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

1. Introduction ................................................................................................................................. 4
2. Requirements ................................................................................................................................. 5
  2.1. Assumptions ............................................................................................................................. 7
3. Architecture ................................................................................................................................. 9
  3.1. Unit diagram ............................................................................................................................ 9
  3.2. State diagram ........................................................................................................................... 9
4. Testing & formal proof .................................................................................................................... 10
  4.1. Listing file ............................................................................................................................... 10
  4.2. Report file ............................................................................................................................... 10
  4.3. Log ........................................................................................................................................ 10
  4.4. Pogs ....................................................................................................................................... 10
5. Comparing SPARK with Java ...................................................................................................... 11
6. Conclusion .................................................................................................................................. 15
7. References .................................................................................................................................... 15
8. Appendix A Unit diagram ............................................................................................................ 16
9. Appendix B State diagram ........................................................................................................... 17
10. Appendix C Listing files ........................................................................................................... 18
11. Appendix D Report file ............................................................................................................. 30
12. Appendix E Log files ................................................................................................................ 32
13. Appendix F POGS report ......................................................................................................... 32
Tables

Table 1: Requirements ........................................................................................................... 6
Table 2: Assumptions ............................................................................................................. 8
Table 3 SPARK vs Java .......................................................................................................... 14
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.

<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors</td>
<td>Maintains and provides access to sensor values</td>
<td>Develop package bodies consistent with specifications provided</td>
</tr>
<tr>
<td>Speedo</td>
<td>Maintains and provides access to speedometer values</td>
<td></td>
</tr>
<tr>
<td>Brakes</td>
<td>Provides control and maintains state of train brakes</td>
<td></td>
</tr>
<tr>
<td>Alarm</td>
<td>Provides control and maintains state of alarm</td>
<td></td>
</tr>
<tr>
<td>Console</td>
<td>Provides interface to ATP controller</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Provides control and maintains state of reset subsystem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maintains count of SPAD events(^1)</td>
<td></td>
</tr>
<tr>
<td>ATP</td>
<td>Overall control of ATP system</td>
<td>Develop specification and body consistent with first 5 packages and test harness</td>
</tr>
</tbody>
</table>

\(^1\) SPAD - ‘Signal Passed at Danger’ event
<table>
<thead>
<tr>
<th>ALL</th>
<th>Support confidence in developed system</th>
<th>Prove run-time exception freedom using SPARK proof tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors</td>
<td>Be confident that the function returns expected result</td>
<td>Use <em>return</em> annotation to specify correctness of the <em>Read_Sensor_Majority</em> function</td>
</tr>
<tr>
<td>ALL</td>
<td>Verify that the implementation is consistent with the specification</td>
<td>Using SPARK proof tools verify that package bodies satisfy specifications</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assuming that undefined signal is of lower level than caution signal</td>
<td>Alarm is enabled when undefined signal is detected by the ATP controller</td>
</tr>
<tr>
<td>Assuming that train stops or slows down considerably after brakes are engaged</td>
<td>After brakes are engaged, system reset is triggered</td>
</tr>
<tr>
<td>Assuming that console is a responsive unit and does not act upon the rest of the system</td>
<td>Console is implemented as a responsive unit, which does not affect the entire system directly</td>
</tr>
<tr>
<td>Assuming there are precisely 3 sensors</td>
<td>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</td>
</tr>
<tr>
<td>Assuming that when proceed signal is detected, alarm should be disabled if brakes are not engaged</td>
<td>Alarm is disabled irrespective of the previous state</td>
</tr>
<tr>
<td>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</td>
<td>SPAD counter is only incremented when danger signal is detected. Deduced based on the abbreviation: “Signal Passed At Danger”</td>
</tr>
<tr>
<td>Assuming that the SPAD counter must</td>
<td>SPAD counter is set to be between 0 and</td>
</tr>
<tr>
<td>Assumption</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>be a positive value from 0 to $2^{31} - 1$</td>
<td>$2^{31} - 1$</td>
</tr>
<tr>
<td>ATP is the only unit that can directly influence the rest of the system</td>
<td>ATP unit sends commands to the rest of the system (active unit)</td>
</tr>
<tr>
<td>Assuming that the whole system, together with the ATP unit, is responsive and does not act upon its state initiatively</td>
<td>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.</td>
</tr>
<tr>
<td>Assuming that when system starts, sensors values are unknown</td>
<td>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</td>
</tr>
<tr>
<td>Assumption is made that the system never terminates</td>
<td>System is passive and ATP.Control procedure awaits to be triggered</td>
</tr>
<tr>
<td>Assuming train stops after brakes are activated</td>
<td>After system activates brakes, reset is triggered</td>
</tr>
<tr>
<td>Assuming ‘subsequent’ means the next signal after</td>
<td>Previous signals are not remembered. On the next run, if proceed signal is returned, control procedure disables alarm if it was previously enabled</td>
</tr>
</tbody>
</table>

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 sparsimp 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)
<table>
<thead>
<tr>
<th>Feature</th>
<th>Java</th>
<th>SPARK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flow controls</strong></td>
<td>if (AStack.Data(1) != AStack.Data(2)</td>
<td>if AStack.Data(1) /= AStack.Data(2)</td>
</tr>
<tr>
<td>(i.e. if … else, break,</td>
<td>&amp;&amp; AStack.Data(1) != AStack.Data(3)</td>
<td>AND AStack.Data(1) /= AStack.Data(3)</td>
</tr>
<tr>
<td>continue, try,</td>
<td>&amp;&amp; AStack.Data(2) != AStack.Data(3)</td>
<td>AND AStack.Data(2) /= AStack.Data(3)</td>
</tr>
<tr>
<td>catch, while, for, etc.)</td>
<td>then</td>
<td>then</td>
</tr>
<tr>
<td></td>
<td>AResult = Undef;</td>
<td>AResult := Undef;</td>
</tr>
<tr>
<td></td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>if (AStack.Data(1) == AStack.Data(2)) then</td>
<td>if AStack.Data(1) = AStack.Data(2) then</td>
</tr>
<tr>
<td></td>
<td>AResult := AStack.Data(1);</td>
<td>AResult := AStack.Data(1);</td>
</tr>
<tr>
<td></td>
<td>else if (AStack.Data(1) == AStack.Data(3)) then</td>
<td>elsif AStack.Data(1) = AStack.Data(3) then</td>
</tr>
<tr>
<td></td>
<td>AResult := AStack.Data(1);</td>
<td>AResult := AStack.Data(1);</td>
</tr>
<tr>
<td></td>
<td>else if (AStack.Data(2) == AStack.Data(3)) then</td>
<td>elsif AStack.Data(2) = AStack.Data(3) then</td>
</tr>
<tr>
<td></td>
<td>AResult := AStack.Data(2);</td>
<td>AResult := AStack.Data(2);</td>
</tr>
<tr>
<td></td>
<td>break; / return; / continue;</td>
<td>end if;</td>
</tr>
<tr>
<td></td>
<td>try{</td>
<td>Proof is used instead of try ... catch flow control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>catch&lt;stmt&gt; ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>while&lt;stmt&gt;</td>
<td>loop</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>for(int i=0;i&lt;const;i++)</td>
<td>for I in &lt;range&gt; loop</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td><strong>Headers</strong></td>
<td>Import is used to import external classes to be able to reference</td>
<td>.ads file identifies specification of a package</td>
</tr>
<tr>
<td>(.h, .ads, uses,</td>
<td>them. No header file exists.</td>
<td>.adb file identifies body of a package</td>
</tr>
<tr>
<td>include, etc.)</td>
<td>Information hiding is done via private/public access modifiers</td>
<td>Units are imported/included into current package via inherit/with</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td>keywords. Once unit is included, its objects (defined in .ads file)</td>
</tr>
<tr>
<td></td>
<td>import java.rmi.*;</td>
<td>are accessible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with Sensors, Alarm, Brakes, Speedo, Console;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--# inherit Sensors, Alarm, Brakes, Speedo, Console;</td>
</tr>
<tr>
<td><strong>Pointers</strong></td>
<td>Java does not work with pointers</td>
<td>SPARK does not work with pointers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GC</strong></td>
<td>Java relies a lot on GC for its</td>
<td>SPARK does not allow</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>Java is an object-oriented programming language and promotes encapsulation in pure form of this widespread paradigm.</td>
<td>public class Alarm{</td>
</tr>
<tr>
<td></td>
<td></td>
<td>private boolean state;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>public void enable(){</td>
</tr>
<tr>
<td></td>
<td></td>
<td>state = true;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>public void disable(){</td>
</tr>
<tr>
<td></td>
<td></td>
<td>state = false;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Code reuse</td>
<td>Promoted with the use of additional classes/packages that encapsulate desired functionality.</td>
<td>package body Alarm is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>State: Boolean;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>procedure Enable is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>begin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>State := true;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>end Enable;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>procedure Disable is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>begin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>State := false;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>end Disable;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Code proof</td>
<td>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.</td>
<td>SPARK proof context defines a separate syntax for additional SPARK tools used for verification of software</td>
</tr>
<tr>
<td>Variables</td>
<td>a) Java does not have out variables.</td>
<td>a) SPARK has out variables (this might be thought as pointer access to a defined variable)</td>
</tr>
<tr>
<td></td>
<td>b) In order to define a subtype, the whole new class must be defined (ranges do not exist).</td>
<td>Example:</td>
</tr>
</tbody>
</table>
procedure Increment (X : in out Counter_Type); is begin X := X + 1; end Increment;

b) SPARK defines subtypes in one line of code
Example:
subtype Speed_Type is Integer range 0..150;

<table>
<thead>
<tr>
<th>Multiple threads</th>
<th>Java promotes use of threads</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Thread thread = new Thread(new Runnable() {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>public void run() {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>});</td>
</tr>
<tr>
<td></td>
<td></td>
<td>thread.start();</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPARK does not have threads</th>
</tr>
</thead>
</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
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.
***************************************************************


Line
1
2 -- Author:    A. Ireland
3 --
4 -- Address:   School Mathematical & Computer Sciences
5 --
6 --
7 -- Edinburgh, EH14 4AS
8 --
9 --
10 -- Last modified: 25/9/2013
11 --
12 -- Filename:    alarm.ads
13 --
14 -- Description: Models the alarm device associated
15 --
16 with the ATP controller.
17
18 package Alarm
19 ---# own State;
20 ---# initializes State;
21 is
22 procedure Enable;
23 ---# global out State;
24 ---# derives State from ;
25
26 procedure Disable;
27 ---# global out State;
28 ---# derives State from ;
29
30 function Enabled return Boolean;
31 ---# global in State;
32
33 end Alarm;
34
35
Note: Flow analysis mode is automatic

--End of file--------------------------------------------------
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------------------------------------------
### ATP

```spc
-- 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;
--- 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-----------------------------------------------
package body ATP is

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;
                    Alarm.Enable;
                end if;

            when Sensors.Danger |
                Brakes.Activate;
                Alarm.Enable;
                Console.Inc_SPAD_Cnt;
            when others |
                if Alarm.Enabled then
                    Alarm.Enable;
                end case;
        end if;

        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
package Brakes

procedure Activate;

procedure Deactivate;

function Activated return Boolean;

end Brakes;

Note: Flow analysis mode is automatic

--End of file--------------------------------------------------
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
1
2 -- Author: A. Ireland
3 --
4 -- Address: School Mathematical & Computer Sciences
5 -- Heriot-Watt University
6 -- Edinburgh, EH14 4AS
7 --
8 -- E-mail: a.ireland@hw.ac.uk
9 --
10 -- Last modified: 25/9/2013
11 --
12 -- Filename: reset.adb
13 --
14 -- Description: Models the console associated with the ATP system, i.e.
15 -- the reset mechanism that is required to disable the
16 -- trains's braking system, as well as a SPAD count.
17
18 --# inherit Brakes, Alarm;
19 package Console
20 --# own Reset_Status, SPAD_Cnt;
21 --# initializes Reset_Status, SPAD_Cnt;
22 is
23 --subtype introduced to specify the range of SPAD counter. SPAD counter > 0 and < 2**31-1
24 subtype SPAD_Cnt_SubType is Integer range 0..Integer'Last;
25
26 procedure Enable_Reset;
27 --# global out Reset_Status;
28 --# derives Reset_Status from ;
29
30 procedure Disable_Reset;
31 --# global out Reset_Status;
32 --# derives Reset_Status from ;
33
34 function Reset_Enabled return Boolean;
35 --# global in Reset_Status;
36
37 procedure Inc_SPAD_Cnt;
38 --# global in out SPAD_Cnt;
39 --# derives SPAD_Cnt from SPAD_Cnt;
40
41 procedure Reset_SPAD_Cnt;
42 --# global out SPAD_Cnt;
43 --# derives SPAD_Cnt from ;
44
45 function SPAD_Cnt_Value return Integer;
46 --# global in SPAD_Cnt;
47
48 end Console;
49
50
Note: Flow analysis mode is automatic

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

24
package body Console

begin
    Reset_Status := true;
end Enable_Reset;

procedure Disable_Reset
begin
    Reset_Status := false;
end Disable_Reset;

function Reset_Enabled return Boolean
begin
    return Reset_Status;
end Reset_Enabled;

procedure Inc_SPAD_Cnt
begin
    if SPAD_Cnt < SPAD_Cnt_SubType'Last then
        SPAD_Cnt := SPAD_Cnt + 1;
    end if;
end Inc_SPAD_Cnt;

procedure Reset_SPAD_Cnt
begin
    SPAD_Cnt := 0;
end Reset_SPAD_Cnt;

begin
    Reset_Status := false;
    SPAD_Cnt := 0;
end Console;
package Sensors 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
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)
--- 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;

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;

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))
  --- and (AStack.Data(1) = AStack.Data(2) and AStack.Data(1) = AStack.Data(3) and AStack.Data(2) = AStack.Data(3))
  --- and (AStack.Data(1) = AStack.Data(3) and AStack.Data(2) = AStack.Data(3))
  --- and (AStack.Data(2) = AStack.Data(3));
  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;

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

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

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

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

Note: Flow analysis mode is automatic

--End of file--------------------------------------------------
package Speedo

is

subtype Speed_Type is Integer range 0..150;

procedure Write_Speed(S: in Speed_Type);

function Read_Speed return Speed_Type;

end Speedo;

Note: Flow analysis mode is automatic

--End of file---------------------------------------------
package body Speedo is
 Speed: Speed_Type;

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

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

begin
  Speed := 0;
end Speedo;

--End of file-----------------------------------------------
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_compile
r_data
config_file=gnat.cfg
source_extension=ada
listing_extension=lst
nodictionary_file
report_file=spark.rep
nohtml

No Index files were used

Meta File(s) used were:
/atp.smf

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

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

No errors found
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: 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.ads
Listing Filename: /home/msc/bm4/public_html/RMSE/CW1/alarm.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.ads
Listing Filename: /home/msc/bm4/public_html/RMSE/CW1/console.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.ads
Listing Filename: /home/msc/bm4/public_html/RMSE/CW1/atp.ads.lst

Note: Automatic flow analysis mode selected

--End of file--------------------------------------------------
12. Appendix E Log files

<table>
<thead>
<tr>
<th>SENSOR-1</th>
<th>SENSOR-2</th>
<th>SENSOR-3</th>
<th>MAJORITY</th>
<th>SPEED</th>
<th>ALARM</th>
<th>BRAKES</th>
<th>RESET</th>
<th>SPADs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>50</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>50</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>55</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>55</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>56</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>56</td>
<td>ON</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>55</td>
<td>ON</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>55</td>
<td>ON</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>54</td>
<td>ON</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>54</td>
<td>ON</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>59</td>
<td>ON</td>
<td>--</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>CAUTION</td>
<td>DANGER</td>
<td>DANGER</td>
<td>DANGER</td>
<td>60</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>1</td>
</tr>
<tr>
<td>DANGER</td>
<td>PROCEED</td>
<td>DANGER</td>
<td>PROCEED</td>
<td>60</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>1</td>
</tr>
<tr>
<td>DANGER</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>CAUTION</td>
<td>66</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>67</td>
<td>ON</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>69</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>PROCEED</td>
<td>CAUTION</td>
<td>PROCEED</td>
<td>PROCEED</td>
<td>69</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
</tbody>
</table>

13. Appendix F POGS report

32