F20RS Coursework 1 Report

A SPARK High Integrity Software Development Exercise

Heriot-Watt University

October 25th 2021

By Drew Tomkins (H00294338)

Table of Contents

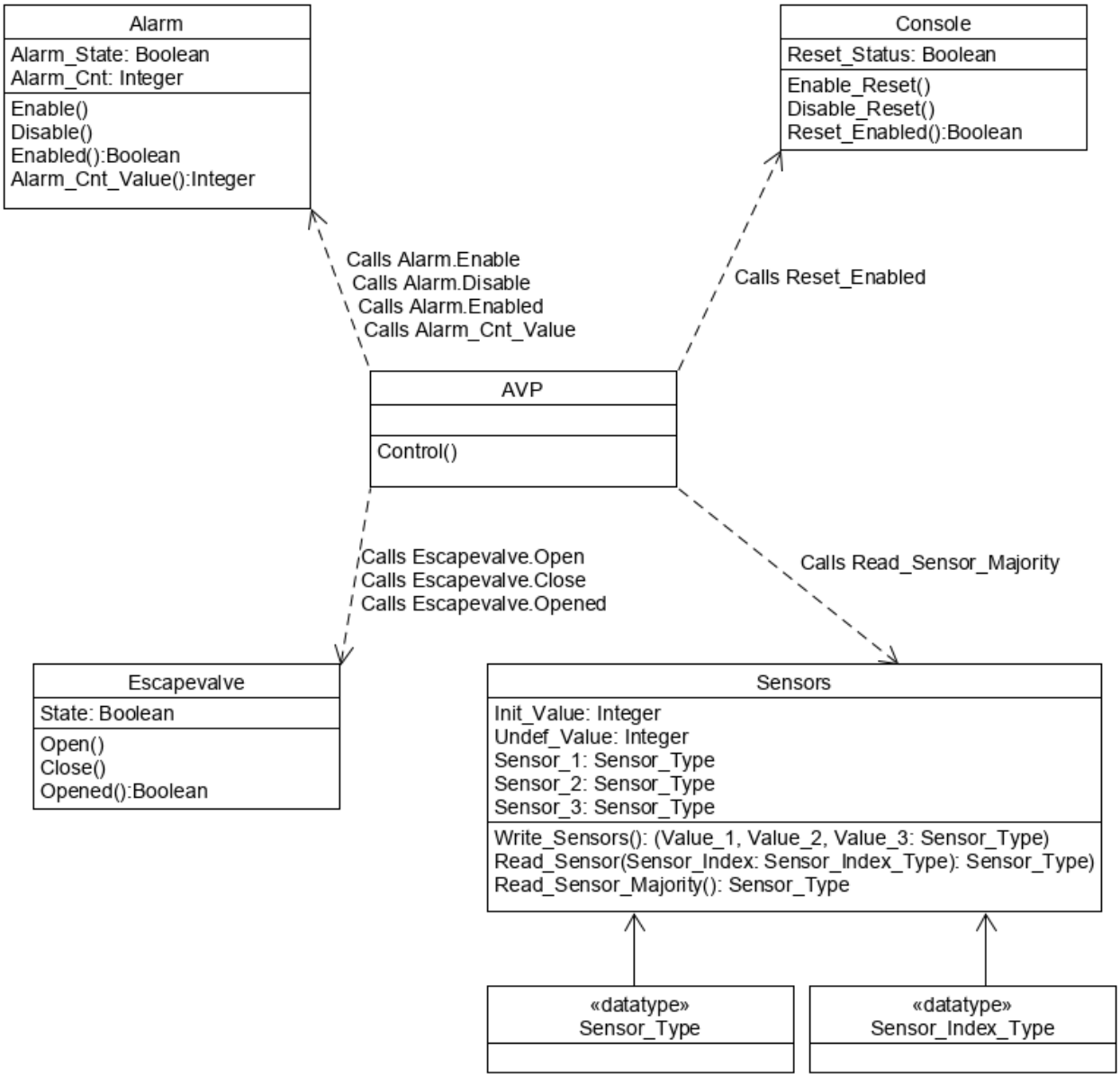
1. Introduction

This project involved creating an Automatic Vessel Protection (AVP) system using the SPARK programming language that prevents explosions on vessels by handling high-pressure hazards in the process. This report contains information regarding the project, such as assumptions made, a diagrammatic representation of the AVP system, listings of the package specification and body source files used to create the system, log files showing the output of the system and how run time exceptions were prevented.

2. Assumptions

- Within the specification, it states that only 3 sensors are used to generate pressure readings, therefore I have decided to use an array of 3 variables representing the types of sensors to make implementation easier on my end.
- When implementing proof contracts for the package specifications, I assumed that from the beginning everything would be functional and constantly active in the system, therefore I made all of my preconditions for the package specifications true to constantly check to see if each procedure was running as intended.
- With the specification stating that the reset mechanism should disable the escape valve and alarm upon normal pressure levels being reached, I assumed that the system could only try and close the valve with the reset mechanism if the valve was open in the first place since it wouldn't make sense to try and reset the escape valve if it was already closed.

3. Diagrammatic Representation of Software Architecture


Alarm
Alarm_State: Boolean
Alarm_Cnt: Integer
Enable()
Disable()
Enabled():Boolean
Alarm_Cnt_Value():Integer
Console
Reset_Status: Boolean
Enable_Reset()
Disable_Reset()
Reset_Enabled():Boolean
Calls Alarm.Enable
Calls Alarm.Disable
Calls Alarm.Enabled
Calls Alarm_Cnt_Value
Calls Reset_Enabled
AVP
Control()
Calls Escapevalve.Open
Calls Escapevalve.Close
Calls Escapevalve.Opened
Calls Read_Sensor_Majority
Escapevalve
State: Boolean
Open()
Close()
Opened():Boolean
Sensors
Init_Value: Integer
Undef_Value: Integer
Sensor_1: Sensor_Type
Sensor_2: Sensor_Type
Sensor_3: Sensor_Type
Write_Sensors(): (Value_1, Value_2, Value_3: Sensor_Type)
Read_Sensor(Sensor_Index: Sensor_Index_Type): Sensor_Type)
Read_Sensor_Majority(): Sensor_Type
«datatype»
Sensor_Type
«datatype»
Sensor_Index_Type

4. Listing of Source Files

4.1. Alarm

4.1.1. Alarm Specification

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       24.8.2021
--
-- Filename:            alarm.ads
--
-- Description:         Models the alarm device associated with
--                      the AVP controller and the alarm count.

pragma SPARK_Mode (On);
package Alarm

is
   pragma Elaborate_Body;

   Alarm_State: Boolean;
   Alarm_Cnt: Integer:= 0;

   procedure Enable
   with
     Global  => (Output => Alarm_State,
         In_Out => Alarm_Cnt),
     Depends => (Alarm_State  => null,
         Alarm_Cnt => Alarm_Cnt),
     -- Verifies the result by ensuring the result of the Read_Sensor function is less than or equal to the
     -- last integer value provided as well as checking to see if the current state is true
     Pre => (True and Alarm_Cnt <= Integer'Last),
     -- Checks to see if the alarm state is active after the procedure
     Post => (Alarm_State);

   procedure Disable
   with
   Global  => (Output => Alarm_State),
     Depends => (Alarm_State  => null),
     -- Checks to see if the provided input is true
     Pre => True,
     -- Checks to see if the alarm state is inactive after the procedure
     Post => (not(Alarm_State));

   function Enabled return Boolean
   with
   Global => (Input => Alarm_State),
     Depends => (Enabled'Result => Alarm_State),
     -- Checks to see if the provided input is true
     Pre => True,
     -- Verifies the result by ensuring the result of the Enabled function is a value representing the alarm state
     Post => (Enabled'Result = Alarm_State);

   function Alarm_Cnt_Value return Integer
   with
   Global => (Input => Alarm_Cnt),
     Depends => (Alarm_Cnt_Value'Result => Alarm_Cnt),
     -- Checks to see if the current alarm count is 0
     Pre => (Alarm_Cnt = 0),
     -- Checks to see if the current alarm count is over or equal to 0
     Post => (Alarm_Cnt_Value'Result >= 0);

end Alarm;
```

4.1.2. Alarm Body

```
-- Author:              Drew Tomkins
--
-- Last modified:       23.10.2021
--
-- Filename:            alarm.adb
--
-- Description:         Provides the package body for the Alarm package.
--                      Models the alarm device associated with
--                      the AVP controller and the alarm count.

pragma SPARK_Mode (On);
package body Alarm
is
   -- Sets off the alarm
   procedure Enable
   is
   begin
      -- Sets the alarm state to true and triggers the alarm
      Alarm_State:= True;
      -- Adds 1 to the alarm count
      Alarm_Cnt:= Alarm_Cnt + 1;

   end Enable;

   -- Turns the alarm off
   procedure Disable
   is
   begin
      -- Sets the alarm state to false and turns the alarm off
      Alarm_State:= False;
   end Disable;

   -- Checks the current alarm state to see if the alarm is currently active
   function Enabled return Boolean
   is
   begin

      -- Return the current state of the alarm
      return Alarm_State;

   end Enabled;

   -- Gets the current alarm count
   function Alarm_Cnt_Value return Integer
   is
   begin

      -- Returns the current count of alarms at that point
      return Alarm_Cnt;

   end Alarm_Cnt_Value;
```

```
begin

   -- Sets the alarm state to false so the alarm begins as off
   Alarm_State := false;
   -- Sets the alarm count to 0
   Alarm_Cnt := 0;

end Alarm;
```

4.2. AVP

4.2.1. AVP Specification

```
-- Author:              Drew Tomkins
--
-- Last modified:       24.10.2021
--
-- Filename:            avp.adb
--
-- Description:         Provides the package specification for the AVP package.
--                      This models the overall control for the AVP system, which
--                      serves as the central system that handles the rest of the
--                      subsystems for the overall system.

pragma SPARK_Mode (On);
-- Inherit the subsystems packages to be used in the AVP package
with Alarm, Escapevalve, Sensors, Console;
package AVP

is
   pragma Elaborate_Body;

   procedure Control
   with
     -- Takes in the current state from the Sensors package
     -- and the current reset status from the Console package
     Global => (Input => (Sensors.State, Console.Reset_Status),
     -- Takes the current alarm state from the Alarm package, the
     -- current status of the escape valve from the EscapeValve package
     -- and the current alarm count from the Alarm package to be modified
     -- as necessary
     In_Out => (Alarm.Alarm_State, Escapevalve.State, Alarm.Alarm_Cnt)),
     -- Checks to see if  the alarm count in the Alarm class is over or equal to 0
     Pre => Alarm.Alarm_Cnt >= 0,
     -- This post-condition checks to see if the end result is true
     Post => True;
end AVP;
```

4.2.2. AVP Body

```
-- Author:              Drew Tomkins
--
-- Last modified:       24.10.2021
--
-- Filename:            avp.adb
--
-- Description:         Provides the package body for the AVP package.
--                      This models the overall control for the AVP system, which
--                      serves as the central system that handles the rest of the
--                      subsystems for the overall system.

pragma SPARK_Mode (On);
with Alarm, Escapevalve, Sensors, Console;
package body AVP

is
   -- Serves as the controller for the entire AVP system and uses the
   -- 4 subsystems accordingly to control it
   procedure Control is
      -- Declares a variable for the sensor value by giving it the Sensor_Type subtype
      -- from the Sensors package
      Sensor_Value: Sensors.Sensor_Type;
   begin

      -- Assigns the reading from the Read_Sensor_Majority function in
      -- the Sensors package to the Sensor_Value variable
      Sensor_Value := Sensors.Read_Sensor_Majority;

      -- Checks to see if the escape valve isn't opened
      if Escapevalve.Opened = false then

         -- Checks to see if the sensor reading was below 100
         if (Sensor_Value < 100) then

            -- Checks to see if the alarm is currently active
            if Alarm.Enabled = True then
               -- Shuts off the alarm if it is active
               Alarm.Disable;
            end if;

            -- Checks to see if the reading is equal to or above 100
         else if (Sensor_Value >= 100) then

               -- If the value is above 100, then this checks to see if
               -- the reading is below or equal to 149
               if (Sensor_Value <= 149) then
                  -- Checks to see if the alarm is disabled
                  if Alarm.Enabled = False then
                     -- Sets off the alarm if it isn't already on
                     Alarm.Enable;
                  else
```

```
                        -- If the value doesn't decrease on the next reading
                        -- then open the escape valve
                        Escapevalve.Open;
                     end if;

                     -- Checks to see if the reading is equal to or over 150
                  else if (Sensor_Value >= 150) then
                        -- Open the escape valve if the above is true
                        Escapevalve.Open;
                        -- Checks to see if the alarm isn't on
                        if Alarm.Enabled = False then
                           -- Sets off the alarm if it isn't already on
                           Alarm.Enable;
                        end if;
                     end if;
                  end if;
               end if;

               -- Checks to see if the reset mechanism is enabled from the console
               if Console.Reset_Enabled = true then
                  -- Reset mechanism turns off the alarm...
                  Alarm.Disable;
                  -- and closes the escape valve
                  Escapevalve.Close;
               end if;
            end if;
         end if;

   end Control;
end AVP;
```

4.3. Console

4.3.1. Console Specification

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       24.8.2021
--
-- Filename:            console.ads
--
-- Description:         Models the console associated with the AVP system, i.e.
--                      the reset mechanism that is required to close the
--                      emergency escape valve.


pragma SPARK_Mode (On);
package  Console

is
   pragma Elaborate_Body;

   Reset_Status: Boolean:= False;

   procedure Enable_Reset
   with
   Global => (Output => Reset_Status),
     Depends => (Reset_Status => null),
     --Checks to see if the provided input is true
     Pre => True,
     -- Checks to see if the reset is enabled after the procedure
     Post => (Reset_Status);

   procedure Disable_Reset
   with
   Global => (Output => Reset_Status),
     Depends => (Reset_Status => null),
     --Checks to see if the provided input is true
     Pre => True,
     -- Checks to see if the reset isn't enabled after the procedure
     Post => (not(Reset_Status));

   function Reset_Enabled return Boolean
   with
   Global => (Input => Reset_Status),
     Depends => (Reset_Enabled'Result => Reset_Status),
     --Checks to see if the provided input is true
     Pre => True,
     -- Verifies the result by ensuring the result of the Reset_Enabled function is a value representing the current reset status
     Post => (Reset_Enabled'Result = Reset_Status);

end Console;
```

4.3.2. Console Body

```
-- Author:              Drew Tomkins
--
-- Last modified:       23.10.2021
--
-- Filename:            console.adb
--
-- Description:         Provides the package body for the Console package.
--                      Models the console associated with the AVP system, i.e.
--                      the reset mechanism that is required to close the
--                      emergency escape valve.


pragma SPARK_Mode (On);
package body Console

is

   -- Enables the reset mechanism
   procedure Enable_Reset
   is
   begin

      -- Sets the reset status to true and enables the reset mechanism
      Reset_Status:= true;


   end Enable_Reset;

   -- Disables the reset mechanism
   procedure Disable_Reset
   is
   begin

      -- Sets the reset status to false and disables the reset mechanism
      Reset_Status:= false;


   end Disable_Reset;

   -- Checks to see if the reset mechanism is enabled
   function Reset_Enabled return Boolean
   is
   begin

      -- Gets the current status of the reset mechanism and returns it
      return Reset_Status;


   end Reset_Enabled;


end Console;
```

4.4. Escapevalve

4.4.1. Escapevalve Specification

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       23.8.2021
--
-- Filename:            escapevalve.ads
--
-- Description:         Models the emergency escape valve associated with the
--                      pressure vessel.

pragma SPARK_Mode (On);
package Escapevalve

is
   pragma Elaborate_Body;

   State: Boolean:= false;

   procedure Open
   with
   Global => (Output => State),
     Depends => (State => null),
     --Checks to see if the provided input is true
     Pre => True,
     -- Checks to see if the escape valve is open by seeing if the state is true
     Post => (State);

   procedure Close
   with
   Global => (Output => State),
     Depends => (State => null),
     --Checks to see if the provided input is true
     Pre => True,
     -- Checks to see if the escape valve is closed by seeing if the state is false
     Post => (not(State));

   function Opened return Boolean
   with
   Global => (Input => State),
     Depends => (Opened'Result => State),
     --Checks to see if the provided input is true
   Pre => True,
     -- Verifies the result by ensuring the result of the Opened function is a value representing the current state of the escape valve
   Post => (Opened'Result = State);

end Escapevalve;
```

4.4.2. Escapevalve Body

```
-- Author:              Drew Tomkins
--
-- Last modified:       23.10.2021
--
-- Filename:            escapevalve.adb
--
-- Description:         Provides the package body for the Escapevalve package.
--                      Models the emergency escape valve associated with the
--                      pressure vessel.

pragma SPARK_Mode (On);
package body Escapevalve

is

   -- Opens the escape valve
   procedure Open
   is
   begin

      -- Sets the state to true and opens the escape valve
      State:= true;

   end Open;

   -- Closes the escape valve
   procedure Close
   is
   begin

      -- Sets the state to true and closes the escape valve
      State:= false;

   end Close;


   -- Checks to see if the escape valve is opened
   function Opened return Boolean
   is
   begin

      -- Gets the current status of the escape valve and returns it
      Return State;

   end Opened;

end Escapevalve;
```

4.5. Sensors

4.5.1. Sensors Specification

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       23.9.2021
--
-- Filename:            sensors.ads
--
-- Description:         Models the 3 pressure sensors associated with the AVP system. Note that
--                      a single sensor reading is calculated using a majority vote
--                      algorithm.

pragma SPARK_Mode (On);
package Sensors

is

   pragma Elaborate_Body;

   Init_Value:  constant Integer := 0;
   Undef_Value: constant Integer := 200; -- range of valid pressure readings is 0..199. A value of 200 denotes an undefined reading.

   subtype Sensor_Type is Integer range 0..200;
   subtype Sensor_Index_Type is Integer range 1..3;
   type Sensors_Type is array (Sensor_Index_Type) of Sensor_Type;

   State: Sensors_Type;

   procedure Write_Sensors(Value_1, Value_2, Value_3: in Sensor_Type)
     with
       Global => (In_Out => State),
       Depends => (State => (State, Value_1, Value_2, Value_3)),
       --  Both pre and post-conditions check to see if there is a state value present
       Pre => True,
       Post => True;

   function Read_Sensor(Sensor_Index: in Sensor_Index_Type) return Sensor_Type
     with
       Global  => (Input => State),
       Depends => (Read_Sensor'Result => (State, Sensor_Index)),
     -- Checks to see if the provided input is true
       Pre => True,
     -- Verifies the result by ensuring the result of the Read_Sensor function is less than or equal to the last integer value provided
       Post => (Read_Sensor'Result <= Integer'Last);

   function Read_Sensor_Majority return Sensor_Type
     with
       Global => (Input => State),
       Depends => (Read_Sensor_Majority'Result => State),
     -- Checks to see if the provided input is true
       Pre => True,
     -- Verifies the result by ensuring the result of the Read_Sensor_Majority function is less than or equal to the last integer value provided
       Post => (Read_Sensor_Majority'Result <= Integer'Last);

end Sensors;
```

4.5.2. Sensors Body

```
-- Author:              Drew Tomkins

-- Last modified:       23.10.2021
--
-- Filename:            sensors.adb
--
-- Description:         Package body for the Sensors package.
--                      Models the 3 pressure sensors associated with the AVP system. Note that
--                      a single sensor reading is calculated using a majority vote
--                      algorithm.

pragma SPARK_Mode (On);
package body Sensors

is

   -- Writes values to each of the 3 sensors
   procedure Write_Sensors(Value_1, Value_2, Value_3: in Sensor_Type)
   is
   begin
      -- For each sensor in the array, a value is assigned to it
      State(1) := Value_1;
      State(2) := Value_2;
      State(3) := Value_3;

   end Write_Sensors;

   -- Reads the sensors that are taken in from the Sensor_Index_Type subtype and returns the sensor in use
   function Read_Sensor(Sensor_Index: in Sensor_Index_Type) return Sensor_Type
   is
      -- Declare the 'sensortype' variable by giving it the Sensor_Type subtype and assigning an initial value to it
      Sensor: Sensor_Type := Init_Value;

   begin

      -- Checks to see if the current value of the sensor index is below the value of the last sensor checked
      if Sensor_Index < Sensor_Index_Type'Last then
         -- If the value is 1 then the sensor is set to the first sensor
         if Sensor_Index = 1 then
            Sensor := State(1);
            -- If the value is 2 then the sensor is set to the second sensor
         elsif Sensor_Index = 2 then
            Sensor:= State(2);
            -- Otherwise the sensor is set to the third sensor
         else
            Sensor:= State(3);
         end if;
      end if;
      -- Return the sensor that was read
      return Sensor;
   end Read_Sensor;
```

```
   -- Get the majority value of the 3 sensor readings
   function Read_Sensor_Majority return Sensor_Type
   is
      -- Declare the sensortype variable by giving if the Sensor_Type subtype
      sensor_reading: Sensor_Type;
   begin
      -- If Sensors 1 and 2 are equal then the reading to be returned will be that of the first sensor
      if State(1) = State(2) then
         Sensor_Reading:= State(1);
         -- If Sensors 1 and 3 are equal then the reading to be returned will be that of the second sensor
      elsif State(1) = State(3) then
         Sensor_Reading:= State(2);
         -- If Sensors 2 and 2 are equal then the reading to be returned will be that of the third sensor
      elsif State(2) = State(3) then
         Sensor_Reading:= State(3);
         -- If there is no majoirty then the reading to be returned will be undefined
      else
         Sensor_Reading:= Undef_Value;
      end if;
      -- Return the majority reading from the 3 sensors
      return Sensor_Reading;
   end Read_Sensor_Majority;

begin
   -- Initialise each sensor in the array with an initial value
   State(1) := Init_Value;
   State(2) := Init_Value;
   State(3) := Init_Value;

end Sensors;
```

5. Summary File

This provides the GNATprove.out summary file for flow analysis and exception freedom purposes.

```
Summary of SPARK analysis
=========================

-----------------------------------------------------------------------------------------------------
SPARK Analysis results        Total        Flow   CodePeer                      Provers   Justified   Unproved
-----------------------------------------------------------------------------------------------------
Data Dependencies                14          14          .                            .           .          .
Flow Dependencies                13          13          .                            .           .          .
Initialization                    9           8          .                            .           .          1
Non-Aliasing                      .           .          .                            .           .          .
Run-time Checks                   1           .          .                            .           .          1
Assertions                        .           .          .                            .           .          .
Functional Contracts             27           .          .    27 (CVC4 53%, Trivial 47%)          .          .
LSP Verification                  .           .          .                            .           .          .
Termination                       .           .          .                            .           .          .
Concurrency                       .           .          .                            .           .          .
-----------------------------------------------------------------------------------------------------
Total                            64    35 (55%)          .                   27 (42%)           .     2 (3%)


max steps used for successful proof: 1

Analyzed 5 units
in unit alarm, 5 subprograms and packages out of 5 analyzed
  Alarm at alarm.ads:17 flow analyzed (0 errors, 0 checks and 1 warnings) and proved (0 checks)
  Alarm.Alarm_Cnt_Value at alarm.ads:54 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Alarm.Disable at alarm.ads:36 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Alarm.Enable at alarm.ads:25 flow analyzed (0 errors, 0 checks and 0 warnings) and not proved, 1 checks out of 2 proved
  Alarm.Enabled at alarm.ads:45 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
in unit avp, 2 subprograms and packages out of 2 analyzed
  AVP at avp.ads:15 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (0 checks)
  AVP.Control at avp.ads:20 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (14 checks)
in unit console, 4 subprograms and packages out of 4 analyzed
  Console at console.ads:19 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (0 checks)
  Console.Disable_Reset at console.ads:35 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Console.Enable_Reset at console.ads:26 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Console.Reset_Enabled at console.ads:44 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
in unit escapevalve, 4 subprograms and packages out of 4 analyzed
  Escapevalve at escapevalve.ads:17 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (0 checks)
  Escapevalve.Close at escapevalve.ads:33 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Escapevalve.Open at escapevalve.ads:24 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Escapevalve.Opened at escapevalve.ads:42 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
in unit sensors, 4 subprograms and packages out of 4 analyzed
  Sensors at sensors.ads:18 flow analyzed (0 errors, 1 checks and 0 warnings) and proved (0 checks)
  Sensors.Read_Sensor at sensors.ads:41 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Sensors.Read_Sensor_Majority at sensors.ads:50 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
  Sensors.Write_Sensors at sensors.ads:33 flow analyzed (0 errors, 0 checks and 0 warnings) and proved (1 checks)
```

6. Log Files

6.1. log.dat - Appendix A.1 env.dat

SENSOR-1	SENSOR-2	SENSOR-3	MAJORITY	ALARM	E-VALVE	RESET	ALARM_CNT
--------	--------	--------	--------	-----	-------	-----	---------
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
HIGH	HIGH	NORMAL	HIGH	--	--	--	0
HIGH	HIGH	NORMAL	HIGH	ON	--	--	1
HIGH	HIGH	NORMAL	HIGH	ON	--	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	HIGH	NORMAL	UNDEF	ON	OPEN	--	1
NORMAL	HIGH	NORMAL	UNDEF	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	ON	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	ON	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	--	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	ON	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	ON	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	--	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	ON	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	ON	1

6.2. log.dat - Assignment env.dat

SENSOR-1	SENSOR-2	SENSOR-3	MAJORITY	ALARM	E-VALVE	RESET	ALARM_CNT
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	--	--	0
UNDEF	NORMAL	NORMAL	NORMAL	--	--	--	0
UNDEF	NORMAL	NORMAL	NORMAL	--	--	--	0
NORMAL	UNDEF	NORMAL	UNDEF	--	--	--	0
NORMAL	UNDEF	NORMAL	UNDEF	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
UNDEF	UNDEF	NORMAL	UNDEF	ON	OPEN	--	1
UNDEF	UNDEF	NORMAL	UNDEF	ON	OPEN	--	1
UNDEF	UNDEF	NORMAL	UNDEF	ON	OPEN	ON	1
UNDEF	UNDEF	NORMAL	UNDEF	ON	OPEN	ON	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	--	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	--	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	ON	1
CRITICAL	CRITICAL	NORMAL	CRITICAL	ON	OPEN	ON	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
HIGH	HIGH	NORMAL	HIGH	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	--	1

7. Protection Against Runtime Exceptions

To protect against runtime exceptions in the code, I implemented preconditions and postconditions into each package specification to check for any potential errors when executing the code. Details regarding the purpose of these conditions and how they protected against runtime exceptions are as follows:

Alarm

Enable

Within this procedure, I aimed to prevent an alarm state not being present as well as ensuring the alarm count wasn't exceeding the integer values which would cause overflow exceptions, so I created preconditions to check if the alarm state was true and that the alarm count was below the last integer value to prevent overflow exceptions from happening. I also added a postcondition for the alarm state to ensure that an alarm state was present at the end of the procedure.

Disable

This is similar to the Enable procedure in terms of exceptions to be prevented and pre/postconditions however the alarm count is not checked here, and the postcondition checks to see if the alarm state was inactive at the end of the procedure.

Enabled

To prevent a runtime exception because of a non-present alarm state, the precondition is the same as the Disable precondition, however I made the postcondition verify that the result of this function was an alarm state so that when checking to see if the alarm state is enabled, an alarm state will have to be present for the function to be proof checked by GNATprove successfully.

Alarm_Cnt_Value

To prevent potential exceptions here with the alarm count thanks to negative numbers, I made a precondition to check if the alarm count was 0 so that the alarm count would have a proper initial value, and set the postcondition up so that the alarm

count would be checked to see if it was 0 or above, ensuring exceptions because of a negative alarm count are prevented.

AVP

Control

Here, I wanted to ensure that certain subsystems wouldn't throw exceptions so I added a precondition relating to the alarm count from the Alarm package, ensuring the value wasn't a negative throughout the execution of the AVP program, and the postcondition I added checked to see if the end result was true so that there was actually a result instead of no result, which could lead to a runtime exception.

Console

Enable_Reset

For this procedure I wanted to prevent exceptions that would occur if an initial result wasn't found which would prevent the rest of the program from executing, so I added a precondition to check if there was an actual input provided beforehand so the procedure had a result to work with. I also added a postcondition to check if the reset status was enabled after the procedure had executed so that the other procedures and functions in the Control package could function with the result instead of throwing an exception.

Disable_Reset

Similar to Enable_Reset, except for the postcondition I checked to see if the reset status was disabled instead of enabled.

Reset_Enabled

Similar to the 2 above procedures, except since this was a function I wanted to ensure that the result I was getting during execution was the reset status and not anything else which could cause a runtime exception as a result of the wrong result, so I used the postcondition to verify that the result of this function was indeed the current reset status to prevent this from happening.

Escapevalve

Open

I wanted to check if the input provided was true so that exceptions caused by a bad check due to an unexpected result would be avoided, so I added a precondition to accomplish that and I implemented a postcondition to check if the state was true, which would ensure the escape valve was open and the procedure had fulfilled its goal with the correct output.

Close

Similar to Open, except for the postcondition I checked to see if the escape valve was closed instead of opened to ensure the result was correct and runtime exceptions would be prevented as a result of this.

Opened

Similar to the 2 above procedures, except since this was a function I wanted to ensure that the result I was getting during execution was the current status of the escape valve and not anything else which could cause a runtime exception as a result of the wrong state being provided in the program execution, so I used the postcondition to verify that the result of this function was indeed the current state of the escape valve to ensure the exceptions wouldn't be able to crop up here.

Sensors

Write_Sensors

With the array of sensors, I wanted to check if the range of the array's contents were suitable and had the expected 3 sensors with written values in them, so I had both the pre and postconditions check for a true result to ensure that each sensor had a value written to it in order for the readings to be correctly processed in the rest of the package, preventing the aforementioned runtime exception in the process.

Read_Sensors

With this function checking the sensor readings to determine if the current value of the sensor index was below the value of the last sensor reading, I wanted to prevent possible exceptions that could be caused by the current reading value being unexpectedly higher than the most recent reading provided or the value being negative, so I had a precondition to check to see if there was a legitimate result

beforehand, then my postcondition here checked to see if the result of this function was below the last provided reading value, therefore preventing the above exceptions since the result is expected.

Read_Sensor_Majority

This is similar to the Read_Sensors function in regard to what exceptions I want to prevent and the conditions I implemented to prevent the exceptions, though with the difference being that this function is checking for the majority reading from the 3 sensors instead of simply getting readings from them.