)

(*
CH_scale_factor := 1000.0 ;
CO_scale_factor := 1000.0 ;
*)

END_IF ;
»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* This section starts up the sniffer system
the first time after a PLC download or a PC
restart. When the PC starts up, it is supposed
to set the flag pc_ready = 2. Otherwise it is
=1 when ready and =0 when not ready. *)

(*      Modifications:
        6/16/00
        Doug Curry
        After refilling the dewer, we are going to start reading at the
        background area again

        8/17/01
        Will Brooks
        Add plc_ready :=0 so that PC stops analyzing until initialization is over

        9 July 2003
        W. Brooks
        Zero gas concentrations to make sure old/corrupt values never get read.

(* The code pumps on region 8 and 9 for a period
of time specified below, then switches to zones
9 and 10. Zone 9 is the background, so the background
is taken first before any other data is taken.*)

IF (pc_ready = 2) OR (initialize_SNIFFER > 0) THEN
  CH_concentration := 0.0 ;   (* zero values in register that are old or were perhaps acquired during LN2 fill *)
  CO_concentration := 0.0 ;
  CO_concentration_error := 1.0 ;
  IF pc_ready = 2 THEN
    initialize_SNIFFER := 1;  (* Either PC or PLC can initiate start of timer *)
    pc_ready := 1 ;
    sampling_valve := 9 ;     (* Return to prepumping the background area *)
    prepumping_valve := 10 ;   (* After initialization, these will increment *)

  END_IF ;

  IF initialize_SNIFFER = 1 THEN (* first time this has been noted *)
    initialize_SNIFFER_timer := timer_register ;
    initialize_SNIFFER := 2 ; (* Flags that we are now timing initialization *)
    sampling_valve := 8;      (* start back off in background area *)
    prepumping_valve := 9;    (* sampling and prepumping will increment in sniffer value
                              and should then become 9 and 10 *)
    time_difference_1 := 0;
  ELSE (* Condition has already been flagged, check how long since it was flagged *)
    test_dint :=    uint_to_dint(timer_register) - uint_to_dint(initialize_SNIFFER_timer) ;
    IF test_dint < 0 THEN  (* timing register rolled over, => compensate *)
      test_dint := test_dint + 65535 ;
      time_difference_1 := dint_to_uint(test_dint) ;
    ELSE
      time_difference_1 := dint_to_uint(test_dint) ;
    END_IF ;
    (* The following number specifies the allowed time delay in 1 ms units. *)
    IF time_difference_1 > 60000 THEN (* 1.5 minute delay, to allow pumping out of background region pipe *)
      initialize_SNIFFER := 0 ;
    END_IF ;
```

> page 25

> page 24

```
  END_IF ;
END_IF ;

»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section Sniffer_Valves *)

(* First version: W. Brooks, Dec 4, 1997 *)

(* Modifications:

      23 March 1998, W. Brooks
      Added synchronization between PC and PLC.
      2 May 2000, W. Brooks
      Moved output of valve values below the if loop
      which checks to see if the PC is ready. Previously,
      if the PC were not ready, it was not definite which
      valves were open.
      6/23/00 Doug Curry
      Took out the low dewer alarm.  This causes the alarm to
      go off anytime the dewer is being refilled.
      9 July 2003 W. Brooks
      Fixed bug that allows 'filling spikes'. Previously, on LN2 fill, PLC
      stopped cycling, initialized, and stopped checking gas concentrations,
      while PC finished taking data and wrote it to concentrations registers.
      This made it possible to assign concentrations taken during filling to zone
      8 or 9, and these concentrations can appear large because the filling
      distorts the background. Now, on the transition from filling to not filling,
      PLC simultaneously initializes and zeros out the old concentrations.
      30 October 2003 W. Brooks
      Undid Doug's change above; put the low dewer alarm back in, BUT now it is
      on the small dewer and it just stops the cycling rather than paging people.
      When transitioning back to the no-problem condition, re-initializes PLC so that
      a fresh background is taken.


(* This section checks to see if the Sniffer solenoid
valves need to be updated to sample a new area in Hall B.
This section checks the flag set by the FTIR spectrometer
PC; if the flag indicates the valves should be changed,
they're changed and a flag is set to indicate to the PC
that the change has been made. Control is then passed
to the next section. *)


IF do_sniffer_valves THEN

    (* Check to see if PC is ready to switch valves *)
    IF ( (pc_ready = 1) AND (wait_for_pc = 0)  AND (liquid_nitrogen_is_filling = 0) ) THEN

      (* Set flag indicating the PLC is busy *)
      plc_ready := 0;

     IF initialize_SNIFFER = 0 THEN

       (* close the solenoid valves presently open *)
       solenoid_valves_1[sampling_valve] := 0;
       solenoid_valves_2[prepumping_valve] := 0;

       (* Go to the next sampling valve/ prepumping valve combination *)
       sampling_valve := sampling_valve + 1 ;
```

> page 27

> page 26

```
     prepumping_valve := prepumping_valve + 1 ;

     (* Temporary to force system to stay in one area *)
     (*IF sampling_valve > 11 THEN
     sampling_valve := 11 ;
     prepumping_valve := 12 ;
     END_IF ;
     *)

     (* Temporary to skip one area with a bad relay *)
     (*IF sampling_valve = 10 THEN
     sampling_valve := 11 ;
     prepumping_valve := 12 ;
     END_IF ;
     *)

     (* Loop back from the last valve to the first, if needed *)
     IF sampling_valve > number_of_active_valves THEN
       (* maximum number of valves exceeded, go back to valve 1 *)
       sampling_valve := 1 ;
     END_IF ;
     IF prepumping_valve > number_of_active_valves THEN
       (* maximum number of valves exceeded, go back to valve 1 *)
       prepumping_valve := 1 ;
     END_IF ;

   END_IF ;

   (* step the solenoid valve output module to the next valve *)
   solenoid_valves_1[sampling_valve] := 1;
   solenoid_valves_2[prepumping_valve] := 1;

  IF initialize_SNIFFER = 0 THEN
    (* Set the flag indicating the PLC is no longer busy *)
    plc_ready := 1 ;

    (* Set the flag telling this program to wait for the PC to respond *)
    (* This is necessary because the pc may be so slow to respond that this section will be
    executed again and the valves switched, because pc_ready still = 1 *)
    wait_for_pc := 1 ;
  END_IF ;

END_IF ;

  (* output the above assignments to the output TIO modules: *)
solenoid_valves_output_1 :=
     bit_to_word( int_to_bool(solenoid_valves_1[16]),
                  int_to_bool(solenoid_valves_1[15]),
                  int_to_bool(solenoid_valves_1[14]),
                  int_to_bool(solenoid_valves_1[13]),
                  int_to_bool(solenoid_valves_1[12]),
                  int_to_bool(solenoid_valves_1[11]),
                  int_to_bool(solenoid_valves_1[10]),
                  int_to_bool(solenoid_valves_1[9]),
                  int_to_bool(solenoid_valves_1[8]),
                  int_to_bool(solenoid_valves_1[7]),
                  int_to_bool(solenoid_valves_1[6]),
                  int_to_bool(solenoid_valves_1[5]),
```

> page 28

> page 27

```
                int_to_bool(solenoid_valves_1[4]),
                int_to_bool(solenoid_valves_1[3]),
                int_to_bool(solenoid_valves_1[2]),
                int_to_bool(solenoid_valves_1[1]) ) ;

   solenoid_valves_output_2 :=
     bit_to_word( int_to_bool(solenoid_valves_2[16]),
                int_to_bool(solenoid_valves_2[15]),
                int_to_bool(solenoid_valves_2[14]),
                int_to_bool(solenoid_valves_2[13]),
                int_to_bool(solenoid_valves_2[12]),
                int_to_bool(solenoid_valves_2[11]),
                int_to_bool(solenoid_valves_2[10]),
                int_to_bool(solenoid_valves_2[9]),
                int_to_bool(solenoid_valves_2[8]),
                int_to_bool(solenoid_valves_2[7]),
                int_to_bool(solenoid_valves_2[6]),
                int_to_bool(solenoid_valves_2[5]),
                int_to_bool(solenoid_valves_2[4]),
                int_to_bool(solenoid_valves_2[3]),
                int_to_bool(solenoid_valves_2[2]),
                int_to_bool(solenoid_valves_2[1]) ) ;

(* Check for a pc_ready transition *)
IF ( pc_ready = 0 ) THEN
   wait_for_pc := 0 ;
END_IF ;

(* Check the status of the liquid nitrogen low level dewer alarm *)

IF  adc4[5] < 2000 THEN
    IF liquid_nitrogen_is_filling = 0 THEN
         trouble_lamp := 1; (* this shouldn't ever happen unless the filling failed and timed out, maybe not even then *)
    END_IF ;
    liquid_nitrogen_level_problem := 1;
    plc_ready := 0 ;
    (* stop cycling valves if we run out of LN2 in small dewer *)
    (* this avoids making false alarms due to detector warming up *)
ELSIF liquid_nitrogen_level_problem = 1 THEN
    (* Here we have transitioned from having an LN2 problem to not having a problem *)
    (* Now reset the sniffer so the spectrometer background isn't stale *)
    initialize_SNIFFER := 1 ;
    wait_for_pc := 0 ;
    liquid_nitrogen_level_problem := 0 ;
ELSE
    liquid_nitrogen_level_problem := 0 ;
END_IF ;

(* Check the status of the liquid nitrogen filling line*)
IF  adc4[6] > 300 THEN
  liquid_nitrogen_is_filling := 1 ;
  restart_OMNIC := 500 ; (* Restart Omnic to avoid memory leak. This also halts PC from taking data *)
ELSE
  IF liquid_nitrogen_is_filling = 1 THEN (* transitioning back from LN2 fill *)
    initialize_sniffer := 1 ;  (* take background after a little pause *)
  END_IF ;
  liquid_nitrogen_is_filling := 0 ;
  restart_OMNIC := 0 ;
```

> page 29

> page 28

```
  END_IF ;

END_IF ;

»»»»»»»»»»»»» Structured Text End «««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««
(* Section Display_Tester *)
(* Shifts bits in registers to test alarm display *)
(* First version: W. Brooks, Jan 17, 2002 *)

IF adc4[7] > 500 THEN (* red button is being pushed in *)
   display_test_counter := display_test_counter + 1 ; (* countdown 'timer' *)
   IF display_test_counter > 100 THEN
      display_test_counter := 0 ; (* reset timer *)
      IF display_test_switch = 1 THEN (* currently shifting display_test_1 variable *)
         (* shift bit left by 1 position *)
         display_test_out_1 := SHL_WORD( IN1:=display_test_out_1, N:=1);
         IF display_test_out_1 = 0 THEN ; (* shifted bit off end *)
            display_test_out_2 := 1 ; (* initialize this variable *)
            display_test_switch := 2 ; (* switch to shifting display_test_2 *)
         END_IF ;
      ELSE
         IF display_test_switch = 2 THEN (* see above comments *)
            (* shift bit left *)
            display_test_out_2 := SHL_WORD( IN1:=display_test_out_2, N:=1);
            IF display_test_out_2 = 0 THEN ; (* shifted bit off end *)
               display_test_out_1 := 1 ;
               display_test_switch := 1 ;
            END_IF ;
         END_IF ;
      END_IF ;
   END_IF ;
END_IF ;
»»»»»»»»»»»»» Structured Text End ««««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* Section VESDA *)

(* First version: W. Brooks, Dec 4, 1997 *)

(* Modifications:
      None so far - this section validated 11 Dec 1997 *)


(* This section checks to see if the concentrations exceed the
lower limit, the warning upper limit, or the alarm upper limit,
and sets the appropriate value for the alarm status. *)

(* In the following it is assumed that the alarm limits for the
'alarm' level are greater than those for the 'warning' level,
which are in turn greater than that of the 'lower' level. If this
is not the case, it is possible for an unreliable result to occur,
because the *order* of the following IF's determines the final level
of alarm set. Could be re-written to be absolutely deterministic
at the possible expense of readability. *)


 (* Modified to refer to adc5 and to use udint, so that the new
 high-resolution adc can be used for the VESDA. W. Brooks, Jan 02 *)

IF do_VESDA THEN      (* Toggle on or off executions of this section. *)

  FOR VESDA_location := 1 TO 4 DO  (* Look at all active locations *)

    IF word_to_udint(adc5[VESDA_location]) > ( VESDA_scale_factor * warning_limit_VESDA[VESDA_location]) THEN
      VESDA_alarm_status[VESDA_location] := 1 ;   (* warning level *)
    ELSE
      VESDA_alarm_status[VESDA_location] := 0 ;   (* no_alarm level *)
    END_IF ;

    IF word_to_udint(adc5[VESDA_location]) > ( VESDA_scale_factor * alarm_limit_VESDA[VESDA_location]) THEN
      VESDA_alarm_status[VESDA_location] := 2 ;   (* alarm level *)
    END_IF ;

  END_FOR ;


  IF adc4[8] < 3500 THEN   (* This input monitors the 5 volt power supply. *)
    trouble_lamp := 1 ;
    alarm_id := 34 ;
  END_IF ;

END_IF ;

»»»»»»»»»»»» Structured Text End ««««««««««««
```

```
»»»»»»»»»»»» Structured Text Start ««««««««««««

(* First version: Jan 2002, W. Brooks.
This section does what is needed for external vesda monitoring.
It keeps track of the maximum values of each vesda detector
for a given time period, then at the end of that time period,
'broadcasts' the maximum values to 4x registers for external monitoring.
The purpose of keeping the maximum values is so that 'spikes' can be
identified.

      The following scheme only works for time delays of
      less than 65535/100 seconds, or 10.92 minutes. *)
      test_dint := uint_to_dint(timer_register) - uint_to_dint(vesda_timer) ;
      IF test_dint < 0 THEN  (* timing register rolled over, => compensate *)
        test_dint := test_dint + 65535 ;
        time_difference_1 := dint_to_uint(test_dint) ;
      ELSE
        time_difference_1 := dint_to_uint(test_dint) ;
      END_IF ;

      IF word_to_uint(adc5[1]) > vesda1max THEN
           vesda1max :=  word_to_uint(adc5[1]) ;
      END_IF ;

      IF word_to_uint(adc5[2]) > vesda2max THEN
           vesda2max :=  word_to_uint(adc5[2]) ;
      END_IF ;

      IF word_to_uint(adc5[3]) > vesda3max THEN
           vesda3max :=  word_to_uint(adc5[3]) ;
      END_IF ;

      IF word_to_uint(adc5[4]) > vesda4max THEN
           vesda4max :=  word_to_uint(adc5[4]) ;
      END_IF ;

      (* The following number specifies the allowed time delay in milliseconds. *)
      IF time_difference_1 > 30000 THEN
         broadcast_vesda1 := ( uint_to_real(vesda1max)-768.0)/3200.0 - 10.0 ;
         broadcast_vesda2 := ( uint_to_real(vesda2max)-768.0)/3200.0 - 10.0 ;
         broadcast_vesda3 := ( uint_to_real(vesda3max)-768.0)/3200.0 - 10.0 ;
         broadcast_vesda4 := ( uint_to_real(vesda4max)-768.0)/3200.0 - 10.0 ;
         vesda1max := 0 ;
         vesda2max := 0 ;
         vesda3max := 0 ;
         vesda4max := 0 ;
      END_IF ;

»»»»»»»»»»»»» Structured Text End ««««««««««««»
```