Rigorous Methods for Software Engineering

Coursework 1

A High Integrity Software Development Exercise

F21RS1

Boris Mocialov
(H00180016)

Heriot – Watt University, Edinburgh
October 2014

Date: 24.10.14

Contents

Tables

1. Introduction

This report discusses the coursework 1 for the module stated in the title. The purpose of this assignment is to implement and verify implemented software component of a simple ATP system using SPARK approach to high integrity Ada. ATP stands for Automatic Train Protection, which is used to ensure safe passage by monitoring a train's speed on the approach to track-side signals and to activate brakes automatically if it is necessary based on sensors values fed into ATP system.

Implementation extended provided SPARK package specifications that define the safety-critical control component of the system. In addition, ATP package, consistent with the given package specifications and test harness, was developed. ATP package implements the intended behaviours of the ATP controller. The resulting implementations define the safety-critical boundary of ATP system.

Verification preceded using SPARK proof tools, namely, SPARK Examiner and SPARK Simplifier, while summaries of entire proofs were produced with the help of POGS using outputs from SPARK proof tools.

2. Requirements

Requirements consist of implementation of package bodies for system-critical control component and an additional unit ATP which implements behaviour of the ATP controller. Below is the list of identified requirements.

Package	Purpose	Task
Sensors	Maintains and provides access to sensor values	Develop package bodies consistent with specifications provided
Speedo	Maintains and provides access to speedometer values	
Brakes	Provides control and maintains state of train brakes	
Alarm	Provides control and maintains state of alarm	
Console	Provides interface to ATP controller Provides control and maintains state of reset subsystem Maintains count of SPAD events[1]	
ATP	Overall control of ATP system	Develop specification and body consistent with first 5 packages and test harness

[1] SPAD - 'Signal Passed at Danger' event

ALL	Support confidence in developed system	Prove run-time exception freedom using SPARK proof tools
Sensors	Be confident that the function returns expected result	Use *return* annotation to specify correctness of the *Read_Sensor_Majority* function
ALL	Verify that the implementation is consistent with the specification	Using SPARK proof tools verify that package bodies satisfy specifications
Requirements identified: 9		

Table 1: Requirements

2.1. Assumptions

Due to the nature of identified requirements, some assumptions were made that led to particular decisions in the design of the system.

Assumption	Decision
Assuming that undefined signal is of lower level than caution signal	Alarm is enabled when undefined signal is detected by the ATP controller
Assuming that train stops or slows down considerably after brakes are engaged	After brakes are engaged, system reset is triggered
Assuming that console is a responsive unit and does not act upon the rest of the system	Console is implemented as a responsive unit, which does not affect the entire system directly
Assuming there are precisely 3 sensors	Data is deduced based on values gathered from 3 sensors. Requirements talk about sensor no. 4 only once and never mentions it again, therefore this information is not taken into account
Assuming that when proceed signal is detected, alarm should be disabled if brakes are not engaged	Alarm is disabled irrespective of the previous state
Assuming that SPAD counter must only be incremented once the system is started and such an increment must happen only when Danger signal is detected	SPAD counter is only incremented when danger signal is detected. Deduced based on the abbreviation: “Signal Passed At Danger”
Assuming that the SPAD counter must	SPAD counter is set to be between 0 and

be a positive value from 0 to 2^{31} -1	2^{31} -1
ATP is the only unit that can directly influence the rest of the system	ATP unit sends commands to the rest of the system (active unit)
Assuming that the whole system, together with the ATP unit, is responsive and does not act upon its state initiatively	Every subsystem, except for ATP, simply holds a state that corresponds to actual physical device of a train. ATP is a logical construct that performs changes to the rest of the system through '*Control'* subprogram. ATP is responsive as well.
Assuming that when system starts, sensors values are unknown	allocated objects for sensors data are initialised to 'undefined' sensors signal values. This may lead to complications, such as that alarm might possibly go off when system starts. Better option could be to set initial values to 'Proceed' signal values
Assumption is made that the system never terminates	System is passive and *ATP.Control* procedure awaits to be triggered
Assuming train stops after brakes are activated	After system activates brakes, reset is triggered
Assuming 'subsequent' means the next signal after	Previous signals are not remembered. On the next run, if proceed signal is returned, control procedure disables alarm if it was previously enabled
Assumptions made: 13	

Table 2: Assumptions

3. Architecture

3.1. Unit diagram

Unit diagram is given in Appendix A.

Boxes Sensors, Speedo, Brakes, Alarm, Console and ATP correspond to units in the system.

Every unit contains objects (variables), procedures/functions (specified) and corresponding function return types.

Dashed arrows correspond to use of particular use of sub-types of a unit.

Bigger boxes symbolise grouping of units and sub-types that lie within same unit file.

Text over arrows would help reader to identify the connection between various boxes.

3.2. State diagram

State diagram is given in Appendix B.

Given diagram describes changes in states that occur within the system. Diagram has two parts: 1. Initialisation stage and 2. Responses stage. Initialisation stage is very short – it sets SPAD counter to 0 and initialises sensors array elements to undefined signal. Responses stage specifies ATPs behaviour every time it is triggered and which states it might set the system to. Response stage is triggered when control procedure is called, represented by circle and envelope (taken from BPMN[2])

[2] BPMN – Business Process Modelling Notation

4. Testing & formal proof

4.1. Listing file

Given in Appendix C

4.2. Report file

Given in Appendix D

4.3. Log

Given in Appendix E

4.4. Pogs

Given in Appendix F (gives full overview over *.sum* files. *Read_Sensor_Majority* function is highlighted to make it easier for the reader to identify it)

To show that the code is free from run-time exceptions, proofs are constructed, which therefore proves that the system will never raise a run-time error. In SPARK, VCs are generated by applying SPARK Examiner to the existing code (i.e. running `spark -vcg @atp.smf`). SPARK Examiner discharges most trivial VCs, while SPARK Simplifier is applied later in the process to discharge most of VCs and prove exception freedom.

After SPARK Simplifier was applied using `sparksimp` on every directory generated by SPARK Examiner (alarm, atp, brakes, console, sensors, speedo), following commands were issued to extract information from *.sum* files, generated by `pogs` to find how many VCs got discharged and how many got proved. The following result was obtained:

```
$ grep -Hrn "fully proved" *
alarm/alarm.sum:114:Total subprograms fully proved:                          3
atp/atp.sum:100:Total subprograms fully proved:                              1
brakes/brakes.sum:114:Total subprograms fully proved:                        3
console/console.sum:179:Total subprograms fully proved:                      7
sensors/sensors.sum:140:Total subprograms fully proved:                      3
speedo/speedo.sum:100:Total subprograms fully proved:                        2


$ grep -Hrn "undischarged" *
alarm/alarm.sum:115:Total subprograms with at least one undischarged VC:     0
atp/atp.sum:101:Total subprograms with at least one undischarged VC:         0
brakes/brakes.sum:115:Total subprograms with at least one undischarged VC:   0
console/console.sum:180:Total subprograms with at least one undischarged VC: 0
sensors/sensors.sum:141:Total subprograms with at least one undischarged VC: 0
speedo/speedo.sum:101:Total subprograms with at least one undischarged VC:   0
```

Figure 1 Proof

5. Comparing SPARK with Java

Some of the core differences between Java and SPARK programming languages include the ability of Java to do GUI programming, while SPARK is incapable of such a feature and its purpose carries no such need. Another main difference is that Java uses JavaVM, which interprets the compiled Java byte code when program is executing, while core SPARK language is compiled and then executed and SPARK proof context code (contract) is processed once by SPARK Tools to produce reports about the intended behavior of the written Ada code and to produce proof of absence of run-time exceptions. SPARK code is thus interpreted only once and not runtime.

Presented here comparison will go into details of the two specified language features and give certain code examples to support the statements made in this section.

Some of the common language features that are used in most of the commercially available programming languages include: (comparisons are done using syntax of core SPARK language and personal knowledge of Java)

Feature	Java	SPARK

Flow controls (i.e. if … else, break, continue, try, catch, while, for, etc.)	if (AStack.Data(1) != AStack.Data(2)
	&& AStack.Data(1) != AStack.Data(3)
	&& AStack.Data(2) != AStack.Data(3))
	then
	AResult = Undef;
	else
	if (AStack.Data(1) == AStack.Data(2)) then
	AResult:= AStack.Data(1);
	else if (AStack.Data(1) == AStack.Data(3)) then
	AResult = AStack.Data(1);
	else if (AStack.Data(2) == AStack.Data(3)) then
	AResult = AStack.Data(2);
	
	break; / return; / continue;
	
	try{
	…
	}catch(<stmt>)
	{
	…
	}
	
	while(<stmt>)
	{
	…
	}
	
	for(int i=0;i<<const>;i++)
	{
	…
	}
		if AStack.Data(1) /= AStack.Data(2)
		AND AStack.Data(1) /= AStack.Data(3)
		AND AStack.Data(2) /= AStack.Data(3)
		then
		AResult := Undef;
		else
		if AStack.Data(1) = AStack.Data(2) then
		AResult := AStack.Data(1);
		elsif AStack.Data(1) = AStack.Data(3) then
		AResult := AStack.Data(1);
		elsif AStack.Data(2) = AStack.Data(3) then
		AResult := AStack.Data(2);
		end if;
		end if;
		
		exit;
		
		Proof is used instead of try … catch flow control
		
		loop
		…
		end loop;
		
		for I in <range> loop
		…
		exit when <stmt>
		end loop;

Headers (.h, .ads, uses, include, etc.)	Import is used to import external classes to be able to reference them. No header file exists. Information hiding is done via private/public access modifiers Example: import java.rmi.*;	.ads file identifies specification of a package .adb file identifies body of a package Units are imported/included into current package via inherit/with keywords. Once unit is included, its objects (defined in .ads file) are accessible Example: with Sensors, Alarm, Brakes, Speedo, Console; --# inherit Sensors, Alarm, Brakes, Speedo, Console;
Pointers	Java does not work with pointers	SPARK does not work with pointers
GC	Java relies a lot on GC for its	SPARK does not allow

	memory allocation. Such constructs as for example recursion would be allowed in Java	allocating indefinite amount of memory

Encapsulation	Java is OO programming language and promotes encapsulation in pure form of this widespread paradigm
	
	`public class Alarm{`
	`    private boolean state;`
	
	`    public void enable()`
	`    {`
	`        state = true;`
	`    }`
	
	`    public void disable()`
	`    {`
	`        state = false;`
	`    }`
	
	`    …`
		Unit encapsulates all available objects within one file .adb and header file (specification) .ads specifies an interface to the unit
		`package body Alarm`
		`is`
		`    State: Boolean;`
		
		`    procedure Enable`
		`    is`
		`    begin`
		`        State := true;`
		`    end Enable;`
		
		`    procedure Disable`
		`    is`
		`    begin`
		`        State := false;`
		`    end Disable;`
		`…`

Code reuse	Promoted with the use of additional classes/packages that encapsulate desired functionality	Allows reuse with the inclusion of desired additional packages/units
Code proof	Java has additional tools for proving the software, written in Java programming language, but existing products have diverged and no official component exists. Dalvik VM tries to add this feature into the development environment	SPARK proof context defines a separate syntax for additional SPARK tools used for verification of software
Variables	a) Java does not have out variables b) In order to define a subtype, the whole new class must be defined (ranges do not exist)	a) SPARK has out variables (this might be thought as pointer access to a defined variable) Example:

<table>
<tr><td></td><td></td><td><pre>procedure Increment (X : in out Counter_Type);
is
begin
    X := X + 1;
end Increment;</pre>
b) SPARK defines subtypes in one line of code
Example:
<pre>subtype Speed_Type is Integer range 0..150;</pre></td></tr>
<tr><td>Multiple threads</td><td>Java promotes use of threads
Example:
<pre>Thread thread = new Thread(new Runnable() {
        public void run() {
            ...
        }
    });
thread.start();</pre></td><td>SPARK does not have threads</td></tr>
</table>

Table 3 SPARK vs Java

As it can be seen from the table given below, both languages have both limitations and benefits depending on their area of use. To state explicitly, Java focuses on performance and availability of the components as well as openness for the developer to be able to choose what he would like to use the language for. The choice can be made ranging from control systems to multimedia/entertainment applications with a very fast response time, but at the same time Java does not provide many tools for code verification and ability to be certain that the implemented code would not ever reach a case where run-time exception would occur. From practice and publicly available reports can be deduced that such cases are extremely high in any industry that chooses to use Java for implementation of its products.

SPARK, on the other hand, with obvious limitations in the area of multimedia, networking, GUIs, does not provide programmer with a long API. SPARK focuses on concise language features that are the most relevant when it comes to creating a responsive system that updates statuses of its components correspondingly.

6. Conclusion

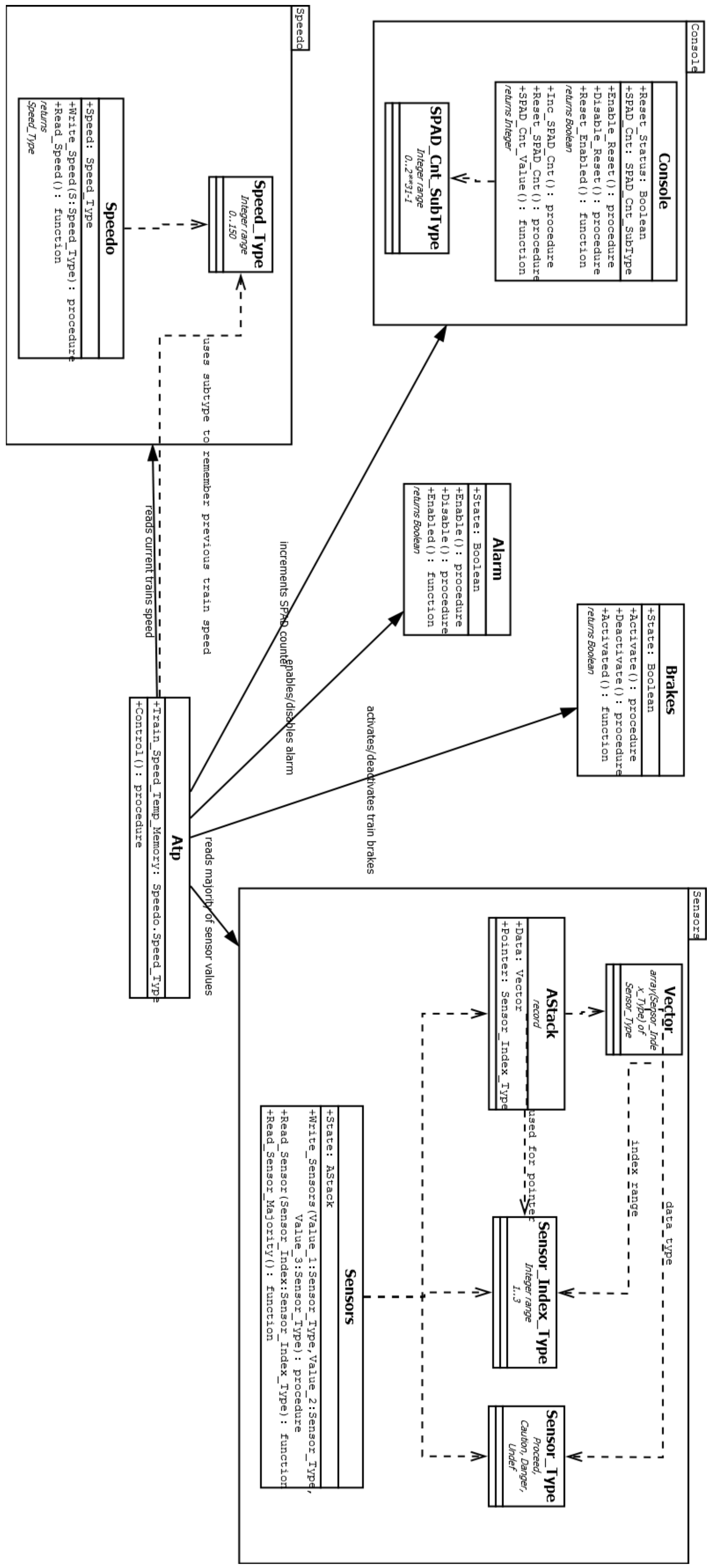
The purpose of this report is to give an overview of an implemented system based on the requirements and specification files given in advance. Some crucial assumptions were made during the implementation that affected the final design of the system. Identified assumptions have to be negotiated with the client (in this case - course coordinator) to be able to produce better requirements and to eliminate ambiguities in the requirements. Diagrams present in the report are included to make understanding of the current state of the implementation easier and reader is supposed to be able to spot problems instantly when going through assumptions, diagrams and additional files. No information was hidden from intended reader and it is expected that the intended reader would not agree with chosen design and made decisions. Overall the task was achieved and the working system is ready to be delivered. All the additional files needed for reviewing are attached in appendices.

At the end of the report one section was dedicated for comparison between Java programming language and SPARK and important aspects identified. Reader, after reading a table of comparisons should be able to have an understanding what SPARK has to offer and what it is suitable for.

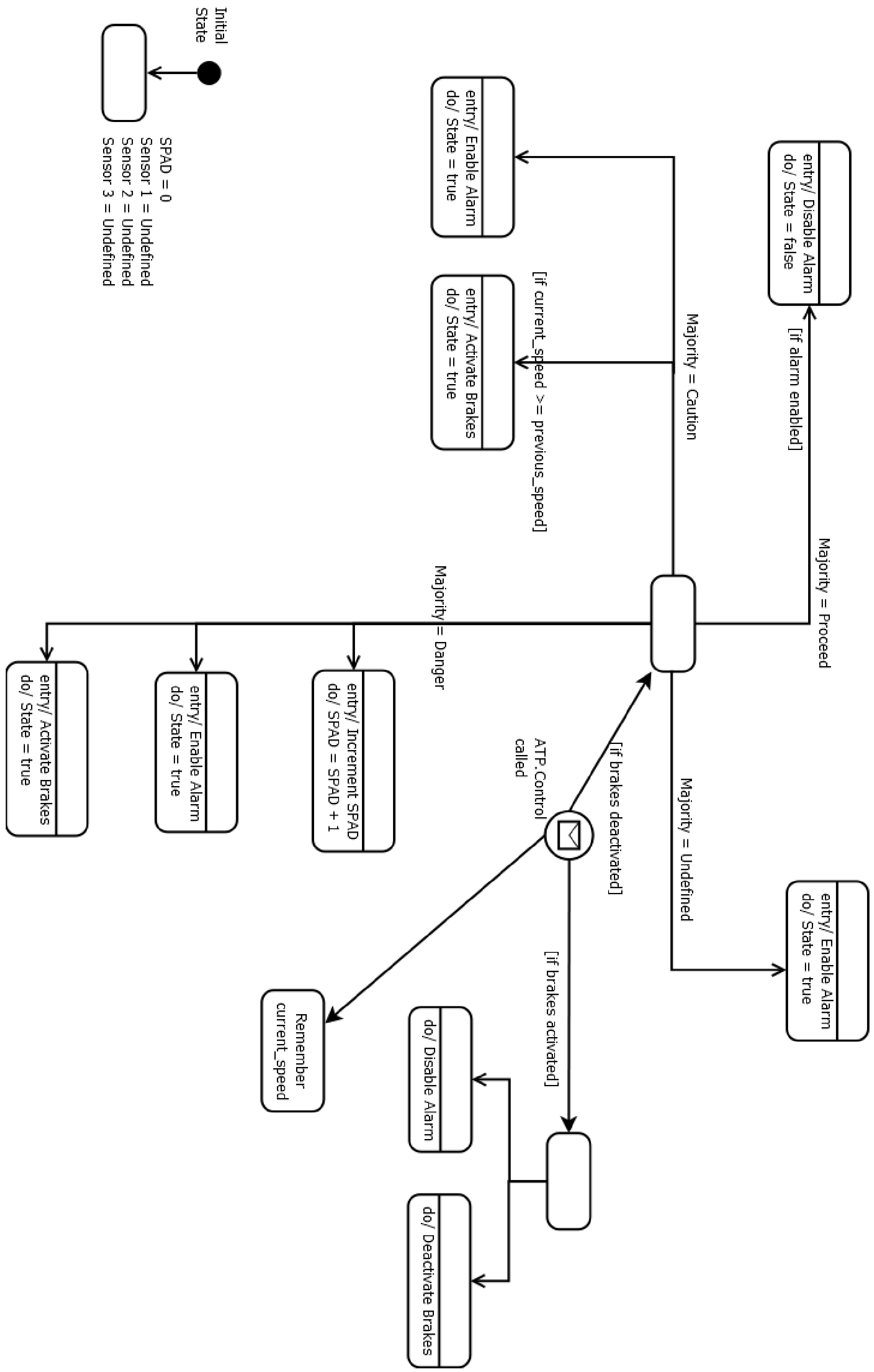
7. References

http://www.spark-2014.org/
http://docs.oracle.com/javase/7/docs/api/
High Integrity ADA: The Spark Approach (J G P Barnes)
Lecture slides

8. Appendix A Unit diagram



9. Appendix B State diagram



10. Appendix C Listing files

```
*******************************************************
                 Listing of SPARK Text
                   Examiner GPL 2012
  Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
*******************************************************


                 DATE : 23-OCT-2014 23:26:01.12


-- Author:                A. Ireland
--
-- Address:               School Mathematical & Computer Sciences
--                        Heriot-Watt University
--                        Edinburgh, EH14 4AS
--
-- E-mail:                a.ireland@hw.ac.uk
--
-- Last modified:         25/9/2013
--
-- Filename:              alarm.ads
--
-- Description:           Models the alarm device associated
--                        with the ATP controller.


package Alarm
   --# own State;
   --# initializes State;
is
   procedure Enable;
   --# global out State;
   --# derives State from ;

   procedure Disable;
   --# global out State;
   --# derives State from ;

   function Enabled return Boolean;
   --# global in State;

end Alarm;



Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
                 DATE : 23-OCT-2014 23:26:01.16

package body Alarm
is
     State: Boolean;

     procedure Enable
     is
     begin
                State := true;
     end Enable;

+++        Flow analysis of subprogram Enable performed
           (information-flow mode): no errors found.


     procedure Disable
     is
     begin
                State := false;
     end Disable;

+++        Flow analysis of subprogram Disable performed
           (information-flow mode): no errors found.


     function Enabled return Boolean
     is
     begin
                return State;
     end Enabled;

+++        Flow analysis of subprogram Enabled performed
           (information-flow mode): no errors found.

begin
     State := false;
end Alarm;

+++        Flow analysis of package initialization
           performed: no errors found.


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
*******************************************************
                 Listing of SPARK Text
                   Examiner GPL 2012
```


```
*******************************************************

                  DATE : 23-OCT-2014 23:26:01.21

--# inherit Sensors, Alarm, Brakes, Speedo, Console;
package ATP
     --# own Train_Speed_Temp_Memory;
     --# initializes Train_Speed_Temp_Memory;
is
     procedure Control;
     --# global in Sensors.State;
     --#        in out Brakes.State;
     --#        in out Alarm.State;
     --#        in Speedo.Speed;
     --#        in out Train_Speed_Temp_Memory;
     --#        in out Console.SPAD_Cnt;
     --# derives Alarm.State from Sensors.State, Alarm.State, Brakes.State &
     --#        Brakes.State from Sensors.State, Train_Speed_Temp_Memory, Speedo.Speed, Brakes.State, Alarm.State &
     --#        Train_Speed_Temp_Memory from Speedo.Speed &
     --#        Console.SPAD_Cnt from Console.SPAD_Cnt, Sensors.State, Brakes.State;
end ATP;

Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
           *****************************************************
                            Listing of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           *****************************************************


                        DATE : 23-OCT-2014 23:26:01.24

with Sensors, Alarm, Brakes, Speedo, Console;
package body ATP
is
     --used within atp package only
     Train_Speed_Temp_Memory: Speedo.Speed_Type;

     procedure Control
     is
                Sensors_Value: Sensors.Sensor_Type;
     begin
                if not Brakes.Activated then
                           Sensors_Value := Sensors.Read_Sensor_Majority;
                           case Sensors_Value is
                                      when Sensors.Proceed =>
                                                 if Alarm.Enabled then
                                                            Alarm.Disable;
                                                 end if;
                                      when Sensors.Caution =>
                                                 if Alarm.Enabled then
                                                            if Speedo.Read_Speed >=
Train_Speed_Temp_Memory then
                                                                       Brakes.Activate;
                                                            end if;
                                                 end if;
                                                 Alarm.Enable;
                                      when Sensors.Danger =>
                                                 Brakes.Activate;
                                                 Alarm.Enable;
                                                 Console.Inc_SPAD_Cnt;
                                      when others =>
                                                 Alarm.Enable;
                           end case;
                else
                           Alarm.Disable;
                           Brakes.Deactivate;
                end if;

                Train_Speed_Temp_Memory := Speedo.Read_Speed;
     end Control;

+++        Flow analysis of subprogram Control performed
           (information-flow mode): no errors found.


begin
     Train_Speed_Temp_Memory := 0;
end ATP;

+++        Flow analysis of package initialization
           performed: no errors found.


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
     *******************************************************
                      Listing of SPARK Text
                        Examiner GPL 2012
       Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
     *******************************************************


                    DATE : 23-OCT-2014 23:26:01.09


-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       25/9/2013
--
-- Filename:            brakes.ads
--
-- Description:         Models the train braking subsystem associated
--                      with the ATP controller.

package Brakes
   --# own State;
   --# initializes State;
is
   procedure Activate;
   --# global out State;
   --# derives State from ;

   procedure Deactivate;
   --# global out State;
   --# derives State from ;

   function Activated return Boolean;
   --# global in State;

end Brakes;


Note: Flow analysis mode is automatic

--End of file--------------------------------------------------
```

```
           ****************************************************
                          Listing of SPARK Text
                            Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ****************************************************


                        DATE : 23-OCT-2014 23:26:01.12

Line
      package body Brakes
      is
           State: boolean;

           procedure Activate
           is
           begin
                      State := true;
           end Activate;

+++        Flow analysis of subprogram Activate performed
           (information-flow mode): no errors found.


           procedure Deactivate
           is
           begin
                      State := false;
           end Deactivate;

+++        Flow analysis of subprogram Deactivate performed
           (information-flow mode): no errors found.


           function Activated return Boolean
           is
           begin
                      return State;
           end Activated;

+++        Flow analysis of subprogram Activated performed
           (information-flow mode): no errors found.

      begin
           State := false;
      end Brakes;

+++        Flow analysis of package initialization
           performed: no errors found.


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
        *****************************************************
                         Listing of SPARK Text
                           Examiner GPL 2012
          Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
        *****************************************************


                        DATE : 23-OCT-2014 23:26:01.17

Line

-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       25/9/2013
--
-- Filename:            reset.adb
--
-- Description:         Models the console associated with the ATP system, i.e.
--                      the reset mechanism that is required to disable the
--                      trains's braking system, as well as a SPAD count.

--# inherit Brakes, Alarm;
package  Console
  --# own Reset_Status, SPAD_Cnt;
  --# initializes Reset_Status, SPAD_Cnt;
is
   --subtype introduced to specify the range of SPAD counter. SPAD counter > 0 and < 2**31-1
   subtype SPAD_Cnt_SubType is Integer range 0..Integer'Last;

   procedure Enable_Reset;
      --# global out Reset_Status;
      --# derives Reset_Status from ;

   procedure Disable_Reset;
     --# global out Reset_Status;
     --# derives Reset_Status from ;

   function Reset_Enabled return Boolean;
     --# global in Reset_Status;

   procedure Inc_SPAD_Cnt;
     --# global in out SPAD_Cnt;
     --# derives SPAD_Cnt from SPAD_Cnt;

   procedure Reset_SPAD_Cnt;
      --# global out SPAD_Cnt;
      --# derives SPAD_Cnt from ;

   function SPAD_Cnt_Value return Integer;
      --# global in SPAD_Cnt;

end Console;


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
           *******************************************************
                           Listing of SPARK Text
                             Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           *******************************************************


                       DATE : 23-OCT-2014 23:26:01.21

Line
with Brakes, Alarm;
package body Console
is
       --hidden from packages using console
       Reset_Status: Boolean;
       SPAD_Cnt: SPAD_Cnt_SubType;

       procedure Enable_Reset
       is
       begin
                    Reset_Status := true;
       end Enable_Reset;

+++        Flow analysis of subprogram Enable_Reset
           performed (information-flow mode): no errors found.


       procedure Disable_Reset
       is
       begin
                    Reset_Status := false;
       end Disable_Reset;

+++        Flow analysis of subprogram Disable_Reset
           performed (information-flow mode): no errors found.


       function Reset_Enabled return Boolean
       is
       begin
                    return Reset_Status;
       end Reset_Enabled;

+++        Flow analysis of subprogram Reset_Enabled
           performed (information-flow mode): no errors found.


       procedure Inc_SPAD_Cnt
       is
       begin
                    if SPAD_Cnt < SPAD_Cnt_SubType'Last then
                    --# check SPAD_Cnt + 1 in Integer;
                    SPAD_Cnt := SPAD_Cnt + 1;
                    end if;
       end Inc_SPAD_Cnt;

+++        Flow analysis of subprogram Inc_SPAD_Cnt
           performed (information-flow mode): no errors found.


       procedure Reset_SPAD_Cnt
       is
       begin
                    SPAD_Cnt := 0;
       end Reset_SPAD_Cnt;

+++        Flow analysis of subprogram Reset_SPAD_Cnt
           performed (information-flow mode): no errors found.


       function SPAD_Cnt_Value return Integer
       is
       begin
                    return SPAD_Cnt;
       end SPAD_Cnt_Value;

+++        Flow analysis of subprogram SPAD_Cnt_Value
           performed (information-flow mode): no errors found.

begin
       Reset_Status := false;
       SPAD_Cnt := 0;
end Console;

+++        Flow analysis of package initialization
           performed: no errors found.


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
          *****************************************************
                         Listing of SPARK Text
                           Examiner GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
          *****************************************************


                       DATE : 23-OCT-2014 23:26:01.00
```

```

-- Author:             A. Ireland
--
-- Address:            School Mathematical & Computer Sciences
--                     Heriot-Watt University
--                     Edinburgh, EH14 4AS
--
-- E-mail:             a.ireland@hw.ac.uk
--
-- Last modified:      25/9/2013
--
-- Filename:           sensors.ads
--
-- Description:        Models the 3 sensors associated with the ATP system. Note that
--                     a single sensor reading is calculated using a majority vote
--                     algorithm.

package Sensors
   --# own State;
   --# initializes State;
is
   type Sensor_Type is (Proceed, Caution, Danger, Undef);
   subtype Sensor_Index_Type is Integer range 1..3;

   -- initial global in State changed to global in out State. This needed to be done as a consequence of the information hiding about State
   procedure Write_Sensors(Value_1, Value_2, Value_3: in Sensor_Type);
   --# global in out State;
   --# derives State from Value_1, Value_2, Value_3, State;

   function Read_Sensor(Sensor_Index: in Sensor_Index_Type) return Sensor_Type;
   --# global in State;

   function Read_Sensor_Majority return Sensor_Type;
   --# global in State;

end Sensors;


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
*******************************************************
                Listing of SPARK Text
                  Examiner GPL 2012
  Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
*******************************************************


            DATE : 23-OCT-2014 23:26:01.06

package body Sensors
--# own State is AStack;
is
       --state is hidden and is comprised of an array and a pointer
       type Vector is array(Sensor_Index_Type) of Sensor_Type;
       type Stack is
                    record
                                Data: Vector;
                                Pointer: Sensor_Index_Type;
                    end record;
       AStack: Stack;


       procedure Write_Sensors(Value_1, Value_2, Value_3: in Sensor_Type)
       --# global in out AStack;
       --# derives AStack from Value_1, Value_2, Value_3, AStack;
       is
       begin
                    AStack.Pointer := 1;
                    AStack.Data(AStack.Pointer) := Value_1;
                    AStack.Pointer := AStack.Pointer + 1;
                    AStack.Data(AStack.Pointer) := Value_2;
                    AStack.Pointer := AStack.Pointer + 1;
                    AStack.Data(AStack.Pointer) := Value_3;
       end Write_Sensors;

+++        Flow analysis of subprogram Write_Sensors
           performed (information-flow mode): no errors found.


       function Read_Sensor(Sensor_Index: in Sensor_Index_Type) return Sensor_Type
       --# global in AStack;
       is
       begin
                    return AStack.Data(Sensor_Index);
       end Read_Sensor;

+++        Flow analysis of subprogram Read_Sensor
           performed (information-flow mode): no errors found.


       function Read_Sensor_Majority return Sensor_Type
       --# global in AStack;
       --# return AResult => ((AStack.Data(1) /= AStack.Data(2) and AStack.Data(1) /= AStack.Data(3) and AStack.Data(2) /= AStack.Data(3)) -> AResult = Undef) and
       --# (AStack.Data(1) = AStack.Data(2) -> AResult = AStack.Data(1)) and
       --# (AStack.Data(1) = AStack.Data(3) -> AResult = AStack.Data(1)) and
       --# (AStack.Data(2) = AStack.Data(3) -> AResult = AStack.Data(2));
       is
                    AResult: Sensor_Type;
       begin
                    AResult := Undef;
                    if AStack.Data(1) /= AStack.Data(2) AND AStack.Data(1) /= AStack.Data(3) AND AStack.Data(2) /= AStack.Data(3) then
                                AResult := Undef;
                    else
                                if AStack.Data(1) = AStack.Data(2) then
                                            AResult := AStack.Data(1);
                                elsif AStack.Data(1) = AStack.Data(3) then
                                            AResult := AStack.Data(1);
                                elsif AStack.Data(2) = AStack.Data(3) then
                                            AResult := AStack.Data(2);
                                end if;
                    end if;

                    return AResult;
       end Read_Sensor_Majority;

+++        Flow analysis of subprogram Read_Sensor_Majority
           performed (information-flow mode): no errors found.


begin
       --simultaneous assignment using aggregate construct
       AStack.Data := Vector'(Sensor_Index_Type => Undef);
       AStack.Pointer := 1;
end Sensors;

+++        Flow analysis of package initialization
           performed: no errors found.


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

```
*******************************************************
                 Listing of SPARK Text
                   Examiner GPL 2012
  Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
*******************************************************


                DATE : 23-OCT-2014 23:26:01.07
```

```

-- Author:             A. Ireland
--
-- Address:            School Mathematical & Computer Sciences
--                     Heriot-Watt University
--                     Edinburgh, EH14 4AS
--
-- E-mail:             a.ireland@hw.ac.uk
--
-- Last modified:      25/9/2013
--
-- Filename:           speedo.ads
--
-- Description:        Models the speedo device associated with the ATP system.

package Speedo
   --# own Speed;
   --# initializes Speed;
is

   subtype Speed_Type is Integer range 0..150;

   procedure Write_Speed(S: in Speed_Type);
   --# global out Speed;
   --# derives Speed from S;

   function Read_Speed return Speed_Type;
   --# global in Speed;

end Speedo;


```

```
Note: Flow analysis mode is automatic

--End of file--------------------------------------------------
```

```
           *****************************************************
                            Listing of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           *****************************************************


                        DATE : 23-OCT-2014 23:26:01.09

Line
      package body Speedo
      is
           Speed: Speed_Type;

           procedure Write_Speed(S: in Speed_Type)
           is
           begin
                      Speed := S;
           end Write_Speed;

+++        Flow analysis of subprogram Write_Speed
           performed (information-flow mode): no errors found.


           function Read_Speed return Speed_Type
           is
           begin
                      return Speed;
           end Read_Speed;

+++        Flow analysis of subprogram Read_Speed performed
           (information-flow mode): no errors found.


           begin
                      Speed := 0;
      end Speedo;

+++        Flow analysis of package initialization
           performed: no errors found.


Note: Flow analysis mode is automatic


--End of file--------------------------------------------------
```

11. Appendix D Report file

```
          *******************************************************
                       Report of SPARK Examination
                            Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
          *******************************************************


                        DATE : 23-OCT-2014 23:26:01.24

Options:
    noswitch
    noindex_file
    nowarning_file
    notarget_compiler_data
    config_file=gnat.cfg
    source_extension=ada
    listing_extension=lst
    nodictionary_file
    report_file=spark.rep
    nohtml
    vcg
    nostatistics
    fdl_identifiers=accept
    flow_analysis=auto
    language=95
    profile=sequential
    annotation_character=#
    rules=lazy
    error_explanations=off
    justification_option=full
    output_directory=.
    output_directory (actual)=/home/msc/bm4/public_html/RMSE/CW1/

Selected files:
   @atp.smf


No Index files were used


Meta File(s) used were:
   atp.smf
      /home/msc/bm4/public_html/RMSE/CW1/sensors.ads
      /home/msc/bm4/public_html/RMSE/CW1/sensors.adb
      /home/msc/bm4/public_html/RMSE/CW1/speedo.ads
      /home/msc/bm4/public_html/RMSE/CW1/speedo.adb
      /home/msc/bm4/public_html/RMSE/CW1/brakes.ads
      /home/msc/bm4/public_html/RMSE/CW1/brakes.adb
      /home/msc/bm4/public_html/RMSE/CW1/alarm.ads
      /home/msc/bm4/public_html/RMSE/CW1/alarm.adb
      /home/msc/bm4/public_html/RMSE/CW1/console.ads
      /home/msc/bm4/public_html/RMSE/CW1/console.adb
      /home/msc/bm4/public_html/RMSE/CW1/atp.ads
      /home/msc/bm4/public_html/RMSE/CW1/atp.adb


Full warning reporting selected


Target configuration file:
Line
   1  package Standard is
   2        type Integer is range -2**31 .. 2**31-1;
   3  end Standard;


Source Filename(s) used were:
   /home/msc/bm4/public_html/RMSE/CW1/sensors.ads
   /home/msc/bm4/public_html/RMSE/CW1/sensors.adb
   /home/msc/bm4/public_html/RMSE/CW1/speedo.ads
   /home/msc/bm4/public_html/RMSE/CW1/speedo.adb
   /home/msc/bm4/public_html/RMSE/CW1/brakes.ads
   /home/msc/bm4/public_html/RMSE/CW1/brakes.adb
   /home/msc/bm4/public_html/RMSE/CW1/alarm.ads
   /home/msc/bm4/public_html/RMSE/CW1/alarm.adb
   /home/msc/bm4/public_html/RMSE/CW1/console.ads
   /home/msc/bm4/public_html/RMSE/CW1/console.adb
   /home/msc/bm4/public_html/RMSE/CW1/atp.ads
   /home/msc/bm4/public_html/RMSE/CW1/atp.adb



Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/sensors.ads
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/sensors.ads.lst

   Unit name:  Sensors
   Unit type:  package specification
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/sensors.adb
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/sensors.adb.lst

   Unit name:  Sensors
   Unit type:  package body
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/speedo.ads
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/speedo.ads.lst

   Unit name:  Speedo
   Unit type:  package specification
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/speedo.adb
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/speedo.adb.lst
```

```
   Unit name:  Speedo
   Unit type:  package body
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/brakes.ads
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/brakes.ads.lst

   Unit name:  Brakes
   Unit type:  package specification
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/brakes.adb
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/brakes.adb.lst

   Unit name:  Brakes
   Unit type:  package body
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/alarm.ads
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/alarm.ads.lst

   Unit name:  Alarm
   Unit type:  package specification
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/alarm.adb
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/alarm.adb.lst

   Unit name:  Alarm
   Unit type:  package body
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/console.ads
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/console.ads.lst

   Unit name:  Console
   Unit type:  package specification
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/console.adb
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/console.adb.lst

   Unit name:  Console
   Unit type:  package body
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/atp.ads
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/atp.ads.lst

   Unit name:  ATP
   Unit type:  package specification
   Unit has been analysed, any errors are listed below.

No errors found


Source Filename:   /home/msc/bm4/public_html/RMSE/CW1/atp.adb
Listing Filename:  /home/msc/bm4/public_html/RMSE/CW1/atp.adb.lst

   Unit name:  ATP
   Unit type:  package body
   Unit has been analysed, any errors are listed below.

No errors found

Note: Automatic flow analysis mode selected


--End of file--------------------------------------------------
```

12. Appendix E Log files

SENSOR-1	SENSOR-2	SENSOR-3	MAJORITY	SPEED	ALARM	BRAKES	RESET	SPADs
--------	--------	--------	--------	-----	-----	------	-----	-----
PROCEED	PROCEED	PROCEED	PROCEED	50	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	50	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	55	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	55	--	--	--	0
CAUTION	CAUTION	CAUTION	CAUTION	56	--	--	--	0
CAUTION	CAUTION	CAUTION	CAUTION	56	ON	--	--	0
CAUTION	CAUTION	CAUTION	CAUTION	55	ON	--	--	0
CAUTION	CAUTION	CAUTION	CAUTION	55	ON	--	--	0
CAUTION	CAUTION	CAUTION	CAUTION	54	ON	--	--	0
CAUTION	CAUTION	CAUTION	CAUTION	54	ON	--	--	0
CAUTION	DANGER	DANGER	DANGER	59	ON	--	--	0
CAUTION	DANGER	DANGER	DANGER	59	ON	ON	--	1
DANGER	PROCEED	DANGER	DANGER	60	ON	ON	ON	1
DANGER	PROCEED	DANGER	DANGER	60	--	--	ON	1
DANGER	CAUTION	CAUTION	CAUTION	66	--	--	--	1
DANGER	CAUTION	CAUTION	CAUTION	66	ON	--	--	1
PROCEED	PROCEED	PROCEED	PROCEED	67	ON	--	--	1
PROCEED	PROCEED	PROCEED	PROCEED	67	--	--	--	1
PROCEED	CAUTION	PROCEED	PROCEED	69	--	--	--	1
PROCEED	CAUTION	PROCEED	PROCEED	69	--	--	--	1

SENSOR-1	SENSOR-2	SENSOR-3	MAJORITY	SPEED	ALARM	BRAKES	RESET	SPADs
--------	--------	--------	--------	-----	-----	------	-----	-----
PROCEED	PROCEED	PROCEED	PROCEED	0	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	0	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	25	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	25	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	50	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	50	--	--	--	0
PROCEED	PROCEED	DANGER	PROCEED	53	--	--	--	0
PROCEED	PROCEED	DANGER	PROCEED	53	--	--	--	0
CAUTION	PROCEED	PROCEED	PROCEED	58	--	--	--	0
CAUTION	PROCEED	PROCEED	PROCEED	58	--	--	--	0
CAUTION	CAUTION	PROCEED	CAUTION	59	--	--	--	0
CAUTION	CAUTION	PROCEED	CAUTION	59	ON	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	60	ON	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	60	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	62	--	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	62	--	--	--	0
CAUTION	PROCEED	CAUTION	CAUTION	63	--	--	--	0
CAUTION	PROCEED	CAUTION	CAUTION	63	ON	--	--	0
PROCEED	CAUTION	CAUTION	CAUTION	62	ON	--	--	0
PROCEED	CAUTION	CAUTION	CAUTION	62	ON	--	--	0
CAUTION	CAUTION	PROCEED	CAUTION	61	ON	--	--	0
CAUTION	CAUTION	PROCEED	CAUTION	61	ON	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	62	ON	--	--	0
PROCEED	PROCEED	PROCEED	PROCEED	62	--	--	--	0
DANGER	DANGER	CAUTION	DANGER	60	--	--	--	0
DANGER	DANGER	CAUTION	DANGER	60	ON	ON	--	1
PROCEED	PROCEED	PROCEED	PROCEED	0	ON	ON	ON	1
PROCEED	PROCEED	PROCEED	PROCEED	0	--	--	ON	1
PROCEED	PROCEED	PROCEED	PROCEED	10	--	--	--	1
PROCEED	PROCEED	PROCEED	PROCEED	10	--	--	--	1
PROCEED	CAUTION	DANGER	UNDEF	30	--	--	--	1
PROCEED	CAUTION	DANGER	UNDEF	30	ON	--	--	1
PROCEED	PROCEED	PROCEED	PROCEED	25	ON	--	ON	1
PROCEED	PROCEED	PROCEED	PROCEED	25	--	--	ON	1
PROCEED	PROCEED	PROCEED	PROCEED	27	--	--	--	1
PROCEED	PROCEED	PROCEED	PROCEED	27	--	--	--	1
UNDEF	UNDEF	UNDEF	UNDEF	25	--	--	--	1
UNDEF	UNDEF	UNDEF	UNDEF	25	ON	--	--	1
PROCEED	PROCEED	PROCEED	PROCEED	25	ON	--	ON	1
PROCEED	PROCEED	PROCEED	PROCEED	25	--	--	ON	1
CAUTION	CAUTION	DANGER	CAUTION	27	--	--	--	1
CAUTION	CAUTION	DANGER	CAUTION	27	ON	--	--	1
CAUTION	DANGER	CAUTION	CAUTION	28	ON	--	--	1
CAUTION	DANGER	CAUTION	CAUTION	28	ON	ON	--	1
DANGER	CAUTION	CAUTION	CAUTION	29	ON	ON	--	1
DANGER	CAUTION	CAUTION	CAUTION	29	--	--	--	1

13. Appendix F POGS report