Eindhoven University of Technology 2IO70 DBL Embedded Systems

# Software Design

Aleksandr Popov
0924595
a.popov@student.tue.nl

Devin van Broekhoven
0839004
d.v.broekhoven@student.tue.nl

Jiddo van Vliet
0821894
j.g.v.vliet@student.tue.nl

Yuntao Li
0910663
y.li.2@student.tue.nl

Dan Cristian Chirascu
0923784
d.c.chirascu@student.tue.nl

Group 17

April 14, 2015

Department of Mathematics and Computer Science

## 1 Introduction

This document describes the process of creating Java-syntax pseudocode that implements the functionality and requirements discussed in document Software Specification as a stepping stone for writing PP2 Assembly code for the sorting machine.

## 2 Overview

Our program consists of several classes that mimic the machine interface. Those classes are:

- `Button` — implements all the buttons (Start/Stop, Abort, Dispenser)
- `BWDetector` — implements the black / white detector
- `ConveyorBelt` — mimics the conveyor belt and its motors
- `Detector` — implements the presence detector and two side detectors
- `Dispenser` — mimics the dispenser, including the motor and the button

In addition, there is a class called `Controller`, which actually contains all the code for controlling the system and interacting with the machine interface defined above.

## 3 Explanation

In this section the code explanation is given, that is, the reasoning behind the choices and the actions that are performed.

Our main controller, contained in the function `work`, contains a `switch`-construction running in an infinite loop. The code is implemented as a state machine, therefore we have a description of states, and for every state there are certain actions to be done and certain allowed guarded transitions to other states. The `work` function just checks if any of the buttons have been pressed by the user, recording that information if necessary. It also performs the abort if necessary, therefore satisfying the condition that machine aborts within 15 ms after the button is pressed (see PP2 Assembly code for more details when ready). Then the function checks the current state and runs one of the functions that correspond to the correct state.

Each of the functions performs in constant time, so that machine always stays responsive to user's input and to the detector checks. First come the actions that need to be performed (e.g. using PWM on dispenser motor), then the checks used to decide if the machine should go on a transition to another state.

Our machine has 3 initialisation states — `INIT`, `INIT1` and `INIT2`. They are used to determine the current position of the dispenser and to bring it into a well-defined initial position by means of using PWM on dispenser motor and checking the dispenser button.

The initialisation phase occurs after the user presses the Start/Stop Button for the first time after power-up of the machine or after abort.

After the initialisation machine rests in `READY` state, waiting for the user to press Start/Stop Button. After it is pressed the machine starts sorting discs, if there are any; otherwise it stays in `READY` state.

Then the machine goes to `SP` state, which is only used on its first run after resting, as during later runs conveyor belt and dispenser move simultaneously. When the dispenser reaches the black/white detector, machine goes to `SD` state. In this state the colour of the disc is determined, and the machine then goes to either a series of `SW` states corresponding to white discs or to `SB` states corresponding to black discs.

In `SW1` state the belt starts moving to the left, and the dispenser continues to move in order to push the next disc to the detector. Once the dispenser reaches a certain position, the input value from the presence detector is checked to determine if there are any discs left to sort. If there aren't any or if user has recently pressed the Stop Button, the machine then goes to `SWS` state, where the dispenser stops in the same position as after initialisation. The belt then continues to move either until the 1 second timer runs out or until the disc is detected at the tray. The belt then stops and machine goes to `READY` state if the disc has reached the tray in 1 second or to `INIT` state if an exception has occurred and the disc has not reached the tray on time.

However, if there are more discs left, then the machine goes to `SW2` state instead, where the belt continues to move and the dispenser moves on to deposit the next disc. When the next disc reaches the black/white detector, machine goes to `SW3` state. Both the belt and the dispenser are stopped. By this point the previous disc should have reached the tray, so if it has not, then the machine should abort (i.e. go to `INIT` state), otherwise it should continue sorting discs and go to `SD` state to do that.

States `SB1` – `SB3`, `SBS` perform the same operations, but the conveyor belt moves in the opposite direction to deposit a black disc.

Finally, operations can be aborted at any point by user pressing the Abort button (machine will stop all the motors and go to `INIT` state).

## 4 Motivation

The design of the program as a state machine as explained in the previous section is due to several requirements:

1. Machine should stay responsive to user commands
2. Machine should be able to abort within 15 ms
3. Machine should employ PWM on the motors
4. Two motors should be able to run simultaneously
5. Machine should reliably react to signals from detectors

The given requirements mean that machine cannot employ any continuous loops (i.e. loops that execute until receiving a certain signal from PP2 input), as that would mean that we cannot run two motors simultaneously because they should use PWM, so the output signal should regularly change. Moreover, that would prevent the machine from reacting to user interacting with the machine, e.g. pressing the Abort button.

Basically, there were two main ways to implement required functionality:

- Make it into a state machine
- Write a scheduler for PP2 and several independent programs that would run pseudo-simultaneously operated by the scheduler

The first option was chosen as less error-prone and with less implementation caveats caused by PP2, as the scheduler would have to replace the Monitor program loaded into PP2 at startup. It is also more easily representable in form of high-level code. The term "state machine" here means that the actions are continuously executed in states while constantly checking for conditions that would cause transitions to other states, and transitions occur almost instantly (i.e. within $\sim 1$ ms).

## 5 Informal Correctness Proof

One can clearly see from the code that neither of the functions runs for more than several milliseconds. Main loop gets executed every 15 ms (see Assembly code), so we can be sure that machine stays responsive to user pressing the buttons, and that also guarantees that machine can abort all its operations within 15 ms after the user presses the Abort button. One can also clearly see from the explanation and the code that the initial position of all parts of the machine in every state is well-defined, so one can see that it should perform as intended unless any mechanical problems occur.