Machin Design

18-3-2015

2IO70

The purpose of this document is to present a Java program that realises the functions specified in the Software Specification document. This program is an intermediate step towards writing the PP2 code that controls the sorting machine.

Version 0.1

Group 16

Rolf Verschuuren

Wigger Boelens

Stefan van den Berg

Dat Phung

Maarten Keet

Tudor Petrescu

**Contents**

[Coding Standards 3](#_Toc414452220)

[Translating to pseudo java: 3](#_Toc414452221)

[Validation of java to transition table 4](#_Toc414452222)

[Validation of timerManage 4](#_Toc414452223)

# Coding Standards

The java pseudo code follows the Google Java Style.   
Source to Google Java Style:<https://google-styleguide.googlecode.com/svn/trunk/javaguide.html>

## Translating to pseudo java:

The java program starts by declaring the output variables. The names of the output variables will keep their original name, without spaces, in a camelCase form. The variable type will be determined from the Output table.

The inputs follow the same pattern.

Every states is represented as a function, keeping their name in the camelCase fashion, they will be all void functions due to the fact that they do not return anything.

Every state function will run preconditions if any, then check for specific input values using ***if*** statements, if an ***if*** statement is satisfied, there will be changes to the output values to match the next states output values, also the display is set to output the next states number, and then the next state function is called according to the state transition diagram, if no ***if*** statement is satisfied the current function is recalled.

The program is always looping, consequence of no deadlocks in the state machine as proven by the uppaal model test.

Example: Initial -> Calibrate\_Sensor

So in this example the function initial is currently running, there are no preconditions to be checked, if the inputs have the desired value, in our case we check if the push button is pressed by the sorter, if so we will have the sorter moved down by activating the sorter motor via having the Hbridge0 variable set to 1. After this we set the display to showcase the number 2 then call calibrateSensor function and if the ***if*** statement wasn’t satisfied we recall initial entering a loop.

# Validation of java to transition table

Every state is represented by a function. The if statements in that function are the transitions which can occur from that state. The timer interrupt and the abort transitions are not represented as if statements, because interrupts go to a separate state(function). In those if statements the values that have to change are changed. The display will also be updated to the correct number of the state. The function timerManage is called in each state. Because with that function we make sure that the all outputs have the correct voltage.

We checked that all states are represented in the java code by a function. We also checked if they have all the transitions as if statements and that the correct values are changed.

# Validation of timerManage

Loop invariant:

All elements before the current element of the array have been set on if they had to be on.

Initialize:

We start with the first element. Thus there are no elements before it and the loop invariant holds.

Step case:

If we’re at element k, then according to the loop invariant all elements before k have been set on if they had to be on. Then if k has to be on (counter<value of k) it will be set on else it will stay off. So now the loop invariant holds for the element k+1