
Final Report

25th of March 2015

2IO70

Version 1

This document will contain the documents of the preceding phases and give an introduction and conclusion to the project. "The Final Report presents the reader with a clear picture of the designed machine, the method of working followed, the specification, validation, and design of the software, and a motivation of the main design decisions."
(Source: *Project Guide Design Based Learning "DBL 2IO70" "Sort It Out"*)

Group 16

Rolf Verschuuren

Wigger Boelens

Stefan van den Berg

Dat Phung

Maarten Keet

Tudor Petrescu

Introduction

In this document you will find the details on how we have designed and built, in the past eight weeks, a sorting machine and the software that runs it. This Final Report will contain the five “Product” documents previously handed in and approved by the tutor, and a “Process” document. The five Product documents explain how we arrived at our final design for both the hardware and the software. In order of when we made them, these are “Machine Design”, where we explain how the machine was designed and why we chose to do it that way. Subsequently comes “Software Specification”, where we made a finite state automaton that the software was going to be based on. Then came the “Software Design” and “Software Implementation and Integration” documents in which we first designed the full program in pseudo-Java code and then subsequently translated this into working Assembly code. Throughout this document there are validation segments in which we explain how we validated our decisions. In the “Validation and Testing” document we look back at these segments and describe the measures we took to ensure that our final product would meet the initial requirements. The second part of the Final Report is the Process document, in this document we describe how we worked as a group over the course of this project, and how we decided to tackle any issues that arose. This Final Report is the final deliverable for the course.

We begin the document with our product and start with “Machine Design”. Next, we give a description of how the PP2 should behave in the “Software Specification” segment. Thereafter, “Software Design” will complement the Final Report. Then the designed program is translated into Assembly in “Software Implementation and Integration”. Last but not least, our product is to be validated. The validation is scattered throughout the document, which will be explained at the end in “System Validation and Testing”.

After the “Product”, we describe our process as a group in “Process”. Finally, we end this document with a conclusion. Included at the end as appendices are a model of the finite-state automaton, made in UPPAAL, a table of the display of states, the Java program, an explanation of the PHP to Assembly compiler and the functions of the compiler, the PHP program, and the Assembly code.

Table of Contents

<i>Introduction.....</i>	<i>2</i>
<i>Product.....</i>	<i>6</i>
Machine Design	6
High level Specification	6
Priorities.....	7
USE-cases.....	8
Starting the machine.....	8
Stop the machine.....	8
Sort unsorted disks	9
Abort the process.....	9
Booting of the machine.....	10
Shutting down the machine	10
User Constraints	11
Safety Properties	11
Explanation of Safety Properties.....	11
Design Decisions.....	12
The Feeder	12
The Transportation and Scanning.....	13
The sorting mechanism	15
Machine interface.....	16
The feeder	16
The position sensor	16
The black white detector.....	16
The Sorter.....	17
The buttons	17
The conveyer belt.....	17
I/O tables	18
Outputs	18
Inputs	18
Picture.....	19
System Validation and Testing	20
Validate High level specifications	20
Validation System Level Requirements	20
Validation Priorities to SLRs.....	21
Testing machine design to the priorities	22
Software Specification	23
Inputs and Outputs.....	23
Inputs	23
Outputs	24
Relations	24
Lens lamp of the black white detector	24
Lens lamp of the position sensor	24
Engine of the conveyor belt	25
Engine of the feeder.....	25

Engine for the sorter	25
Display for the state	25
Design Decisions.....	25
Feeder	25
Lens lamp position	25
Conveyor belt	25
Lens lamp colour	25
Push button.....	26
Description of States	26
Initial_state	26
Calibrate_Sorter	26
Resting_state.....	26
Running_state	27
Running_Wait.....	27
Running_Timer_Reset	27
Motor_Up.....	28
Motor_Up_Stop	28
Motor_Down.....	28
Motor_Down_Stop	29
White_Wait	29
White_Wait_Stop.....	29
Running_Timer.....	30
Motor_Up_Timer	30
White_Wait_Timer.....	30
Motor_Down_Timer	31
Aborted	31
Running_Stop	31
State transitions.....	32
Finite-state Automaton.....	33
Tests done.....	33
Validation of “Inputs and Outputs”	34
Validation of “Relations”	34
Validation of “Description of States”	34
Starting the machine.....	34
Stopping the machine.....	35
Sort unsorted discs	36
Abort the process.....	37
Validation of “State Transitions”	38
Validation of “Finite-state Automaton”	38
Validation of “UPPAAL model”	38
Software Design.....	39
Coding Standards	39
Translating to pseudo java:	39
Translating from Java to PHP.....	39
Design decisions for the Java code	40
Validation.....	41
Validation of java to transition table.....	41
Validation of timerManage	41
Control flow validation.....	41

Software Implementation and Integration	42
Java to PHP	42
Example of Java to PHP	42
Example of PHP to Assembly	42
Java to PHP	44
Validation of Java to PHP	44
System Validation and Testing	45
Validation Policy	45
Validating the machine to the priorities.....	45
Validating the machine to the USE-cases.....	46
Conclusion	46
Work Plan.....	47
Workday.....	47
Problems	47
Conclusion	48
Appendix 1: UPPAAL model.....	49
Appendix 2: Java Program	50
Appendix 3: Explanation of the compiler.....	54
Preprocessing	54
Splitting	54
Compiling	54
Combining	55
Formatting.....	55
Appendix 4: Explanation of the compiler functions.....	56
Appendix 5: PHP Program	60
Appendix 6: Assembly Program	64

Product

Machine Design

In this phase, we explain the design of our machine and how we decided on this design, and why we decided on this design. To do this we will take a look at our requirements and priorities. Afterwards we will look at the design and the decisions leading to that design.

High level Specification

The specification as given in the Technical Guide

The goal of this project is to build a simple sorting machine that is able to separate small objects, plastic discs that may be either black or white, into two sets: the black discs and the white discs. (...) The machine must contain at least one **conveyor belt**.

(...)

The machine is to be operated by means of two push buttons, called **“START/STOP”** and **“ABORT”** (...) By pressing button **“START/STOP”** the machine is started. (...) If 4 seconds after (...) expected arrival time the presence detector has not signalled the arrival of a disc (...) the machine stops (...) If, during the sorting process. The push button **“START/STOP”** is pressed the machine (...) continues its normal operation until the current disc has been deposited into the correct tray. Then, the machine stops. (...) Push button **“ABORT”** (...) makes the machine halt immediately. (...) Pressing this button while the machine is in its resting state has no effect. (...) If subsequently, the push button **“START/STOP”** is pressed once, the machine returns to its resting state.

To be able to guarantee that the mechanism depositing discs onto the **conveyor belt** stops in a well-defined state, this mechanism must be equipped with (at least) one **switch** to signal that this mechanism has reached the correct state.

Our specification

We have to make a so-called sorting machine. This machine should be able to separate, by colour, small black and white plastic discs. The requirements are as follows, the machine should:

- Have at least one conveyor belt.
- Have two buttons called **“START/STOP”** and **“ABORT”**.
- Start when the machine is in a resting state and **“START/STOP”** is pressed.
- Stop when the machine is running and **“START/STOP”** is pressed, before stopping it should sort all discs that are on the belt.
- Abort when **“ABORT”** is pressed, this should halt the machine immediately unless it's in the resting state.
- Go to a resting state when the machine is in a halting state and **“START/STOP”** is pressed.
- Have at least one switch to signal when the machine is in a resting state.

Priorities

1. We define reliability as the ability of the machine to correctly sort all the inputted disks. We validate the reliability of the machine by checking the correctness of the code running the machine and also by conducting long-term test. Reliability is mainly reflected in our decision to encase the conveyer belt so that it is prevented any possibility of the discs, that are transported, to slip out. The goal of the project cannot be met with an unreliable design.
2. The speed of the machine is defined by the number of disks sorted in a unit of time. We search to select the design solution that improves this number. Speed is essential to offer a pleasant experience operating the machine. Speed is also the first thing that stands out when two machines of this sort are compared.
3. We define robustness as the fact that the machine does not break easily. The validation is if the machines state wouldn't be changed, they wouldn't break during: build phase, test phases, simulations, transportation and the end process, all during the period of the project cycle. Then we can consider the machine to be robust. Robustness can be observed from our design solution from the partial encasing used. Also the disc container was design to be robust do to its shape, size and simplicity. We do not meet our project goal if the machine isn't capable of running during the final process.
4. We define user accessibility as the ease in which the user takes the actions required from the machine. Validation is done by checking the compatibility of the design and the user constrains. The disc container was built with user accessibility in mind, it is fairly easy and fast to load discs. The reason why this priority is important is that the machine requires a user to be operated and in consequence its operation must be possible.
5. We define amount of space by the amount of floor space that the machine occupies. Checking if there are useless components in the machine or other components that can be replaced with smaller counterparts without influencing the priorities above does validation of the low amount of space. From this perspective the current Feeder occupies a small amount a space, while the other feeder design would of forced us to add an extra floor extension because of its large dimensions. The reason of this priority is to ease the transportation and storage of the machine.
6. The Difficulty of Building is self-explanatory. We validate this be checking if there are any useless components. In our decision to have the conveyer belt larger, trying to fit on the platform size, we simplified the design and left more physical space to work on the other components connected to the machine. Opting for such a priority would make our solution easy to implement.
7. The Amount of Parts of the Machine is also self-explanatory. We also check if there are any useless parts. An example were we used very little parts by choice in our machine is the feeder component. Reasons why we picked this priority is that it might improve the overview of the machine and also the error-detection.

For the validation of these priorities see "Testing machine design to the priorities".

System Level requirements

The system level requirements consist of 3 parts. These 3 parts are the USE-cases, the safety properties and the user constraints.

USE-cases

There are 6 USE-cases, which are described below.

Starting the machine

Primary Actor	Machine operator (student or teacher at TU/e)
Scope	A sorting machine
Brief	The machine operator starts the machine, machine parts go to their initial state and the machine starts sorting.
Postconditions	The machine starts the sorting process.
Preconditions	-
Trigger	Booting the machine / finished the abort or start/stop routine
Basic Flow:	<ol style="list-style-type: none">1. Machine puts devices in their initial state.2. The user presses the START/STOP button

Stop the machine

Primary Actor	Machine operator (student or teacher at TU/e)
Scope	A sorting machine
Brief	The machine is waiting for the current process to end before it is send into an inactive state.
Postconditions	The machine is sent into an inactive state with no process interrupted.
Preconditions	The machine is running.
Trigger	The START/STOP button is pressed.
Basic Flow:	<ol style="list-style-type: none">1. The machine finishes sorting the disks currently in the machine2. The machine enters an inactive state and will not take any more disks form the storage* unless the START/STOP button is pressed

Sort unsorted disks

Primary Actor	Machine operator (student or teacher at TU/e)
Scope	A sorting machine
Brief	The machine sorts the unsorted disks provided into two separate containers based on colour.
Postconditions	There are no unsorted disks left All sorted disks are in a container based on their colour
Preconditions	The machine is not already running.
Trigger	The user provides unsorted disks and presses the “START” button.
Basic Flow:	<ol style="list-style-type: none">1. An unsorted disk is moved to the colour detector2. The machine decides to which of the two containers the disk needs to be moved3. The machine moves the disk to the designated container4. The machine repeats step 2 through 4 until all disks have been sorted5. The machine pauses within 4 seconds

Abort the process

Primary Actor	Machine operator (student or teacher at TU/e)
Scope	A sorting machine
Brief	The machine should immediately stop doing anything.
Postconditions	The machine stopped running and is ready to start again.
Preconditions	The machine is sorting discs.
Trigger	The use wants to immediately stop the machine.
Basic Flow:	<ol style="list-style-type: none">1. The machine stops transporting the discs. And doesn't put any more discs on the transporting mechanism.2. The user is required to remove all discs that are neither in the container unit nor sorted.3. When the user removed all unsorted discs that were not in the container unit he presses the START/STOP button.

Booting of the machine

Primary Actor	Machine operator (student or teacher at TU/e)
Scope	A sorting machine
Brief	The machine will prepare to start the program. And do the required actions.
Postconditions	The machine is ready to get instructions of the user.
Preconditions	The machine is off.
Trigger	N/a
Basic Flow:	<ol style="list-style-type: none">1. Connect the PP2-board to the pc.2. Plug the pp2-board in to the power socket.3. Start the debugger4. Connect the pp2-board using the debugger.5. Load the program into the debugger.6. Run the program.

Shutting down the machine

Primary Actor	Machine operator (student or teacher at TU/e)
Scope	A sorting machine
Brief	User unplugs the power supply and disconnects the processor from the PC and the machine.
Post conditions	The PC can be used for other things and the processor and machine can be stored separately.
Preconditions	Everything is in its initial state or the machine has stopped.
Trigger	N/a
Basic Flow:	<ol style="list-style-type: none">1. Unplug the power supply of the machine.2. Unplug the power supply of the processor.3. Disconnect the processor from the machine.4. Disconnect the PC from the processor.

User Constraints

- Before the start button is pressed, the user is required to place all discs to be sorted in the container unit
- While the machine is running the user is not allowed to move the machine or touch anything except the buttons.
- When the abort button is pressed or the machine has been shut down, the user is required to remove all discs that are neither in the container unit nor sorted.

Safety Properties

1. After pressing an emergency button, within 50ms there should be no moving part in the machine
2. If all disks are sorted the machine should stop within 4 seconds.
3. After the start-up of the machine, the assembly program should not stop until the machine is shut down.
4. The outputs connected to the h-bridge may never be powered on at the same time.
5. The outputs connected to the motors should never output more than 9 volts

Explanation of Safety Properties

1. When there is an emergency it is important that whatever is going wrong will not get worse. One of the ways this can happen is for instance that someone's finger gets stuck, to minimize damage to this finger the machine should stop quite fast. After discussion we decided 50ms would be a reasonable maximum stop time as it whatever is going wrong will not get worse in 50ms.
2. To minimize electricity usage we think that the machine should not keep running while there are no disks in it.
3. If the assembly program stops while the machine is still running, we can no longer control the machine. We can for instance no longer detect when the emergency button is pressed, meaning we cannot guarantee safety property #1.
4. The H-bridge should never have two inputs powered on at the same time. Because then you create a short circuit.
5. According to the project guide this is the maximum voltage the motors are certified to work with.

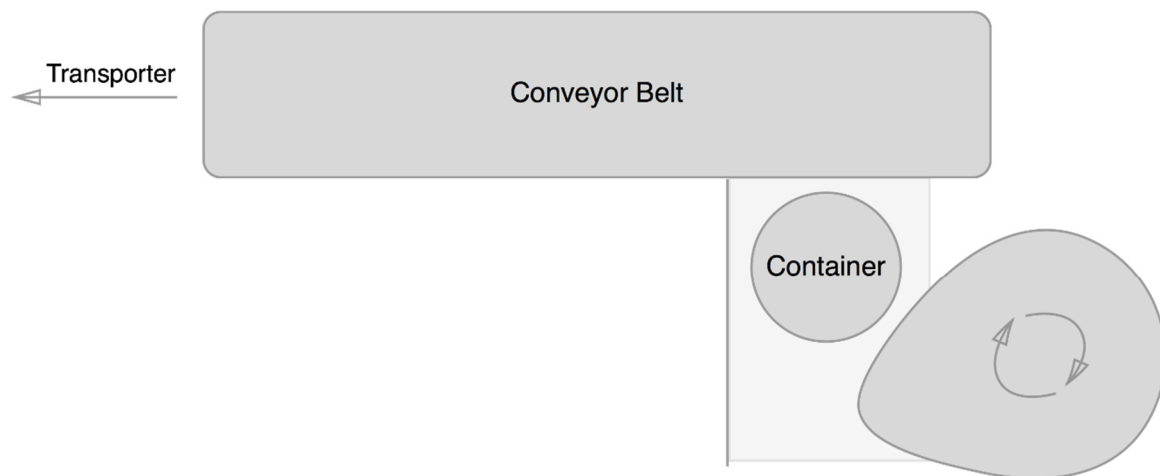
Design Decisions

The way we approached the design of the machine is by separating the machine into multiple parts. Those parts exist out of: the feeder, the transportation mechanism, and the sorter.

The Feeder

The feeder has as objective that it needs to somehow get the disks from the container onto the conveyor belt. This is needed for the use case “Sort unsorted disks”.

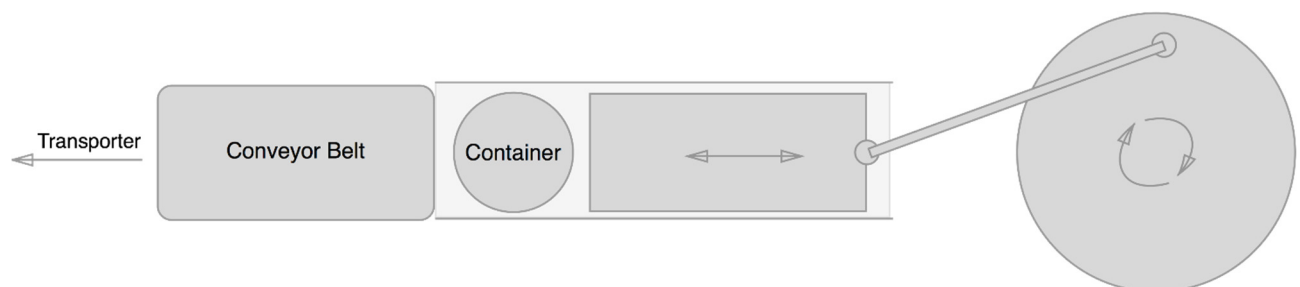
For the design of this feeder we had two competing designs. Both use the two hollow tubes stacked as a container. We chose to do this because they are completely reliable in containing the disks and because a new disk simply falls out if the bottom one is removed, they are very fast. Because the container is made off two big parts and some small parts to make them stack, the container is also very robust. It's quite easy to put the disks into the big hole at the top, so user accessibility was very high. In short, the first solution that came to mind scored extremely high on all priorities and we looked no further.



The first design for the feeder consist of 3 important parts. First you have the container. The container drops a disk, which is then pushed onto the conveyor belt using a cam. A wall to the left of the container makes sure the disk is pushed up and not to the left.

Our second feeder design also consisted of a block that pushes the disk. To make this block move a lever attached to a wheel is used. Rotating the wheel makes the block move back and forth, pushing disks onto the conveyor belt.

Both designs correctly implemented the use cases. To test which one would be better we



build both and tested them. They scored the same on almost all top priorities. They were

both completely reliable for instance. There was also no difference in speed, both would push a disk onto the conveyor belt with every turn of their wheels. Both did not hinder the user, so the good user accessibility of the container was unchanged. When we came to the last three priorities there were some differences making us choose the first design: It was easier to build, used less parts and was a lot more compact.

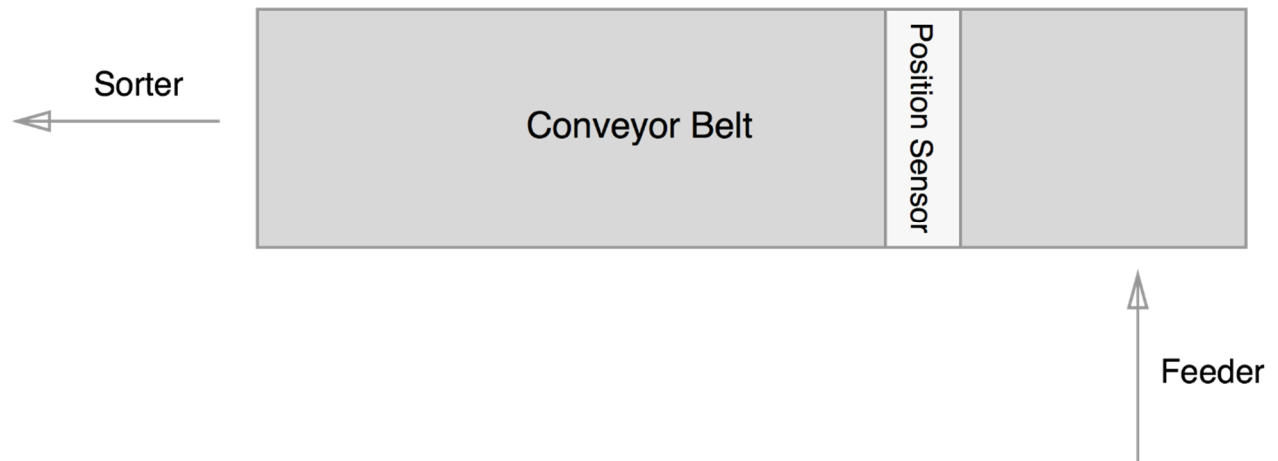
The Transportation and Scanning

When considering the transportation method we had a 3 main ideas. The first one was that we used a short conveyor belt. The second idea was about a long conveyor belt. And the last idea used a turning wheel and 2 conveyor belts. All these ideas included a conveyor belt because that was required.

The thought behind the short conveyor belt was that in the feeding mechanism would push the discs hard enough so that we could put the sensors on that part and to have a small but conveyor belt to transport the discs. The conveyor was short because nothing needed to happen on it. Thus it would only be there because it was a requirement. To us it seemed a bit useless to not do anything on the conveyors belts. So that was when the second arose.

The second idea had a long conveyor belt to put the sensors on. And also a part of the separating mechanism. The conveyor belt would limit how fast the machine can run but all the actions would happen on the conveyor belt so that time wouldn't be wasted. It also isn't that hard to create a long conveyor belt so we kept the idea in mind.

Our final idea was that there would be some sort of wheel with separate compartments for discs in the centre which would rotate and put discs on to two different conveyor belts. Each conveyor belt led to a storage unit of the sorted discs. The problem with this idea was that it would be hard to prevent the discs from spinning out of the compartments when they shouldn't while still being able to let the discs go out when they had to. Because we couldn't get it to work the idea was dropped and we went back to the idea about a long conveyor belt.



We were capable of realizing the of the long conveyor belt. But during the build of the conveyor belt we noticed that it would not be tight enough around the gears. Thus we tried to remove a small part of the belt. But this still didn't have to effect we hoped for. So we added a third gear in the middle which tightened the belt to an acceptable state.

The conveyor belt was still far from perfect because it would tilt at certain points and the discs could fall off. So to prevent it we build 2 walls around the belt. On the first part they are low because the low walls were more robust than the high walls and for the user it is easier to access the discs on the conveyor belt. The high walls have been secured using 4 pillars because that made it robust enough to make sure they didn't break. The walls had to be high because we needed to put a set of sensors on it.

Those sensor had to be above the conveyor belt. They also needed to be at an angle to work properly. That was required else the sensor wouldn't be able to check if the disc was black or white.

The other set of sensors didn't need to be place at an angle thus they were simply put on each side of the conveyor belt. This set of sensor would then be capable to scan if there was a disc on that spot of the conveyor belt. This sensor is need to time at which moment the other set of sensor had to check the colour of the disc. And it is also used to check if there are any more discs left to scan.

The sorting mechanism

For the mechanism that does the actual sorting we chose between a couple of different designs. These designs are listed and explained below.

The first, and most simple design was to use just one conveyor belt that would move left or right based on the colour of the disks. This design is listed under the use of the conveyor belt above, this is why I will not describe it again.

The second design is a slight improvement on the first one where we would use a second, shorter, conveyor belt to do the sorting. This design would place the two conveyor belts in a T-shape with the colour check done on the first one, after which the second conveyor belt moves left or right. We considered this design an improvement on the first one because the second conveyor belt could be made much shorter. This means that the design can sort faster than the single conveyor belt one.

The second conveyor belt was faster than the first design with only one belt, however we soon realized that we could do this even faster. By removing the second belt and replacing it with a seesaw that could be angled to face one of the two sorted containers, we could increase the speed even more. Since the disk would essentially be sorted the moment it reached the end of the conveyor belt. This would be a great design, was it not for the fact that the seesaw required a lot of height. In fact, the entire machine looked like it was placed on stilts, requiring us to use lots of parts and having a lot of wasted space underneath. This design could do it faster at the cost of requiring more space than any of the others.

While the use of a seesaw sped up the sorting process, it also took a lot more space, so we went back to the drawing board and discarded this idea. Instead coming up with a wedge that would be slide onto the conveyor belt from the side whenever a disk of a certain colour is detected. This would then allow the conveyor belt to push the disk against the wedge making a roughly 45° angle thus pushing the disk of the side of the belt and into the collection box. The second colour could just continue while the wedge was pulled back and off the end of the belt. This means that the design cuts off part of the machine at the end and allowing us to make the machine lower than before.

We liked the idea of letting the conveyor belt doing the sorting by placing a wedge in the way, but after some thinking we realized that it could be done both faster and more compact. The trick was to change the direction in the wedge moves from horizontal to vertical. Doing so moves the entire mechanism, aside from the wedge itself, in an upright position pushing it very close to the machine. Aside from saving space, this also allowed the wedge to move much less, since it only has to move just over 1cm above the conveyor belt rather than move all the way over it to the side. This final design does not sacrifice any reliability from its predecessors while being the fastest. It also takes by far the lowest amount of floor space, characterized by the fact that this final design including this sorting mechanism is our only design that fits on only one of the two provided floor plates. For these reasons we believe this design for the sorting mechanism to be the best.

Machine interface

The feeder

The motor for the feeder turns a clam. With that motor turning clockwise the disc, which is on the surface in front of the clam, will be pushed off the surface and on to the conveyor belt. To make sure the engine runs clockwise the minus has to be connected to the connection closest to the spot where 6V is marked. We connect this engine to the 3rd output of the pp2-processor.

The position sensor

The way a position sensor is set up is by using a lens lamp and a phototransistor. The lens lamp will be shining in the direction of the phototransistor. The light from the lens lamp makes the phototransistor send a signal to the pp2-processor. If a disc comes in between the lens lamp and the phototransistor then there won't shine any light at the phototransistor and thus it won't send a signal to the pp2-processor. The phototransistor is connected to the 8th input of the pp2-board. The phototransistor is polarized and thus it is important that it is connected correctly. The correct way to connect is with the ground to the connection closest to the white spot on the phototransistor. The lens lamp isn't polarized and does not move in any direction and thus it doesn't matter in which connection the ground is. The lens lamp is connected to the 2nd output of the pp2-processor.

The black white detector

The black white detector uses the same components as the position sensor but they are implemented in a different way. The way in which the colour is detected is by the reflection of light on the disc. Because white discs reflect light very well the phototransistor does pick up some light and thus sends a signal. Black disc on the other hand do not reflect enough light to let the phototransistor pick it up. Thus a white disc can be detected if the sensors are placed in the correct way.

To make sure the phototransistor picks up only the reflected light a cap is placed over it with a hole in the middle. So only light from in front of it will influence the phototransistor. But to make sure that the reflected light can pass through that hole the sensor must be placed at an angle. The reflected light, which is detected by the phototransistor, is at its strongest when the lens lamp is also placed at an angle.

We connected the lens lamp in the same way as the lens lamp of the position sensor only now to the 6th output of the pp2-processor. The phototransistor is also connected as described in the position sensor only now to the 3rd inputs.

The sorter

The divider uses a so-called “H-bridge” to move up and down. We use output 0 and output 1 to control the H-bridge, which in turn controls the motor moving the divider. We connect the ground of the H bridge with the output 0 to the 6-side of the motor. Now when we power up output 0 the divider will move up. When we power up output 1 the divider will move down. Output 0 and output 1 are never allowed to be on at the same time, which is also stated in the safety properties. We want to move the divider as fast as possible so we always use the maximum allowed voltage of 9 volts. To detect when the divider is in its upmost position we use a push sensor. When the PP2 detects that this push sensor is pressed we immediately cut the power to output 0. We do not detect when the divider is at the bottom, because as soon as the push sensor is not pressed then there isn't enough space for a disc to go underneath. Thus we simply power on the motor for a set amount of time. This time should be enough to make it move to the bottom but not low enough to interfere with the conveyer belt.

The buttons

The button that is used to start/stop the machine will be button 0. The button to abort the machine will be button 1.

The conveyer belt

The conveyer belt uses 5 gears of which only 3 touch the conveyer belt. 2 of those 3 gears are used to make sure the conveyer belt is horizontal and the third one is used to make the conveyer belt turn. The third gear is connected to a metal rod. On that metal rod another gear is connected and that gear will be turned using the gear which is connected to the engine. Because we have those gears in between the direction in which the engine turns has to be counter clockwise. Then the conveyer belt does turn clockwise and the discs will be moved in the right direction. To let the engine turn clockwise we have to connect the ground to the connection closest to the 9V. This engine is connected to the 3rd output.

I/O tables

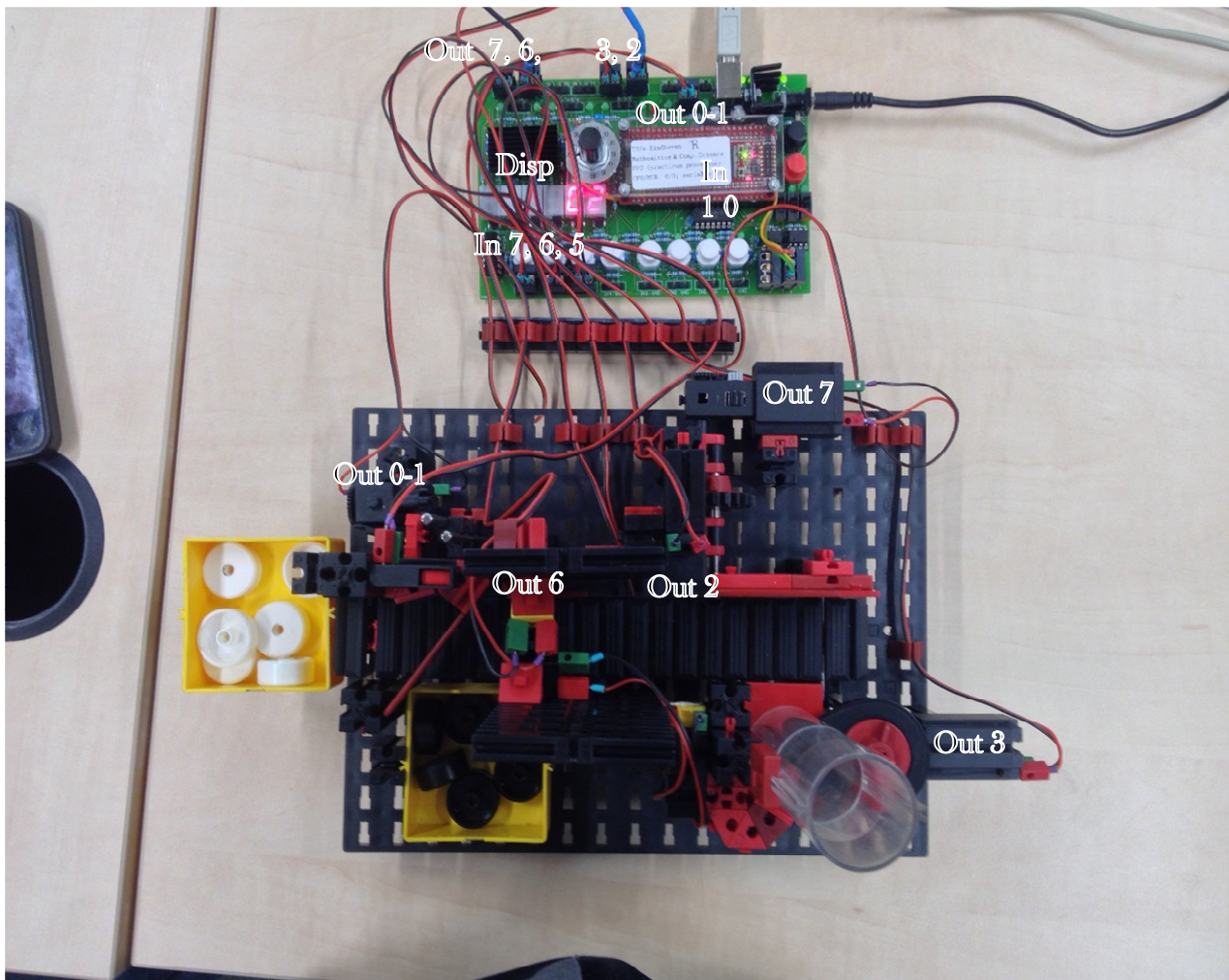
Outputs

Output	The range/type of the value
Start/Stop button	Boolean value
Abort button	Boolean value
Push button(sensor)	Boolean value
Colour detector	Boolean value
Position detector	Boolean value
Timer	Values range from seconds to clock ticks

Inputs

Input	The range/type of the value
Lens lamp 1	Boolean value
Lens lamp 2	Boolean value
Conveyer engine	Between 6 and 9 V (Volts) while running 0 V when not running
Feeder engine	Between 3 and 7 V while running 0 V when not running
Sorter engine	Between 6 and 9 V while running 0 V when not running
Display	Integer value, positive

Picture



System Validation and Testing

Validate High level specifications

Our high level specifications are correct, because in the exercise it is said that a sorting machine for black and white discs should be made. And it also is said that we need at least one conveyor belt.

Validation System Level Requirements

The high level specification defines the basic flow of the use-cases, user constraints and safety properties. At the same time, we validate the System Level Requirements through the high level specification. “Sort unsorted discs” is correct, because the high level specification mentions that the machine should sort discs. Aborting the process happens because in every machine something could go wrong and thus it needs to be able to be stopped at any point in time. “Starting the machine” and “Stopping the machine” are actions which are also needed for machines because else you couldn’t make them stop or start doing what they are supposed to do. “Booting up the machine” and “shutting down the machine” is required, because the disc sorter has to be turned on and off, in order for it to fulfil its purpose.

Before the start button is pressed the user is required to place all discs to be sorted in the container unit. The discs should be placed in the container, so that the machine is able to sort the discs.

While the machine is running the user is not allowed to move the machine or touch anything except the buttons. If the user makes contact with either the conveyor belt or the discs while they’re on the conveyor belt, the machine might not be able to separate the discs correctly.

When the abort button is pressed or the machine has to be shut down, the user is required to remove all discs that are neither in the container unit nor sorted. The user is supposed to do this, so that the machine will be able to restart the sorting process with a new disc.

After pressing an emergency button, within 50 ms there should be no moving parts in the machine. The machine should immediately abort its current process, according to the high level specification, although this is not realisable. Therefore, this is set to be within 50 ms.

According to the High level Specification the machine should stop sorting if there is no more disk signalled after 4s. We made this into a safety property, because a running machine with no use is only going to possibly harm people getting in contact or the machine itself.

According to what the high level specification offer, there is nothing that could stop the assembly program as long as the code is correctly written for this purpose, we don’t consider accidents and flaws, the only way for the program to end is by powering off the machine.

The outputs connected to the h-bridge may never be powered on at the same time. If this happens, the PP2 processor short circuit, and the machine won’t work anymore.

Validation Priorities to SLRs

Reliability:

The use-cases describe how we want to sort multiple coloured disks, because we want the sorting to be done as accurately as possible we chose reliability as one of our priorities.

Accessibility:

The use-cases describe that the user has to remove all disks from the machine after the “ABORT” button is pressed. Because of this we want to make the machine somewhat open, so the user can remove the disks with relative ease.

Speed:

The use-cases describe how we want to sort multiple coloured disks, because we want the sorting to be done as fast as possible we chose speed as one of our priorities.

Robustness:

The use-cases describe that the user has to remove all disks from the machine after the “ABORT” button is pressed. For this reason we want the machine to be fairly durable so that the user does not easily damage it. Additionally, since the machine contains a number of engines and moving parts, it will be vibrating ever so slightly. These vibrations should also not cause any damage to the machine leading to our priority of robustness.

Amount of space:

This priority does not have a clear relation to our SLRs, however, we believe that a small machine capable of accomplishing the same task is generally better than a larger version. This is because the machine has to be stored or placed somewhere, leaving you with more space for other machines. This is why we chose for minimizing floor space as one of our priorities.

Difficulty of building:

This priority also does not have a clear relation to our SLRs, but this would make our job as builders easier. It would also allow for greater rates of production of the machine. For these reasons we chose difficulty of building as one of our priorities.

Amount of parts:

This priority also does not have a clear relation to our SLRs. A lot of parts, though, would make our machine more expensive and harsher on the environment, leading us to make the amount of parts one of our priorities.

Because the priorities “Amount of space”, “Difficulty of building” and “Amount of parts” have no clear relationship to the SLRs we chose to put them on the bottom of our priority list.

Testing machine design to the priorities

1. Perform a test with alternating black and white discs to test the moving of the divider multiple times and check that the discs are sorted right and all discs were sorted.
2. Check if it sorts 10 discs within 30s with a load of white discs, black discs and alternating black and white discs
3. Let the machine perform a run without pushing buttons and with pushing the abort button while running and check if nothing breaks.
4. Look at points in the machine where a disc could get stuck and check if you can access the disc to remove it.
5. Check if the machine fits on 1 floorboard of the Fischer Technik.
6. Check if you can build the machine within 1.5 hours with 2 people.
7. Check if there are any parts without a function.

Software Specification

In the Software Specification phase, we give an as accurately as possible description of the required behaviour of the PP2, without describing how this is achieved, and a UPPAAL model of this behaviour. In order to do this, we translate the system level requirements to a high level specification of what the software controlling the physical machine should do.

Inputs and Outputs

Inputs

Input	The range/type of the value	Abbreviation
Start/Stop button	Boolean value	In 0
Abort button	Boolean value	In 1
Push button(sensor)	Boolean value	In 5
Colour detector	Boolean value	In 6
Position detector	Boolean value	In 7
Timer	Values range from seconds to clock ticks	Tim

The **Start/Stop** and **Abort buttons** speak for themselves. They are either pressed or not pressed.

Push button(sensor): the sorter touches the push sensor or doesn't touch it, to detect the sorter's position.

The **position sensor** and **colour detector** are either on or off.

The **timer** is a count-down timer that is set to a certain value and runs at a frequency of 10 kHz. All given times were calculated by taking the average time of ten measurements, using 50 to 60% of the Potentiometer on the PP2 board. Thus, the sorting mechanisms are faster in reality. The input of a timer is set to a defined value or not set.

TEnd is the moment of termination of the timer, so when the timer reaches zero.

Motor Down is defined as the time it takes for the engine of the sorter to move the sorter from the lowest point to the highest point, until sorting mechanism touches the push sensor. This takes 0.30 seconds.

Motor Up is the state of the sorter moving from the highest point to the bottom of the engine sorter. Since the engine sorter for Motor Down and Motor Up have the same voltage, this will take 0.30 seconds as well.

Sort is the amount of time it takes for a disc to be transported from the black/white detector to the end of the conveyor belt, which is measured to be 0.85 seconds.

Belt is the period that a disc travels from the feeder to the end of the conveyor belt, until the disc reaches the tray for black discs. This action takes 2.0 seconds.

Tic is defined as one clock tick of the PP2. A clock tick is incredibly fast.

Outputs

Output	The range/type of the value	Abbreviation
Lens lamp 1	Boolean value	Out 2
Lens lamp 2	Boolean value	Out 6
Conveyer engine	Between 6 and 9 V (Volts) while running 0 V when not running	Out 7
Feeder engine	Between 3 and 7 V while running 0 V when not running	Out 3
Sorter engine	Between 6 and 9 V while running 0 V when not running	Out 0-1
Display	Integer value, positive	Disp

Lens lamp position and **lens lamp sorter** are the lamps that make up part of the sensors and can be turned on or off.

The **conveyor and feeder engines** respectively move the conveyor belt and the feeder. They are either on or off.

Hbridge0 indicates whether the sorter moves up or not. On the other hand, whereas **Hbridge1** shows that the sorter moves down or halts.

The **display** shows the state that the machine is currently in. Depending on the available time, we might or might not implement this.

The **Timer start** output is the same as the Timer input, except that the timer counts down.

Relations

Lens lamp of the black white detector

The lens lamp of the black white detector will be on when the machine is sorting. Thus the lens lamp will react to the input of the “START/STOP” button and the “ABORT” button. The lens lamp will go on when the machine is in resting state and the “START/STOP” button is pressed and it will go off when the “ABORT” button is pressed while the machine was running.

Lens lamp of the position sensor

The lens lamp of the position sensor reacts only to the “START/STOP” button and the “ABORT” button. The lens lamp will be on after the “START/STOP” button is pressed and the machine is in its resting state. If at any other point in time the “ABORT” button is pressed it will go off. When the “START/STOP” button is pressed and the machine is running then the lens lamp also goes off.

Engine of the conveyor belt

The engine of on the conveyer belt only reacts to the input of the “START/STOP” button and the “ABORT” button. The engine will start then the machine is in its resting state and the “START/STOP” button is pressed. If however the “START/STOP” button is pressed and the machine is not in its resting state then the machine will stop after it completed its current cycle. Whenever the “ABORT” button is pressed the engine stops within 50ms.

Engine of the feeder

The engine for the feeder also only reacts to the input of the “START/STOP” button and the “ABORT” button. This engine also starts when the machine is in tis resting state and the “START/STOP” button is pressed. If however the machine is running then the engine will stop. When the “ABORT” button is pressed the engine stops within 50ms.

Engine for the sorter

When the machine is running the engine of the sorter reacts to inputs of the colour detector, the push sensor and the timer. When a signal is received from the colour detector the engine pushes the sorter up, the engine then waits until the timer gives a signal to go down again after it let the discs through, it knows when it is in the correct “up” position from the push sensor . If the “START/STOP” button is pressed when the machine is in its resting state, then the sorter will wait for a signal from the timer that marks the end of the current cycle. If at any time the ““ABORT”” button is pressed, the sorting mechanism is to stop within 50ms.

Display for the state

The display output depends on what state we are currently in. The corresponding state to a number can be found in appendix 2 .

Design Decisions

Feeder

The feeder in constantly on because of priority 2, speed, mentioned in the Machine Design document. Another reason is that there's a turning part that needs to spin through to get to its initial position to be able to deposit discs again.

Lens lamp position

We chose to have the lens lamp for position sensor constantly on, because it's easier to code resulting in spending less time on it. The optimization is minimal if we would turn them off every time there's a gap between discs, because of the feeder being quite fast in depositing the next disc.

Conveyor belt

The conveyor belt is constantly running, because the feeder is constantly pushing discs onto the conveyor belt. This goes hand in hand with our second priority, which is speed.

Lens lamp colour

Like with the position sensor, it's easier to code that it is continuously on. The light being off if it's possible, would again be a minimal improvement, because the gaps between discs being pushed on the conveyor belt is the same as with the black white detector.

Push button

We use the push button, because of priority 1, correctness, to know if the sorter arm is at its highest point. We need to know this, because we need to know when to stop the motor making the sorter arm going up.

Description of States

Initial_state

In the initial state the machine starts calibrating the sorting mechanism by moving it up.

Outputs	Value for output
Lens lamp position	0
Lens lamp sorter	0
Engine conveyor	0
Engine feeder	0
Hbridge0	0
Hbridge1	0
Display	0
Timer start	0

Calibrate_Sorter

In the calibrate sorter state the sorting mechanism moves down until it is just above the conveyor belt.

Outputs	Value for output
Lens lamp position	0
Lens lamp sorter	0
Engine conveyor	0
Engine feeder	0
Hbridge0	0
Hbridge1	1
Display	1
Timer start	0

Resting_state

In the resting state the sorting machine is at rest and waiting for the user to press the START/STOP button.

Outputs	Value for output
Lens lamp	0
Lens lamp	0
Engine conveyor	0
Engine feeder	0
Hbridge0	0
Hbridge1	0
Display	2
Timer start	0

Running_state

In the running state the sorting mechanism, the conveyor belt, the position detector, and the colour detector are turned on.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	0
Hbridge1	0
Display	3
Timer start	2 s + Belt

Running_Wait

In this state a disc has been detected and that disc is moving along the conveyor belt to the sorter.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	0
Hbridge1	0
Display	4
Timer start	2 s + Belt

Running_Timer_Reset

In this state a new disc was detected and the timer has been reset.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	0
Hbridge1	0
Display	5
Timer start	2 s + Belt

Motor_Up

In this state the motor of the sorter is moving up until it hits the push button.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	1
Hbridge1	0
Display	6
Timer start	Sort

Motor_Up_Stop

In this state the motor of the sorter is moving up until it hits the push button. And the machine has to stop because the start stop button was pressed.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	1
Hbridge1	0
Display	14
Timer start	Sort

Motor_Down

In the Motor_Down state, the sorter is moved down.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	0
Hbridge1	1
Display	8
Timer start	0

Motor_Down_Stop

In Motor_Down_Stop, the sorter is moved down, after the start/stop button has been pressed.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	0
Hbridge1	1
Display	16
Timer start	0

White_Wait

In this state the machine waits until the colour detector has detected a white disc.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	0
Hbridge1	0
Display	7
Timer start	Sort

White_Wait_Stop

In this state the machine waits until the colour detector has detected a white disc, after the START/STOP button has been pressed.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	1
Hbridge0	0
Hbridge1	0
Display	15
Timer start	Sort

Running_Timer

Running_Timer is the state that sets the interrupt timer to make sure the machine stops after the current cycle.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	0
Hbridge0	0
Hbridge1	0
Display	9
Timer start	Belt

Motor_Up_Timer

Motor_Up_Timer is the state that sets the interrupt timer to make sure the machine stops after the current cycle.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	0
Hbridge0	0
Hbridge1	0
Display	10
Timer start	Belt

White_Wait_Timer

White_Wait_Timer is the state that sets the interrupt timer to make sure the machine stops after the current cycle.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	0
Hbridge0	0
Hbridge1	0
Display	11
Timer start	Belt

Motor_Down_Timer

Motor_Down_Timer is the state that sets the interrupt timer to make sure the machine stops after the current cycle.

Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	0
Hbridge0	0
Hbridge1	0
Display	12
Timer start	Belt

Aborted

Aborted is the state where the machines goes to if the abort button is pressed, the machine has come to a halt.

Outputs	Value for output
Lens lamp position	0
Lens lamp sorter	0
Engine conveyor	0
Engine feeder	0
Hbridge0	0
Hbridge1	0
Display	17
Timer start	0

Running_Stop

Running_Stop gives the same outputs as the Running state, the only difference being a running timer in the stop process.

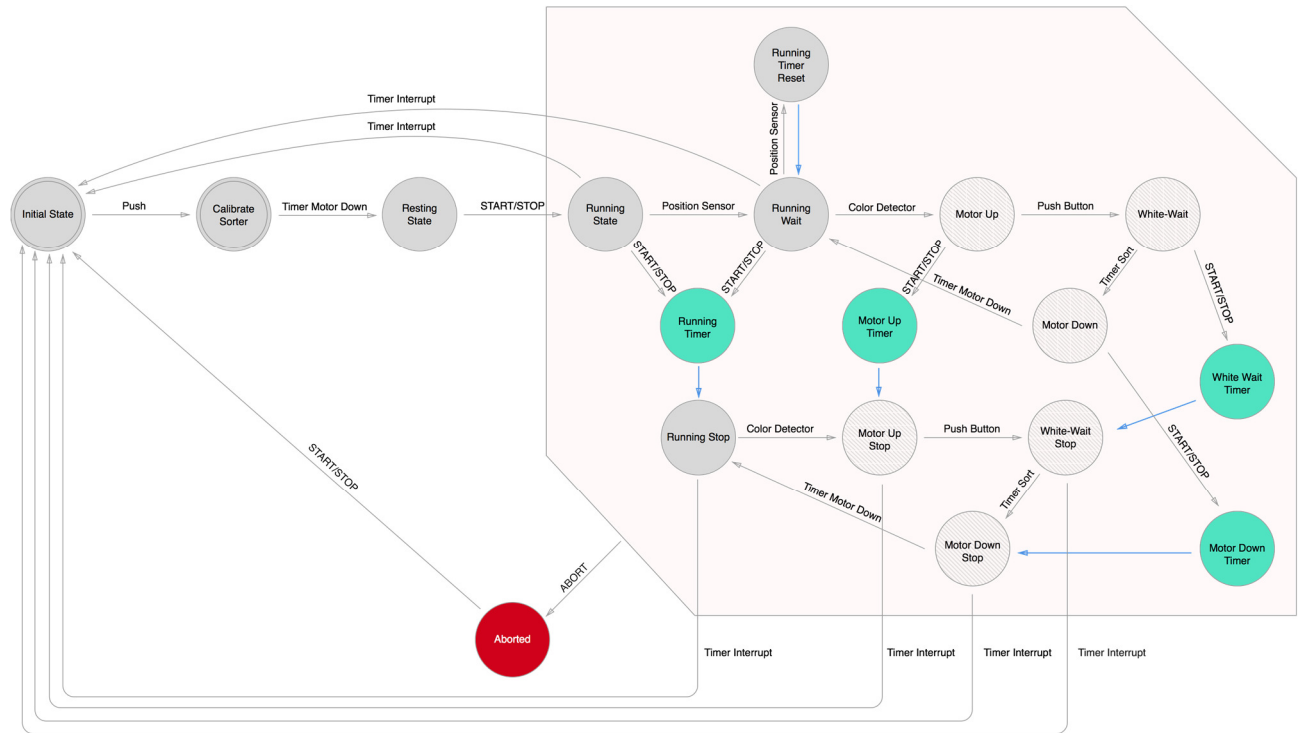
Outputs	Value for output
Lens lamp position	1
Lens lamp sorter	1
Engine conveyor	1
Engine feeder	0
Hbridge0	0
Hbridge1	0
Engine sorter	0
Display	13
Timer start	Belt

State transitions

Current state	Input	Input value	Next State
Initial	Push	1	Calibrate_Sorter
Calibrate_Sorter	Push	0	Resting
Resting	StartStop	1	Running
Running	Timer	TEnd	Initial
Running	PositionSensor	0	Running_Wait
Running	Abort	1	Aborted
Running	StartStop	1	Running_Timer
Running_Wait	Timer	TEnd	Initial
Running_Wait	PositionSensor	0	Running_Timer_Reset
Running_Wait	ColorDetector	1	MotorUp
Running_Wait	StartStop	1	Running_Timer
Running_Wait	Abort	1	Aborted
Running_Timer_Reset	Tick	1	Running_Wait
Running_Timer_Reset	Abort	1	Aborted
MotorUp	PushButton	1	WhiteWait
MotorUp	StartStop	1	Motor_Up_Timer
MotorUp	Abort	1	Aborted
WhiteWait	StartStop	1	White_Wait_Timer
WhiteWait	Abort	1	Aborted
WhiteWait	Timer	SORT	MotorDown
MotorDown	StartStop	1	Motor_Down_Timer
MotorDown	Abort	1	Aborted
MotorDown	Timer	Motor Down	Running_Wait
Running_Timer	Timer	Tic	Running_Stop
Running_Timer	Abort	1	Aborted
Motor_Up_Timer	Timer	Tic	Motor_Up_Stop
Motor_Up_Timer	Abort	1	Aborted
White_Wait_Timer	Timer	Tic	White_Wait_Stop
White_Wait_Timer	Abort	1	Aborted
Motor_Down_Timer	Timer	Tic	Motor_Down_Stop
Motor_Down_Timer	Abort	1	Aborted
Motor_Up_Stop	PushButton	1	White_Wait_Stop
Motor_Up_Stop	Abort	1	Running_Stop
Motor_Up_Stop	Timer	Timer Interrupt	Initial
Motor_Up_Stop	Abort	1	Aborted
White_Wait_Stop	Timer	SORT	Motor_Down_Stop
White_Wait_Stop	Abort	1	Aborted
White_Wait_Stop	Timer	Timer Interrupt	Initial
Motor_Down_Stop	Timer	Motor Down	Running_Stop
Motor_Down_Stop	Abort	1	Aborted
Motor_Down_Stop	Timer	Timer Interrupt	Initial
Running_Stop	ColorDetector	1	Motor_Up_Stop
Running_Stop	Abort	1	Aborted
Running_Stop	Timer	Timer Interrupt	Initial
Aborted	StartStop	1	Initial

Finite-state Automaton

Blue line means that the trigger for the transition is a clock tick.



Tests done

On the next page is the UPPAAL model. This UPPAAL model has been tested for 2 safety properties. The first one is “After the start-up of the machine, the assembly program should not stop until the machine is shut down.” This has been tested using the following property “A[] not deadlock”, and we didn’t have a deadlock. The second safety property which was tested is: “The outputs connected to the h-bridge may never be powered on at the same time.” This was tested using the following property “A<> !(hbridge0==1 && hbridge1=1)”. This one was also correct.

Validation

Validation of “Inputs and Outputs”

We see that the inputs and outputs of Software Specification are correct. The inputs of Machine Design should be equal to the outputs of Software Specification, which they are.

Validation of “Relations”

The relations between the inputs and outputs can be validated with the input/output tables. For all inputs, we have outputs. These outputs depend on one or more inputs, which is described in the Relations.

Validation of “Description of States”

To validate the states we will look at the USE-cases again to see if every USE-case is implemented. To do this we look at the basic flow and trigger of every use case and see what states we use to realize this.

We also validate the states to the relations. For every USE-case we looked at what states would be necessary to achieve it.

Starting the machine

Preconditions: -

Trigger: Booting the machine / finished the abort or start/stop routine

Postconditions: The machine starts the sorting process.

Basic Flow	State	Explanation
Before Trigger	Any State	It does not really matter which state the machine is in before the trigger
After Trigger	Initial State	Initial state is the first state, so after booting the machine we will be here. Finishing the abort or start/stop routine will also end in the initial state
1. Machine puts devices in their initial state.	Initial State + Calibrate Sorter + Resting State	The only thing that needs to be put into an initial state is the sorter mechanism. In initial state the machine moves the sorter up until it touches the push button. It then transitions to Calibrate Sorter where it starts moving down. After a set amount of time it will stop moving the sorter and transition to the resting state. This way we know exactly where the sorter is positioned
1. The user presses the START/STOP button	Running State	From the Resting State the transition to the running state is pressing the START/STOP button
Postconditions	Running State	The running state is the start of the sorting process

Stopping the machine

Preconditions: The machine is running.

Trigger: The START/STOP button is pressed.

Postconditions: The machine is sent into an inactive state with no process interrupted.

Basic Flow	State	Explanation
Preconditions	Not initial state, Calibrate Sorter or aborted	When the machine is not in any of these states it is running.
After Trigger	One of the (greenblue) Timer states	When the START/STOP is pressed the machine transitions to a timer start state, which starts a timer and stops the feeder mechanism.
1. The machine finishes sorting the discs currently in the machine	One of the sorting states	While the timer is running the machine keeps sorting. The timer is the time it takes for the conveyor belt to make a complete rotation, guaranteeing there are no more discs on the belt.
1. The machine enters an inactive state and will not take any more discs from the storage* unless the START/STOP button is pressed.	Initial State + Calibrate Sorter + Resting State	After going through the initialize process we go back to the resting state, which waits on the START/STOP button.
Postconditions	Resting State	Resting state in an inactive state and we finished the sorting process.

Sort unsorted discs

Preconditions: The machine is not already running.

Trigger: The user provides unsorted discs and presses the “START” button.

Postconditions: There are no unsorted discs left, all sorted discs are in a container based on their colour.

Basic Flow	State	Explanation
Preconditions	Resting State	The program first initializes and then waits for the user to press that start button. This waiting happens in the Resting State. In the resting state the machine is not running
After Trigger	Running State	Pressing START/STOP is the input to transition to the running state
1. An unsorted disc is moved to the colour detector	Running Wait + Running Timer Rest	When moving to the colour detector it will have to pass the position Sensor which is the input to move to Running Wait, the disc is then still in front of the position sensor so the program moves to Running Timer Rest
1. The machine decides to which of the two containers the disc needs to be moved	Running Wait + Running Timer Rest OR Motor Up + White-Wait	Depending on whether the disc is white or black the sorter either needs to move down or keep its down position. If it keeps its down position it should just keep checking for an unsorted disc and when it detects one it will move to Running Timer Rest If it needs to move up the colour detector will detect a white disc and therefore transition to Motor Up. Moving the sorter up will trigger the pushButton, which is the input to transition to White-Wait
2. The machine moves the disc to the designated container	Running Wait + Running Timer Rest OR Motor Down + Running Wait	If the sorter did not detect a white disc we are still waiting like in basic flow 2. If it did detect one then while the disc is moving to the designated container the sorttimer will count down making the machine transition to Motor Down
3. The machine repeats step 2 through 4 until all discs have been sorted	-	
4. The machine pauses within 4 seconds	Initial State + Calibrate Sorter + Resting State	If there are no discs anymore the machine will stay in Running Wait waiting for the timer interrupt which will come within 4 seconds, making the machine transition to initial state. There it will reset the sorter and transition to the resting state
Postconditions	Resting State	We repeated the sorting step until all discs were sorted, meaning all discs are now sorted

Abort the process

Preconditions: The machine is sorting discs

Trigger: The user wants to immediately stop the machine.

Postconditions: The machine stopped running and is ready to start again.

Basic Flow	State	Explanation
Preconditions	Every that is not initial state, Calibrate Sorter, resting state or Aborted	All other states are states in which discs are being sorted
After Trigger	Aborted	Every state (apart from the one mentioned in before trigger) have a line to abort with Abort as input
1. The machine stops transporting the discs. And doesn't put any more discs on the transporting mechanism.	Aborted	Because the machine is now in the abort state, which has all outputs set to 0, nothing will be moving.
1. The user is required to remove all discs that are neither in the container unit nor sorted.	Aborted	The machine will remain in Abort until the user presses START/STOP. This means everything is stopped and the user can safely remove all discs
2. When the user removed all unsorted discs that were not in the container unit he presses the START/STOP button.	Initial State + Calibrate Sorter + Resting State	Pressing the START/STOP button is the input for the transition to Initial State There it will reset the sorter and transition to the resting state
Postconditions	Resting State	We are in the resting state, so the machine has stopped running. The resting State is also the state from which you can start the machine again

Booting of the machine and Shutting down the machine do nothing with our software. This means they do not use states. This also means we can't validate those USE-Cases here.

Validation of “State Transitions”

The description of our machine states is validated through its representation in the transition table. No state is excluded from being represented in the state transition table, all transitions will have the initial transition state differ from the end state.

Validation of “Finite-state Automaton”

When we were making our finite-state automaton we looked at our state description and made sure that all states were represented, then we used our state transition table to make sure all transitions were correctly implemented.

Validation of “UPPAAL model”

All transitions which exist in the UPPAAL model also occur in the Finite State Automaton. And the same action has to be performed to take that transition. Also all states of the Finite State Automaton occur in the UPPAAL model. The states of the UPPAAL model also have the outputs in them. The states of the Finite State Automaton do not have the outputs in them. Thus we validate the values of the outputs, which are in the states, to the description of the states.

Software Design

In the Software Design phase, we 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.

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>.

PHP code used in this project follows the Zend Framework Coding Standard for PHP.

Source: <http://framework.zend.com/manual/1.12/en/coding-standard.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 state 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 \$branchTO where to branch to2 then call calibrateSensor function and if the if statement wasn't satisfied we recall initial entering a loop.

Translating from Java to PHP

The java code was written such that the conversion process to php is as easy as possible.

All variable in java will have the "\$" sign added at the beginning of their name to comply with the php standards. The "\$" sign has no influence in the java program variable naming, while in php it is mandatory.

Design decisions for the Java code

In translating our transition table to a Java program we made a number of decisions shaping the code, these decisions are outlined in this section.

We started by looking at our transition table, in this table we had our transitions ordered by the “current state”, the state where the transition starts. Then there were some inputs that could trigger a transition from this state to a number of other states. Because of this we thought it would make sense to write a function for each state, since it would allow our code to essentially be a condensed version of the transition table. Where the code would be ordered by the “current state”, and each state would have a number of outgoing transitions to other states. This resulted in the following blueprint for each of our functions:

```
void function(){
    timerManage();           // The function that manages the outputs and PWM
    $temp = getButtons();    // Store the buttons currently being pressed in a temporary
    variable

    if( condition for transition ){           // Do this once for every outgoing transition from this
    state
        Changes required to change to the new statement;
    }

    function();           // Call on the same function again to recheck the buttons and
    continue running the machine with timerManage()
}
```

Then we made an extra function which will be called from each function to do the PWM. This function is called timerManage. This function firstly gets the voltage which the output needs from the array.

This function has a variable called counter which increments each time the outputs have been set. That value is take modulo 12. So it will leave the outputs which need 12 volts on all the time. The reason why the values which need less than 12 volt will be turned off after they have been on for long enough. That goes as follows. First it checks if the engine needs to be on by checking if the voltage it needs is higher than counter. If the output needs to be on then it gets the location of the value in the array. And then does 2 to the power of the location. So now the correct output will be set on. Then the value of 2 to the power will be added to the variable engines. Then after all 7 outputs have been through that loop then it will set the output to the value of engines. So the lights which needed to be on will be on. Now the value of counter will increment each time and take modulo 12.

We also choose to save certain values, which may not be expected to be saved. In this section I will explain why we save the 2 variables. The first one is the variable of the location of the code. This has been saved because then we then we are capable of changing the return address after the timer interrupt. Because when an timer interrupt occurs we want to return to the initial state and the position where we were before. We also saved the original position of the stack pointer for when we come back from the timer interrupt to make sure that we empty the stack. Because there may be some values on the stack from before the timer interrupt. Thus to remove them we set the stack pointer to its original value.

Validation

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 (value of $k > \text{counter}$) it will be set on else it will stay off. So now the loop invariant holds for the element $k+1$

Termination:

The loop will terminate when k is greater than 7. Because we do not have any more outputs.

Control flow validation

Because the Java code has been validated to the state description and the transition table, which, in turn, have been validated with the UPPAAL model and shown to be correct and in tune with the initial description of the sorting machine. This means that the Java program, being a one-to-one translation of the finite state automaton, also has a correct control flow.

Software Implementation and Integration

Now we show the data representation and coding standard we chose that is used to write the Assembly Language.

Java to PHP

The Java to PHP conversion is usually natural, the two languages sharing most syntax but there are some differences we must note down. We are not required to create a class in PHP. The initialization will differ in PHP from Java, but they share the same core in the end. Also while we have some of the variables initialized globally in Java, in PHP they will be local. Having no class will make the class initialization irrelevant in PHP and that's why its missing. The later functions in the Java code right after the function TimerManage are included in the PHP code using "include "functions.php";". In TimerManage, % operation is replaced by the mod() function. Due to our PHP compiler limitations we are required to use variables as arguments when calling certain functions like for example storeData. The PHP code has been added as appendix 3.

Example of Java to PHP

```
141 void running() {
142     timerManage();
143
144     //check if we need to pause
145     $startStop = getButtonPressed(0);
146
147     if ($startStop == 1) {
148         //stop the feeder engine
149         storeData(0,"outputs",FEEDERENGINE);
150
151         //set the timer
152         setCountdown(BELT * 10);
153
154         //update the state
155         storeData(9,"state",0)
156
157         runningTimer();
158     }
159
160     //check if a disk is at the position detector
161     $position = getButtonPressed(7);
162     if ($position == 0) {
163         //reset the countdown, a disk was detected
164         setCountdown(BELTROUND+BELT);
165
166         //update the state
167         storeData(4,"state",0);
168
169         runningWait();
170     }
171
172     //Loop
173     running();
174 }
```

```
198 function running()
199 {
200     timerManage();
201
202     //check if we need to pause
203     $startStop = getButtonPressed(0);
204     if ($startStop == 1) {
205         //stop the feeder engine
206         $temp = 0;
207         storeData($temp, 'outputs', FEEDERENGINE);
208         unset($temp);
209
210         //exit after 1 rotation of the belt
211         setCountdown(BELT * 10);
212
213         //update the state
214         $state = 9;
215         storeData($state, 'state', 0);
216         unset($state);
217
218         runningTimer();
219     }
220     unset($startStop);
221
222     //check if a disk is at the position detector
223     $position = getButtonPressed(7);
224     if ($position == 0) {
225         //reset the countdown, a disk was detected
226         setCountdown(COUNTDOWN);
227
228         //update the state
229         $state = 4;
230         storeData($state, 'state', 0);
231         unset($state);
232         runningWait();
233     }
234     unset($position);
235
236     //Loop
237     running();
238 }
239 }
```

Example of PHP to Assembly

To make things more clear, we will show you an example of how the PHP is transformed into Assembly:

PHP	Assembly
197 <i>//state 3</i>	
198 function running()	
199 {	
200 timerManage();	265 running: BRS timerManage
201	
202 <i>//check if we need to pause</i>	
203 \$startStop = getButtonPressed(0);	266 PUSH R3
	267 LOAD R3 0
	268 BRS _pressed
	269 PULL R3
	270 SUB SP 5
	271 PULL R3
	272 ADD SP 4
204 if (\$startStop == 1) {	273 CMP R3 1
	274 BEQ conditional3
205 <i>//stop the feeder engine</i>	
206 \$temp = 0;	289 conditional3: LOAD R4 0
207 storeData(\$temp, 'outputs', FEEDERENGINE);	290 STOR R4 [GB +outputs + FEEDERENGINE]
208 unset(\$temp);	291
209	
210 <i>//exit after 1 rotation of the belt</i>	
	292 PUSH R5 ;reset timer
	293 PUSH R4
	294 LOAD R5 -16
	295 LOAD R4 0
	296 SUB R4 [R5+13]
	297 STOR R4 [R5+13] ;set timer
	298 LOAD R4 BELT * 10
	299 STOR R4 [R5+13]
	300 PULL R4
211 setCountdown(BELT * 10);	301 PULL R5
212	
213 <i>//update the state</i>	
214 \$state = 9; <i>//TODO: echte state</i>	302 LOAD R4 9 ;\$state = 9
215 storeData(\$state, 'state', 0);	303 STOR R4 [GB +state + 0]
216 unset(\$state);	304
217	
218 runningTimer();	305 BRA runningTimer
219	
220 }	
221 unset(\$startStop);	275 return3:
222	
223 <i>//check if a disk is at the position detector</i>	
224 \$position = getButtonPressed(7);	276 PUSH R3
	277 LOAD R3 7
	278 BRS _pressed
	279 PULL R3
	280 SUB SP 5
	281 PULL R3
	282 ADD SP 4
225 if (\$position == 1) {	283 CMP R3 1
	284 BEQ conditional4
226 <i>//reset the countdown, because a disk was just detected</i>	
227 setCountdown(COUNTDOWN);	308 conditional4: PUSH R5 ;reset timer
	309 PUSH R4
	310 LOAD R5 -16
	311 LOAD R4 0
	312 SUB R4 [R5+13]
	313 STOR R4 [R5+13] ;set timer
	314 LOAD R4 COUNTDOWN
	315 STOR R4 [R5+13]
	316 PULL R4
	317 PULL R5
228	
229 <i>//update the state</i>	
230 \$state = 4;	318 LOAD R4 4
231 storeData(\$state, 'state', 0);	319 STOR R4 [GB +state + 0]
232 unset(\$state);	320
233 runningWait();	321 BRA runningWait
234 }	
235 unset(\$position);	285 return4:
236	
237 <i>//Loop</i>	
238 running();	286 BRA running
239 }	

Important things to note:

The line numbers of the assembly jump at some points, for example at assembly line number 274. This is because in assembly you will first get the whole function and then at the bottom the if statements in this function. In PHP however the if statements are inline.

Another thing that is different is some functions that need more code in assembly. For example the function “getbuttonpressed” which is used on PHP line 203 takes a few lines more lines in assembly.

Java to PHP

The Java to PHP conversion is usually natural, the two languages sharing most syntax but there are some differences we must note down. We are not required to create a class in PHP. The initialization will differ in PHP from Java, but they share the same core in the end. Also while we have some of the variables initialized globally in Java, in PHP they will be local. Having no class will make the class initialization irrelevant in PHP and that's why its missing. The later functions in the Java code right after the function TimerManage are included in the PHP code using "include "functions.php";". In TimerManage, % operation is replaced by the mod() function. Due to our PHP compiler limitations we are required to use variables as arguments when calling certain functions like for example storeData. The PHP code has been added as appendix 5.

Validation of Java to PHP

Because of the natural similarity and ease of conversion, the PHP codes correctness can be correlated to its java counterpart, the correctness of the java code was validated in the Validation part of the Software Design.

System Validation and Testing

Finally, we demonstrate that the final product meets its initial requirements, i.e. we prove that the executable code correctly implements the System Level Requirements, and that the implementation doesn't do more than is expected.

Validation Policy

In our documents we have validated every element of contents in a separate Validation section at the end of the document or near to it.

Machine Design will have at the end of the document a Validation section (pg. 20) which includes the Validation of High Level Specifications and the Validation of the System Level Requirements, also adding Validation to Design Priorities.

Software Specification Document will have a Validation section (pg. 34) that will contain the validation of the Inputs and Outputs, the Relation of Inputs and Outputs, the Description of States, the State Transitions, the Finite State Automaton and the UPPAAL model.

Software Design will have a Validation section (pg. 41) close to the end of the document being afterwards followed by the Program Code. The Validation will contain the validation of the java code to the transition table (from the Software Specification), validation of the timerManage function (this function needed separate formal proof for its inner loop) and Control flow validation.

Software Implementation and Integration Document will have at the end a Validation section (pg. 44) containing validation of the PHP code to java and the validation of the Assembly code to the PHP compiler.

Validating the machine to the priorities

We validated the machine to be reliable by making it run and sort 100 discs, the results of multiple test concluded that the machine had faulted once in sorting one disc during the 100 discs test, thus exceeding the 95 % reliability we determined the machine needed to be considered reliable.

Throughout tests of the machine we determined that a full container of 12 discs, 6 black and 6 white randomly placed in the container, is sorted in 11 seconds. This results meets our expectancy to sort more than a disc per second.

During previous tests the machine didn't break physically, thus we consider the machine to be robust.

The machine is user accessible, once set up as described in the documentation the user is only required to utilize two push buttons and insert all the discs in the container. During testing all push buttons worked as intended and the sorter didn't create problems of any sort, due to carefully placed walls and the movement direction imposed by the feeder and conveyer belt the discs during testing ended up only in their specific trays, most of the machine is opened so if the machine is aborted any discs is in reach.

The machine was built on only one floorboard indirectly limiting our space and such obtaining a normal sized machine.

The machine was built in time to respect the group established dead line. Thus we consider easy to build.

The overall machine doesn't use more parts than necessary, the machine contains a conglomerate of pieces that replaces a single piece, with the same functionality, only in the case that the single piece is unavailable or doesn't offer the same advantage as the conglomerate when querying through the higher priorities, the most common is that a single part doesn't provide enough robustness or might make the machine fault.

Validating the machine to the USE-cases

The machine was tested in real life, during the tests the behavior was according to the USE-cases (Machine Design). The machine booted up, it started once the START/STOP button was pressed and stopped when the START/STOP button was pressed again and the last disc on the conveyor belt was sorted. When the ABORT button was pressed during the running phase the plastic wear halted immediately. The display outputted correctly every state in which the machine was, during the tests the discs were sorted properly and when there were no more discs left, the machine stopped after under 4 seconds. The machine was then powered off with no difficulty.

Conclusion

The machine delivers satisfactory results, it accomplishes the project goal and fulfilled the group expectations.

Process

Work Plan

To streamline the group process we needed a Work Plan. We started this Work Plan with the inventory of the goals and objectives of each phase of the project. For the roles in the group we chose to have them the same as described in /Project Guide Design Based Learning "DBL 2IO70" "Sort It Out"/.

Then we come to the definition of our terms. We chose to have abbreviations of the phases and the tasks. This way we can refer to them without having to waste a lot of space if we mention them multiple times. Also the roles have their abbreviations.

Before we use those abbreviations we first have an inventory of the amount of work and an overview of the main deliverables. The amount of work is given per phase and week in a nifty table. The overview of deliverables contains who's responsible for a certain deliverable and the date and week the deliverable is due.

Then we come to the weekly tables. Tuesday and Friday we have a tutor meeting and we work afterwards till in the afternoon. On Wednesday we have Data Structures in the morning and work on the project afterwards. Those times are included in the tables. Everyone has his column with his role if applicable. For every hour and person it's defined what he will be working on.

With this Work Plan and the collective logbook we're able to have an indication of how much time was spent on each task by each member. If necessary action can be taken based on this indication.

If unforeseen problems arise and the deadline is close, this means we have to work harder. Deadlines aren't easily moved. If someone spends too less time according to the Work Plan it's expected he does his work at home.

Workday

For us, a normal workday is structured as follows: we start each workday with a list of items that needs to be done in order to complete the document for that week. The list is written on the whiteboard that is available in the room. Then members are assigned to a task in consultation. After the completion of a task, it is checked off or removed from the whiteboard, and the member that was responsible for it continues to work on the next item of the inventory until there are no more available assignments. Next, they will help another group member with their duty. This cycle repeats itself whenever we are together. On Wednesday, the document is wrapped up and cross-read. The person that bears the responsibility for the document hands in the current document for feedback when possible. On Friday, the document is updated according to the feedback given by the tutor. Subsequently, the finalised document is cross-read, and handed in by the person responsible for the document.

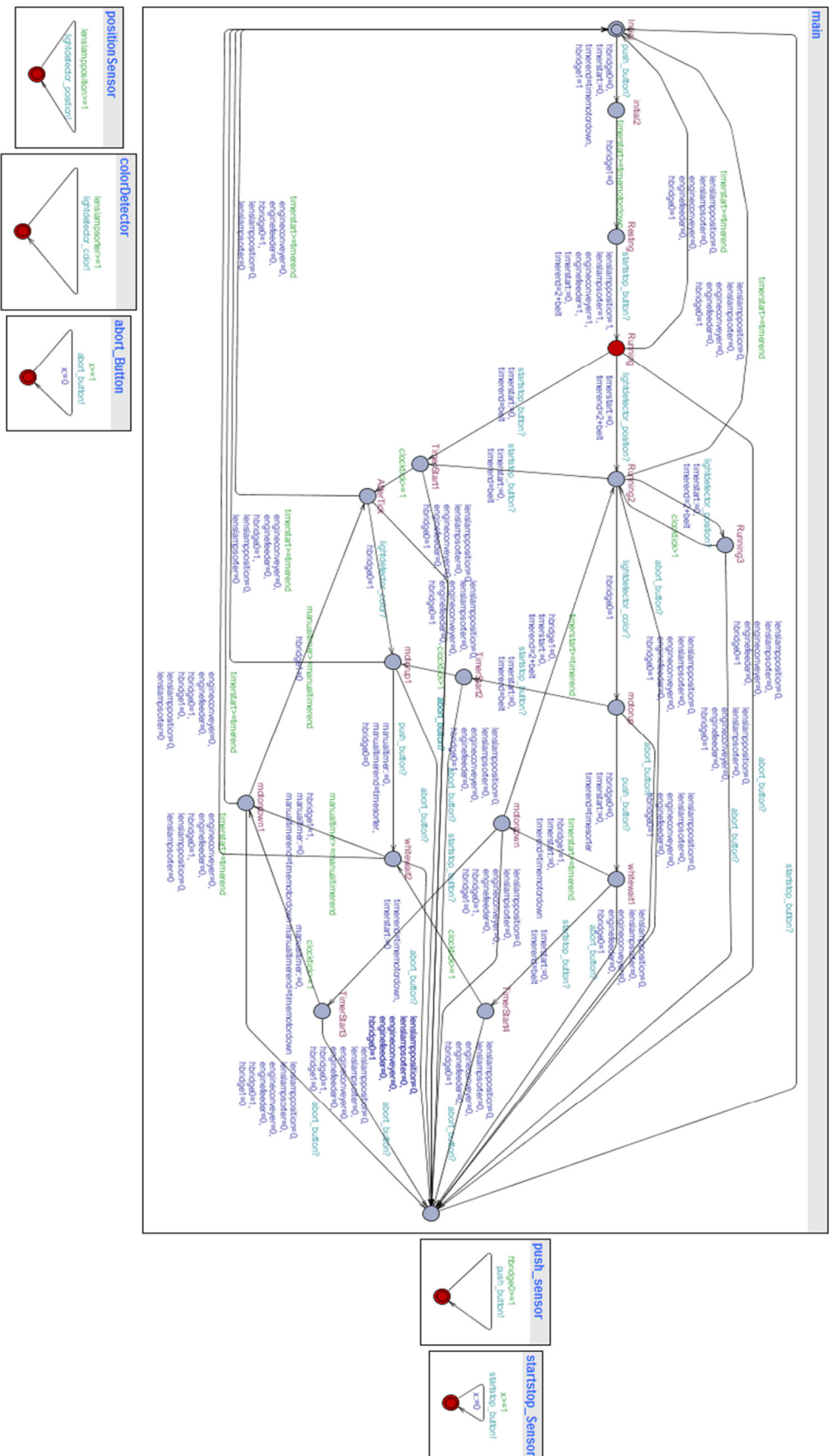
Problems

There was a problem with the group not functioning as was expected. The logbook indicated that some members contributed less than other members. As a result, other members had to compensate for it by spending more time on the project. Therefore, we decided to address this problem in the meetings and to distribute the workload more evenly.

Conclusion

Over the course of these past 8 weeks we worked on making a sorting machine and the software that runs it. We did this by going through multiple phases, starting with Machine Design, where we designed the machine itself. Moving to Software Specification, where we created a finite state automaton, then Software Design and Software Implementation and Integration where we respectively designed a pseudo-Java program and then translated that into Assembly for the PP2. While making these documents we validated each part to what we did before to make sure that we made the right decision every time. While the project took a lot of our time each week, we liked doing it, and the end result was very satisfying. We hope that the skills we have acquired over the course of this project, both those for designing and building a product and those for working in a group, will help us in future projects both here in the TU/e and beyond.

Appendix 1: UPPAAL model



Appendix 2: Table of the display of states

Number	State
	Boot
0	Initial State
1	Calibrate Sorter
2	Resting State
3	Running State
4	Running Wait
5	Running Timer
6	Motor up
7	White Wait
8	Motor Down
9	Running Timer
10	Motor Up Timer
11	White Wait Timer
12	Motor Down Timer
13	Running Stop
14	Motor Up Stop
15	White Wait Stop
16	Motor Down Stop
17	Aborted

Appendix 3: Java Program

```

1  /*
2   *Sort of a simulation of the PP2 program
3   *controlling the Fischer
4   *Technik in order to sort black and white discs.
5   *
6   *@author Maarten Keet
7   *@author Stefan van den Berg
8   *@author Rolf Verschuuren
9   *@author Wigger Boelens
10  *@team Group 16
11  *@since 13/3/2015
12  */
13
14
15 class SoftwareDesign {
16     /**@CODE**
17     //inputs
18     int $push, $startStop, $abort, $position,
19         $colour;
20
21     //variables
22     int $state = 0;
23     int $sleep = 0;
24     int $temp = 0;
25     int $location;
26     int $counter = 0;
27     int $engines;
28
29
30     //constants
31     final int TIMEMOTORDOWN = 30;
32     final int BELTROUND = 2000;
33     final int BELT = 1200;
34     final int SORT = 850;
35     final int LENSAMPPOSITION = 5,
36         LENSAMPSORTER = 6,
37         HBRIDGE0 = 0,
38         HBRIDGE1 = 1,
39         CONVEYORBELT = 3,
40         FEEDERENGINE = 7,
41         DISPLAY = 8,
42         LEDSTATEINDICATOR = 9;
43
44     public static void main(String args[]) {
45         SoftwareDesign softwareDesign = new
46             SoftwareDesign();
47
48
49         //values for the data segment
50         softwareDesign.initVar("outputs", 12);
51         softwareDesign.initVar("stackpointer", 1);
52         softwareDesign.initVar("offset", 1);
53
54         //store the offset of the program, this
55         //is used in the interrupt
56         softwareDesign.storeData(startOfTheCode,
57             "offset", 0);
58
59         //store the value of the stackpointer, so
60         //we can clear the stack
61         //easily
62         softwareDesign.storeData(SP,
63             "stackpointer",
64             0);
65
66         $counter = 0;
67
68         //reset outputs
69         softwareDesign.storeData(0, "outputs",
70             softwareDesign
71                 .HBRIDGE0);
72         softwareDesign.storeData(0, "outputs",
73             softwareDesign
74                 .HBRIDGE1);
75         softwareDesign.storeData(0, "outputs",
76             softwareDesign
77                 .LENSAMPPOSITION);
78         softwareDesign.storeData(0, "outputs",
79             softwareDesign
80                 .LENSAMPSORTER);
81         softwareDesign.storeData(0, "outputs",
82             softwareDesign
83                 .LEDSTATEINDICATOR);
84         softwareDesign.storeData(0, "outputs",
85             softwareDesign
86                 .DISPLAY);
87         softwareDesign.storeData(0, "outputs",
88             softwareDesign
89                 .CONVEYORBELT);
90         softwareDesign.storeData(0, "outputs",
91             softwareDesign
92                 .FEEDERENGINE);
93
94         //start moving the sorter up
95         softwareDesign.storeData(9, "outputs",
96             softwareDesign
97                 .HBRIDGE0);
98
99         //go to the first state and set the
100         //value for the display
101         softwareDesign.$state = 0;
102         softwareDesign.initial();
103     }
104
105     //state 0
106
107     void initial() {
108         setStackPointer(
109             getData("stackpointer", 0));
110         timerManager();
111         //check if the sorter push button is
112         //pressed
113         $push = getButtonPressed(5);
114         if ($push == 1) {
115             //move the sorter down
116             storeData(0, "outputs", HBRIDGE0);
117             storeData(9, "outputs", HBRIDGE1);
118             //update the state
119             $state = 1;
120             //reset sleep for the next function
121             $sleep = 0;
122             calibrateSorter();
123         }
124     }
125
126     //state 1
127     void calibrateSorter() {
128         timerManager();
129         //the sorter is now moving down,
130         //and we're waiting for it to reach the
131         //bottom
132         if ($sleep == TIMEMOTORDOWN * 1000) {
133             //stop the sorter
134             storeData(0, "outputs", HBRIDGE1);
135             //update the state
136             $state = 2;
137             //reset sleep
138             $sleep = 0;
139             resting();
140         }
141         //loop
142         $sleep++;
143         calibrateSorter();
144     }
145
146     //state 2
147     void resting() {
148         timerManager();
149         //the program waits for the user to
150         //press the start/stop
151         $startStop = getButtonPressed(0);
152         if ($startStop == 1) {
153             //sleep so we don't go to the pause
154             //immediately
155             sleep(2000);
156             //power up the lights
157             storeData(12, "outputs",
158                 LENSAMPPOSITION);
159             storeData(12, "outputs",
160                 LENSAMPSORTER);
161             //start up the belt and the feeder
162             storeData(9, "outputs", CONVEYORBELT);
163             storeData(5, "outputs", FEEDERENGINE);
164             //set and start the countdown
165             setCountdown(BELTROUND + BELT);
166             startCountdown();
167             //update the state
168             $state = 3;
169             running();
170         }
171         //loop
172         resting();
173     }
174
175     //state 3
176     void running() {
177         timerManager();
178         //check if we need to pause
179         $startStop = getButtonPressed(0);
180         if ($startStop == 1) {
181             //stop the feeder engine
182             storeData(0, "outputs", FEEDERENGINE);
183             //set the timer
184             setCountdown(BELT);
185             //update the state
186             $state = 9;
187             runningTimer();
188         }
189         //check if a disk is at the position
190         //detector
191         $position = getButtonPressed(7);
192         if ($position == 1) {
193             //reset the countdown, because a
194             //disk was detected
195             setCountdown(BELTROUND + BELT);
196             //update the state
197             $state = 4;
198             runningWait();
199         }
200         //loop
201         running();
202     }
203
204     void runningWait() {
205         timerManager();
206         //check if we need to pause
207         $startStop = getButtonPressed(0);
208         if ($startStop == 1) {
209             //stop the feeder engine
210             storeData(0, "outputs", FEEDERENGINE);

```

```

211 //set the timer
212 setCountdown(BELT);
213 //update the state
214 $state = 9;
215 runningTimer();
216 }
217 //check if a disk is at the positiond
218 // detector
219 $position = getButtonPressed(7);
220 if ($position == 0) {
221 //reset the countdown,because a
222 // disk was detected
223 setCountdown(BELTROUND + BELT);
224 //update the state
225 $state = 5;
226 runningTimerReset();
227 }
228 //check if a white disk is at the color
229 // detector
230 $colour = getButtonPressed(6);
231 if ($colour == 1) {
232 //move the sorter up
233 storeData(9, "outputs", HBRIDGE0);
234 //update the state
235 $state = 6;
236 motorUp();
237 }
238 //loop
239 runningWait();
240 }
241
242 //state 5
243 void runningTimerReset() {
244 timerManager();
245 //update the state
246 $state = 5;
247 runningWait();
248 }
249
250 //state 6
251 void motorUp() {
252 timerManager();
253 //check if we need to pause
254 $startStop = getButtonPressed(0);
255 if ($startStop == 1) {
256 //stop the feeder engine
257 storeData(0, "outputs", FEEDERENGINE);
258 //set the timer
259 setCountdown(BELT);
260 motorUpTimer();
261 }
262 //check if the sorter push button is
263 // pressed
264 $push = getButtonPressed(5);
265 if ($push == 1) {
266 //stop the engine,because it is in
267 // the right position
268 storeData(0, "outputs", HBRIDGE0);
269 //update the state
270 $state = 7;
271 whiteWait();
272 }
273 //loop
274 motorUp();
275 }
276
277 //state 7
278 void whiteWait() {
279 timerManager();
280 //we are waiting for the white disk to
281 // be sorted
282 if ($sleep == SORT * 1000) {
283 //start moving the sorter down
284 storeData(9, "outputs", HBRIDGE1);
285 //update the state
286 $state = 8;
287 //reset sleep for the next function
288 $sleep = 0;
289 motorDown();
290 }
291 }
292 //check if we need to pause
293 $startStop = getButtonPressed(0);
294 if ($startStop == 1) {
295 //stop the feeder engine
296 storeData(0, "outputs", FEEDERENGINE);
297 //set the timer
298 setCountdown(BELT);
299 //update the state
300 $state = 11;
301 whiteWaitTimer();
302 }
303 //loop
304 $sleep++;
305 whiteWait();
306 }
307
308 //state 8
309 void motorDown() {
310 timerManager();
311 //the sorter is moving down
312 if ($sleep == TIMEMOTORDOWN * 1000) {
313 //stop the sorter
314 storeData(0, "outputs", HBRIDGE1);
315 //update the state
316 $state = 9;
317 //reset sleep for the next function
318 $sleep = 0;
319 runningWait();
320 }
321 //check if we need to pause
322 $startStop = getButtonPressed(0);
323 if ($startStop == 1) {
324 //stop the feeder engine
325 storeData(0, "outputs", FEEDERENGINE);
326 //set the timer
327 setCountdown(BELT);
328 motorDownTimer();
329 }
330 //loop
331 $sleep++;
332 motorDown();
333 }
334
335 //state 9
336 void runningTimer() {
337 timerManager();
338 //update state
339 $state = 13;
340 runningStop();
341 }
342
343 //state 10
344 void motorUpTimer() {
345 timerManager();
346 //update state
347 $state = 14;
348 motorUpStop();
349 }
350
351 //state 11
352 void whiteWaitTimer() {
353 timerManager();
354 //update state
355 $state = 15;
356 whiteWaitStop();
357 }
358
359 //state 12
360 void motorDownTimer() {
361 timerManager();
362 //update state
363 $state = 16;
364 motorDownStop();
365 }
366
367 //state 13
368 void runningStop() {
369 timerManager();
370 //check if a white disk is at the
371 // colour detector
372 $colour = getButtonPressed(6);
373 if ($colour == 1) {
374 //move the sorter engine up
375 storeData(9, "outputs", HBRIDGE0);
376 //update the state
377 $state = 10;
378 motorUpStop();
379 }
380 //loop
381 runningStop();
382 }
383
384 //state 14
385 void motorUpStop() {
386 timerManager();
387 //check if the sorter push button is
388 // pressed
389 $push = getButtonPressed(5);
390 if ($push == 1) {
391 //stop the engine for the sorter
392 storeData(0, "outputs", HBRIDGE0);
393 //update the state
394 $state = 11;
395 whiteWaitStop();
396 }
397 motorUpStop();
398 }
399
400 //state 15
401 void whiteWaitStop() {
402 timerManager();
403 //check if the white disk has been sorted
404 if ($sleep == SORT * 1000) {
405 //start moving the sorter down
406 storeData(9, "outputs", HBRIDGE1);
407 //update the state
408 $state = 12;
409 //reset the sleep for the next
410 // function
411 $sleep = 0;
412 motorDown();
413 }
414 //loop
415 $sleep++;
416 whiteWaitStop();
417 }
418
419 //state 16
420 void motorDownStop() {
421 timerManager();
422 //check if the sorter has moved down
423 if ($sleep == TIMEMOTORDOWN) {
424 //stop the engine of the sorter
425 storeData(0, "outputs", HBRIDGE1);
426 //update the state
427 $state = 9;
428 //reset sleep for the next function
429 $sleep = 0;
430 runningWait();
431 }
432 }

```

```

433 //loop
434 $sleep++;
435 motorDownStop();
436 }
437
438 //not a state
439 void timerInterrupt() {
440 //show that we have timer interrupt
441 $state = 18;
442 //make the sorter move up
443 storeData(9, "outputs", HBRIDGE0);
444 //stop all other outputs
445 storeData(0, "outputs", HBRIDGE1);
446 storeData(0, "outputs", LENSAMPPOSITION);
447 storeData(0, "outputs", LENSAMPSPORTER);
448 storeData(0, "outputs",
449 LEDSTATEINDICATOR);
450 storeData(0, "outputs", DISPLAY);
451 storeData(0, "outputs", CONVEYORBELT);
452 storeData(0, "outputs", FEEDERENGINE);
453 //make sure that the outputs get set
454 // immediately
455 timerManage();
456 //set the display to the state of initial
457 $state = 0;
458
459 initial();
460 }
461
462 void abort() {
463 //stop all outputs
464 storeData(0, "outputs", HBRIDGE0);
465 storeData(0, "outputs", HBRIDGE1);
466 storeData(0, "outputs", LENSAMPPOSITION);
467 storeData(0, "outputs", LENSAMPSPORTER);
468 storeData(0, "outputs",
469 LEDSTATEINDICATOR);
470 storeData(0, "outputs", DISPLAY);
471 storeData(0, "outputs", CONVEYORBELT);
472 storeData(0, "outputs", FEEDERENGINE);
473 //make sure the outputs stop immediately
474 timerManage();
475 //update the state to be correct in
476 // aborted
477 $state = 17;
478 aborted();
479 }
480
481 }
482
483 //state 17
484 void aborted() {
485 timerManage();
486 //check if we can start again
487 $startStop = getButtonPressed(0);
488 if ($startStop == 1) {
489 //start moving the sorter up for
490 // calibration
491 storeData(1, "outputs", HBRIDGE0);
492 //update the state
493 $state = 0;
494 initial();
495 }

```

```

496 //loop
497 aborted();
498
499 }
500
501 void timerManage() {
502
503
504 //make sure that when counter can not
505 // be higher than 12
506 mod(13, $counter);
507 //get the voltage of output $location
508 int $voltage = getData("outputs",
509 $location);
510 //power up the output when it needs to
511 if ($voltage > $counter) {
512 $engines += pow(2, $voltage);
513 }
514 //check if we are in a new iteration
515 if ($counter == 0) {
516 //set the first part of the display
517 $temp = getData("state", 0);
518 mod(10, $temp);
519 display($temp, "display", "1");
520
521 }
522
523 //check if we are at the end of the
524 // iteration
525 if ($counter == 12) {
526 //set the second part of the display;
527 $temp = getData("state", 0);
528 $temp = $temp / 10;
529 mod(10, $temp);
530 display($temp, "display", "01");
531 }
532
533 //check if we did all outputs
534 if ($location > 7) {
535 display($engines, "leds", "");
536 //set the variables for the next run
537 $engines = 0;
538 $location = 0;
539 $counter++;
540
541 //check if abort is pressed
542 $abort = getButtonPressed(1);
543 if ($abort == 1) {
544 abort(); //stop the machine
545 }
546 return;
547 }
548
549
550 $location++;
551 timerManage();
552 }
553 }

```

Appendix 4: Explanation of the compiler

The compiler works in phases. We will go through these phases 1 by 1 to explain how the compiler does its job: compiling PHP-like code to assembly. Throughout the phases the compiler keeps track of the line number of the PHP code it is currently compiling and uses that, when an error occurs, to give information where the error is. The compiler is written in PHP5.6 and uses a command line interface.

Preprocessing

In the first phase, the input code will be made ready for the next steps. A few things happen in this phase: First the file is read into the memory. The next step is that all comments, newlines and extra spaces are stripped from the file. The file is then split into single lines using the “;” symbol that denotes the end of a line. The code is divided in three segments. The first segment starts at `/**COMPILER`, everything before this statement is ignored.

The preprocessor further removes some special statements that are needed to make valid php such as “global” and changes some shortcuts in their full version. For example `$abc++` will be changed into `$abc+=1`. This ensures that the compiler only needs to be able to handle `$abc+=1`.

Splitting

In the second phase the code is split up by function. Every function gets his own array with all the lines that are in that function. The code not inside of a function goes into a separate array.

Compiling

The third phase is the most important one. It starts by compiling the code that is at the start and not inside a function. While compiling it keeps track of what functions are called and adds these, if they are not already compiled, to the toCompile queue. This helps in making sure there is no dead code, as a function that is never called, will not be compiled. The compiler adds the function “main”, which is the default start point of the code, to the queue and starts processing it.

After compiling the main function it will continue in the next function in the toCompile queue and keep doing this till the toCompile queue is empty.

The compiling itself is not a lot more than a lot of regex and switch statements that look at the input and make a output from that. At the first notion of a variable a register is assigned to it. The code then uses this register in place of the variable. Some more difficult statements, like the function display which displays something, will BRS to premade assembly code that handles that. The compiler keeps track of which segments of the premade assembly code are used.

When the compiler meets an if statement, it saves the code inside it to a new function named “condtional*i*” where *i* is the amount of conditionals that have already been seen. It then places this function in the toCompile queue. It also saves the location of the end of the if statement, so it will later know where to return when the if function has ended.

For every line it compiles, it takes the corresponding line of PHP and inserts it as a comment in the assembly. This is to help in debugging.

Combining

After there are no functions left in the toCompile queue, the combining phase starts. In this phase all the functions and the code outside the functions are combined into a single array. This phase also adds the used premade functions at the top and inserts the return statements at the correct position.

Formatting

The last phase is the least interesting. It goes through the, now compiled code, and formats it. It uses either the length of the longest function name or the number 25 depending on which is larger to insert spaces in front of every line of code in a way everything lines up nicely. It also makes sure the comments line up nicely.

The last step the compiler takes is writing the compiled code to a file and using the assembler provided to create the hex code.

Appendix 5: Explanation of the compiler functions

storeRam(\$location, \$value)

Store a value in the ram.

\$location	The location (a variable) to store the value in the ram
\$value	The value to store, needs to be a variable
return	void

getRam(\$location)

Get a value from the ram.

\$location	The location (a variable) where the value is stored
return	The value that is stored at the location

display(\$what, \$onWhat, \$location = '000001')

Display something on either the display or the leds.

Possible values for \$onwhat:

- leds: the leds at the top
- leds2: the leds to the right
- display: the display

\$what	What to display, must be a variable
\$onWhat	On what to display
\$location position	Where to show the value when using the display, defaults to the right
return	void

pow(\$number,\$power)

Get the power of a number

\$number	The number to power
\$power	The power value
return	Int; The result

mod(\$what, \$variable)

Take the modulo of a number

\$what Modulo what

\$variable Variable to modulo over

return void

getInput(\$writeTo, \$type)

Get button or analog input. When you just want the input of 1 button, use `getButtonPressed` instead.

\$writeTo Variable to write the input to

\$type Type of input, possible values are: buttons, analog

return void

getButtonPressed(\$button)

Check if a button is pressed. Puts the result into R5.

\$button Which button to check (input a variable)

return Int; Whether or not the button is pressed.

installCountdown(\$functionName)

Install the countdown.

\$functionName The name of the function where the timer should go to

return void

startCountdown()

Start the countdown.

Retrun void

pushStack(\$variable)

Push a variable to the stack

\$variable The variable to push to the stack

return void

pullStack(\$variable)

Pull a variable from the stack.

<code>\$variable</code>	The variable where the pulled variable is put into
<code>return</code>	<code>void</code>

setCountdown(\$countdown)

Set the timer interrupt to a value. It will first reset the timer to 0.

<code>\$countdown</code>	How long the countdown should wait, in timer ticks
<code>return</code>	<code>void</code>

getData(\$location, \$offset)

Get data. Use offset 0 when it is just a single value.

<code>\$location</code>	The location where the variable is stored
<code>\$offset</code>	The offset of the location
<code>return</code>	The value of the data segment

storeData(\$variable, \$location, \$offset)

Store data. Use offset 0 when it is just a single value.

<code>\$variable</code>	The variable to store
<code>\$location</code>	The name of the location where the variable is stored
<code>\$offset</code>	The offset of the location
<code>return</code>	<code>void</code>

sleep(\$howLong)

Pause the program.

<code>\$howLong</code>	How long to sleep in clockticks
<code>return</code>	<code>void</code>

initVar(\$variable,\$places)

Initialize a variable that is used in that data segment.

<code>\$variable</code>	The name of the variable
<code>\$places</code>	How long the array is
<code>return</code>	<code>void</code>

branch(\$branchTO)

Branch to a function.

<code>\$branchTO</code>	where to branch to
-------------------------	--------------------

return void

moveFunction(\$branchTO)

Move a function in the assembly code.

\$branchTO Where to branch to

return void

Appendix 6: PHP Program

```

1 <?php
2 /* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */
3
4 /**
5  * Sort of a simulation of the PP2 program controlling the Fischer
6  * Technik in order to sort black and white discs.
7  * @team Group 16
8  * @author Stefan van den Berg
9  * @author Rolf Verschuuren
10 * @author Wigger Boelens
11 * @since 13/3/2015
12 */
13 include 'functions.php';
14 /**COMPILER**
15 moveFunction('timerInterrupt', 1);
16 moveFunction('timerManage', 50);
17
18 /**DATA**
19 initVar('offset', 1);
20 initVar('stackPointer', 1);
21 initVar('outputs', 12);
22 initVar('state', 1);
23
24 /**CODE**
25 define('TIMEMOTORDOWN', 150); //how long the sorter takes to move down
26 define('BELT', 2000);
27 define('BELTROUND', 2000); //Time for the belt to make a rotation
28 define('SORT', 200); //Clockticks to make a rotation
29 define('COUNTDOWN', 30000);
30 //outputs
31 define('LENSLAMPPPOSITION', 2);
32 define('LENSLAMPSORTER', 6);
33 define('HBRIDGE0', 0);
34 define('HBRIDGE1', 1);
35 define('CONVEYORBELT', 7);
36 define('FEEDERENGINE', 3);
37 define('DISPLAY', 8);
38 define('LEDSTATEINDICATOR', 9);
39
40 //not a state
41 function main()
42 {
43     global $counter, $location;
44
45     //store the offset of the program, this is used in the interrupt
46     storeData(RS, 'offset', 0);
47     //install the countdown
48     installCountdown('timerInterrupt');
49
50     //save the Location of the stackPointer, so we can clear the stack
51     storeData(SP, 'stackPointer', 0);
52
53     //the variables that are the same throughout the program:
54     $counter = 0;
55     $location = 0;
56     $sleep = 0;
57
58     //stop everything
59     $temp = 0;
60     storeData($temp, 'outputs', HBRIDGE1);
61     storeData($temp, 'outputs', LENSLAMPPPOSITION);
62     storeData($temp, 'outputs', LENSLAMPSORTER);
63     storeData($temp, 'outputs', LEDSTATEINDICATOR);
64     storeData($temp, 'outputs', DISPLAY);
65     storeData($temp, 'outputs', CONVEYORBELT);
66     storeData($temp, 'outputs', FEEDERENGINE);
67
68     //show the state
69     $state = 0;
70     storeData($state, 'state', 0);
71
72     //set HBridge so the sorter starts moving up
73     $temp = 10;
74     storeData($temp, 'outputs', HBRIDGE0);
75     unset($temp, $state);
76
77     //go to the first state
78     initial();
79 }
80
81 //state 0
82 function initial()
83 {
84     global $sleep;
85     //disable the Lights on the right hand side
86     $temp = 0;
87     display($temp, 'leds2');
88
89     $temp = getData('stackPointer', 0);
90     setStackPointer($temp);
91
92     timerManage();
93
94     //check if the sorter push button is pressed
95     $push = getButtonPressed(5);
96     if ($push == 1) {
97         //move sorter down
98         $temp = 0;
99         storeData($temp, 'outputs', HBRIDGE0);
100        $temp = 10;
101        storeData($temp, 'outputs', HBRIDGE1);
102
103        //update state
104        $temp = 1;
105        storeData($temp, 'state', 0);
106        unset($temp);
107
108        //reset sleep for the next function
109        $sleep = 0;
110
111        calibrateSorter();
112    }
113    unset($push);
114
115    //Loop
116    initial();
117 }
118
119 //state 1
120 function calibrateSorter()
121 {
122     global $sleep;
123     timerManage();
124
125     //the sorter is now moving down,
126     //we're waiting for it to reach its bottom position
127     if ($sleep == TIMEMOTORDOWN) {
128         //stop the sorter
129         $temp = 0;
130         storeData($temp, 'outputs', HBRIDGE1);
131
132         //update the state
133         $state = 2;
134         storeData($state, 'state', 0);
135         unset($state);
136
137         //reset sleep for the next state
138         $sleep = 0;
139         resting();
140     }
141
142     //Loop
143     $sleep++;
144     calibrateSorter();
145 }
146
147 //state 2
148 function resting()
149 {
150     timerManage();
151
152     //the program is now waiting for the user to press start/stop
153     $startStop = getButtonPressed(0);
154     if ($startStop == 1) {
155         //sleep so we don't go to pause immediately
156
157         //power up the Lamps
158         $temp = 12;
159         storeData($temp, 'outputs', LENSLAMPPPOSITION);
160         unset($temp);
161         timerManage();
162         sleep(1000);
163         $temp = 12;
164         storeData($temp, 'outputs', LENSLAMPSORTER);
165         unset($temp);
166         timerManage();
167         sleep(2000);
168
169         //start up the belt and feeder
170         $temp = 9;
171         storeData($temp, 'outputs', CONVEYORBELT);
172         $temp = 9;
173         storeData($temp, 'outputs', FEEDERENGINE);
174         unset($temp);
175
176         //set and start the countdown for the moment there are no more
177         disks
178         //this countdown will reset every time a disk is found
179         //when it triggers, timerInterrupt will be ran.
180         setCountdown(COUNTDOWN);
181         startCountdown();
182
183         //update the state
184         $state = 3;
185         storeData($state, 'state', 0);
186         unset($state);
187
188         running();
189     }
190     unset($startStop);
191
192     //Loop
193     resting();
194 }
195
196 //state 3
197 function running()
198 {
199     timerManage();
200
201     //check if we need to pause
202     $startStop = getButtonPressed(0);
203     if ($startStop == 1) {
204         //stop the feeder engine
205         $temp = 0;
206         storeData($temp, 'outputs', FEEDERENGINE);
207         unset($temp);
208
209         //exit after 1 rotation of the belt
210         setCountdown(BELT * 10);
211
212         //update the state
213         $state = 9; //TODO: echte state
214         storeData($state, 'state', 0);
215         unset($state);
216
217         runningTimer();
218     }
219     unset($startStop);
220
221     //check if a disk is at the position detector
222     $position = getButtonPressed(7);
223     if ($position == 0) {
224         //reset the countdown, because a disk was just detected
225         setCountdown(COUNTDOWN);
226     }
227 }

```

```

228
229 //update the state
230 $state = 4;
231 storeData($state, 'state', 0);
232 unset($state);
233 runningWait();
234 }
235 unset($position);
236
237 //Loop
238 running();
239 }
240
241 //state 4
242 function runningWait()
243 {
244     timerManage();
245
246     //check if we need to pause
247     $startStop = getButtonPressed(0);
248     if ($startStop == 1) {
249         //stop the feeder engine
250         $temp = 0;
251         storeData($temp, 'outputs', FEEDERENGINE);
252         unset($temp);
253
254         //exit after 1 rotation of the belt
255         setCountdown(BELT * 10);
256
257         //update the state
258         $state = 9;
259         storeData($state, 'state', 0);
260         unset($state);
261
262         runningTimer();
263     }
264     unset($startStop);
265
266     //check if a disk is at the position detector
267     $position = getButtonPressed(7);
268     if ($position == 0) {
269         //reset the countdown, because a disk was just detected
270         setCountdown(COUNTDOWN);
271
272         //update state
273         $state = 5;
274         storeData($state, 'state', 0);
275         unset($state);
276
277         runningTimerReset();
278     }
279
280     unset($position);
281
282     //check if a white disk is at the colour detector
283     $colour = getButtonPressed(6);
284     if ($colour == 1) {
285         //move the sorter up so the disk goes to the correct box
286         $temp = 10;
287         storeData($temp, 'outputs', HBRIDGE0);
288
289         //stop the feeder engine
290         $temp = 0;
291         storeData($temp, 'outputs', FEEDERENGINE);
292         unset($temp);
293
294         //update state
295         $state = 6;
296         storeData($state, 'state', 0);
297         unset($state);
298
299         motorUp();
300     }
301     unset($colour);
302 }
303 //Loop
304 runningWait();
305 }
306
307 //state 5
308 function runningTimerReset()
309 {
310     timerManage();
311
312     //update state
313     $state = 4;
314     storeData($state, 'state', 0);
315     unset($state);
316
317     runningWait();
318 }
319 }
320
321 //state 6
322 function motorUp()
323 {
324     global $sleep;
325     timerManage();
326
327     //check if we need to pause
328     $startStop = getButtonPressed(0);
329     if ($startStop == 1) {
330         //stop the feeder engine
331         $temp = 0;
332         storeData($temp, 'outputs', FEEDERENGINE);
333         unset($temp);
334
335         //exit after 1 rotation of the belt
336         setCountdown(BELT * 10);
337
338         //update the state
339         $state = 10;
340         storeData($state, 'state', 0);
341         unset($state);
342
343         motorUpTimer();
344     }
345     unset($startStop);
346
347
348 //check if the sorter push button is pressed
349 $push = getButtonPressed(5);
350 if ($push == 1) {
351     //stop the sorter engine, because its at its highest position
352     $temp = 0;
353     storeData($temp, 'outputs', HBRIDGE0);
354     unset($temp);
355
356     //update state
357     $state = 7;
358     storeData($state, 'state', 0);
359     unset($state);
360
361     //set sleep for the next function
362     $sleep = 0;
363
364     whiteWait();
365 }
366 unset($push);
367
368 //Loop
369 motorUp();
370 }
371
372 //state 7
373 function whiteWait()
374 {
375     global $sleep;
376     timerManage();
377
378     //we are waiting for the white disk to be sorted
379     if ($sleep == SORT) {
380         //start moving the sorter down
381         $temp = 10;
382         storeData($temp, 'outputs', HBRIDGE1);
383         unset($temp);
384
385         //make sure the timerinterrupt is correct
386         setCountdown(COUNTDOWN);
387
388         //update state
389         $state = 8;
390         storeData($state, 'state', 0);
391         unset($state);
392
393         //reset sleep for the next function
394         $sleep = 0;
395         motorDown();
396     }
397
398     //check if we need to pause
399     $startStop = getButtonPressed(0);
400     if ($startStop == 1) {
401         //stop the feeder engine
402         $temp = 0;
403         storeData($temp, 'outputs', FEEDERENGINE);
404         unset($temp);
405
406         //exit after 1 rotation of the belt
407         setCountdown(BELT * 10);
408
409         //update the state
410         $state = 11;
411         storeData($state, 'state', 0);
412         unset($state);
413
414         whiteWaitTimer();
415     }
416     unset($startStop);
417
418     //Loop
419     $sleep++;
420     whiteWait();
421 }
422 }
423
424 //state 8
425 function motorDown()
426 {
427     global $sleep;
428     timerManage();
429
430     //check if a white disk is at the colour detector
431     $colour = getButtonPressed(6);
432     if ($colour == 1) {
433         //move the sorter up so the disk goes to the correct box
434         $temp=0;
435         storeData($temp, 'outputs', HBRIDGE1);
436         $temp = 10;
437         storeData($temp, 'outputs', HBRIDGE0);
438         unset($temp);
439
440         //update state
441         $state = 6;
442         storeData($state, 'state', 0);
443         $sleep=0;
444         unset($state);
445
446         motorUp();
447     }
448     unset($colour);
449
450     //the sorter is moving down, we are waiting for that to complete
451     if ($sleep == TIMEMOTORDOWN) {
452         //stop the sorter, its where it should be
453         $temp = 0;
454         storeData($temp, 'outputs', HBRIDGE1);
455         $temp = 7;
456         storeData($temp, 'outputs', FEEDERENGINE);
457         unset($temp);
458
459         //update state
460         $state = 4;
461         storeData($state, 'state', 0);
462         //reset sleep for the next function
463         $sleep = 0;
464     }
465 }

```

```

466     unset($state);
467     runningWait();
468 }
469
470 //check if we need to pause
471 $startStop = getButtonPressed(0);
472 if ($startStop == 1) {
473     //stop the feeder engine
474     $temp = 0;
475     storeData($temp, 'outputs', FEEDERENGINE);
476     unset($temp);
477
478     //exit after 1 rotation of the belt
479     setCountdown(BELT * 10);
480
481     //update the state
482     $state = 12;
483     storeData($state, 'state', 0);
484     unset($state);
485
486     motorDownTimer();
487 }
488 unset($startStop);
489
490 //Loop
491 $sleep++;
492 motorDown();
493 }
494
495 //state 9
496 function runningTimer()
497 {
498     timerManage();
499
500     //update state
501     $state = 13;
502     storeData($state, 'state', 0);
503     unset($state);
504
505     runningStop();
506 }
507
508 //state 10
509 function motorUpTimer()
510 {
511     timerManage();
512
513     //update state
514     $state = 14;
515     storeData($state, 'state', 0);
516     unset($state);
517
518     motorUpStop();
519 }
520
521 //state 11
522 function whiteWaitTimer()
523 {
524     timerManage();
525
526     //update state
527     $state = 15;
528     storeData($state, 'state', 0);
529     unset($state);
530
531     whiteWaitStop();
532 }
533
534 //state 12
535 function motorDownTimer()
536 {
537     timerManage();
538
539     //update state
540     $state = 16;
541     storeData($state, 'state', 0);
542     unset($state);
543
544     motorDownStop();
545 }
546
547 //state 13
548 function runningStop()
549 {
550     timerManage();
551
552     //check if a white disk is at the colour detector
553     $colour = getButtonPressed(6);
554     if ($colour == 1) {
555         //stop the sorter engine, because its at its highest position
556         $temp = 10;
557         storeData($temp, 'outputs', HBRIDGE0);
558
559         //stop the feeder engine
560         $temp = 0;
561         storeData($temp, 'outputs', FEEDERENGINE);
562         unset($temp);
563
564         //update state
565         $state = 10;
566         storeData($state, 'state', 0);
567         unset($state);
568
569         motorUpStop();
570     }
571     unset($colour);
572
573     //Loop
574     runningStop();
575 }
576
577 //state 14
578 function motorUpStop()
579 {
580     timerManage();
581
582     //check if the sorter push button is pressed
583
584
585     $push = getButtonPressed(5);
586     if ($push == 1) {
587         //stop the engine of the sorter
588         $temp = 0;
589         storeData($temp, 'outputs', HBRIDGE0);
590         unset($temp);
591
592         //update state
593         $state = 11;
594         storeData($state, 'state', 0);
595         unset($state);
596
597         whiteWaitStop();
598     }
599     unset($push);
600
601     //Loop
602     motorUpStop();
603 }
604
605 //state 15
606 function whiteWaitStop()
607 {
608     global $sleep;
609     timerManage();
610
611     //check if the white disk has been sorted
612     if ($sleep == SORT) {
613         //it has, so lets start moving the sorter down
614         $temp = 10;
615         storeData($temp, 'outputs', HBRIDGE1);
616         $temp = 0;
617         storeData($temp, 'outputs', FEEDERENGINE);
618         unset($temp);
619
620         //update state
621         $state = 12;
622         storeData($state, 'state', 0);
623         unset($state);
624
625         $sleep = 0;
626         motorDownStop();
627     }
628
629     //Loop
630     $sleep++;
631     whiteWaitStop();
632 }
633
634 //state 16
635 function motorDownStop()
636 {
637     global $sleep;
638     timerManage();
639
640     //check if the sorter has moved down
641     if ($sleep == TIME MOTORDOWN) {
642         //it has, so lets stop it
643         $temp = 0;
644         storeData($temp, 'outputs', HBRIDGE1);
645         unset($temp);
646
647         //update the state
648         $state = 9;
649         storeData($state, 'state', 0);
650         unset($state);
651
652         $sleep = 0;
653         runningStop();
654     }
655
656     //Loop
657     $sleep++;
658     motorDownStop();
659 }
660
661 //not a state
662 function timerInterrupt()
663 {
664     timerManage();
665     //show that we are in the timer interrupt
666     $temp = 5;
667     display($temp, 'display');
668
669     //start moving the sorter up, to start the calibration
670     $temp = 10;
671     storeData($temp, 'outputs', HBRIDGE0);
672
673     //stop the rest
674     $temp = 0;
675     storeData($temp, 'outputs', LENS LAMPPOSITION);
676     storeData($temp, 'outputs', LENS LAMP SORTER);
677     storeData($temp, 'outputs', LED STATE INDICATOR);
678     storeData($temp, 'outputs', DISPLAY);
679     storeData($temp, 'outputs', CONVEYORBELT);
680     storeData($temp, 'outputs', FEEDERENGINE);
681
682     //reset, because we will no longer be in timerInterrupt
683     display($temp, 'display');
684     unset($temp);
685
686     //go back to initial
687     $temp = getData('offset', 0);
688     $temp2 = getFuncLocation('initial');
689     $temp += $temp2;
690
691     addStackPointer(2);
692     pushStack($temp);
693     addStackPointer(-1);
694 }
695
696 //not a state
697 function abort()
698 {
699     //free some memory
700     unset($engines);
701 }

```

```

704 //prevent timerinterrupt
705 setCountdown(1000);
706 $temp = getData('stackPointer', 0);
707 setStackPointer($temp);
708
709 //stop everything
710 $temp = 0;
711 storeData($temp, 'outputs', HBRIDGE1);
712 storeData($temp, 'outputs', HBRIDGE0);
713 storeData($temp, 'outputs', LENSAMPLPOSITION);
714 storeData($temp, 'outputs', LENSAMPLSORTER);
715 storeData($temp, 'outputs', LEDSTATEINDICATOR);
716 storeData($temp, 'outputs', DISPLAY);
717 storeData($temp, 'outputs', CONVEYORBELT);
718 storeData($temp, 'outputs', FEEDERENGINE);
719 unset($temp);
720
721 //apply the changes to actually stop it
722 timerManage();
723
724 //update the state
725 $state = 17;
726 storeData($state, 'state', 0);
727
728
729 //show we aborted
730 $state = 7;
731 display($state, 'leds2', 0);
732 unset($state);
733
734 aborted();
735 }
736
737 //state 17
738 function aborted()
739 {
740     //prevent timer interrupt
741     setCountdown(1000);
742     timerManage();
743     //check if we can start again
744     $startStop = getButtonPressed(0);
745     if ($startStop == 1) {
746         //start moving the sorter up, to start the calibration
747         $temp = 10;
748         storeData($temp, 'outputs', HBRIDGE0);
749         unset($temp);
750
751         //update the state
752         $state = 0;
753         storeData($state, 'state', 0);
754         unset($state);
755
756         initial();
757     }
758     unset($startStop);
759     aborted();
760 }
761 }
762
763 //not a state
764 function timerManage()
765 {
766     global $location, $counter, $engine, $sleep;
767
768     if ($location == 0) {
769         $engines = 0;
770     }
771
772     //makes sure that when $counter >12 it will reset to 0
773     mod(12, $counter);
774
775     //get the voltage of output $location
776     $voltage = getData('outputs', $location);
777
778     //power up the output when it needs to
779     if ($voltage > $counter) {
780         $voltage = $location;
781         $voltage = pow(2, $voltage);
782         $engines += $voltage;
783     }
784
785     //check if we did all outputs
786     if ($location == 7) {
787         //actually output the result
788         sleep(1);
789         display($engines, 'leds');
790
791         unset($voltage);
792         //check if abort is pressed
793         $abort = getButtonPressed(1);
794         if ($abort == 1) {
795             abort();//STOP THE MACHINE!
796         }
797         unset($abort);
798
799         //check if we are in a new iteration
800         if ($counter == 6) {
801             //set the first part of the display
802             $temp = getData('state', 0);
803             mod(10, $temp);
804             display($temp, 'display', 1);
805             unset($temp);
806         }
807         //check if we are at the end of the iteration
808         if ($counter == 11) {
809             //set the second part of the display;
810             pushStack($sleep);
811
812             $temp = getData('state', 0);
813             //get the last digit of the state
814             //we have no variables left, so we use $sleep
815
816             $sleep = $temp;
817             mod(10, $sleep);
818             $temp -= $sleep;
819             $temp /= 10;
820             //display the last digit
821             display($temp, 'display', 2);
822
823             pullStack($sleep);
824             unset($temp);
825         }
826
827         //set the variables for the next run
828         $engines = 0;
829         $location = 0;
830         $counter++;
831
832         //and return to where we came from
833         return;
834     }
835 }
836
837 //Loop
838 $location++;
839 branch('timerManage');
840 }

```

Appendix 7: Assembly Program

```
1 @DATA
2 offset DS 1
3 stackPointer DS 1
4 outputs DS 12
5 state DS 1
6
7 @CODE
8
9         TIMEMOTORDOWN EQU 150
10        BELT EQU 2000
11        BELTROUND EQU 2000
12        SORT EQU 200
13        COUNTDOWN EQU 30000
14        LENSAMPLPOSITION EQU 2
15        LENSAMPLSORTER EQU 6
16        HBRIDGE0 EQU 0
17        HBRIDGE1 EQU 1
18        CONVEYORBELT EQU 7
19        FEEDERENGINE EQU 3
20        DISPLAY EQU 8
21        LEDSTATEINDICATOR EQU 9
22 begin:  BRA main
23
24
25
26 _timer:      MULS R5 10
27              PUSH R4
28              LOAD R4 R5
29              LOAD R5 -16
30              LOAD R5 [R5+13]
31              SUB R5 R4
32              LOAD R4 -16
33 _wait:       CMP R5 [R4+13]
34              BMI _wait
35              PULL R4
36              RTS
37
38 _pressed:    PUSH R4
39              PUSH R5
40              LOAD R4 R3
41              LOAD R5 2
42              BRS _pow
43              LOAD R3 R5
44              LOAD R5 -16
45              LOAD R4 [R5+7]
46              DIV R4 R3
47              MOD R4 2
48
49              PUSH R4
50              ADD SP 1
51
52              PULL R5
53              PULL R4
54
55              RTS
56
57 _pow:        CMP R4 0
58              BEQ _pow1
59              CMP R4 1
60              BEQ _powR
61              PUSH R3
62              PUSH R4
63              SUB R4 1
64              LOAD R3 R5
65 _powLoop:    MULS R5 R3
66              SUB R4 1
67              CMP R4 0
68              BEQ _powReturn
69              BRA _powLoop
70 _powReturn:  PULL R4
71              PULL R3
72              RTS
73 _pow1:       LOAD R5 1
74              RTS
75 _powR:       RTS
76
77
78 _Hex7Seg:    BRS _Hex7Seg_bgn
79 _Hex7Seg_tbl: CONS %01111110
80              CONS %00110000
81              CONS %01101101
82              CONS %01111001
83              CONS %00110011
84              CONS %01011011
85              CONS %01011111
86              CONS %01110000
87              CONS %01111111
```

;sleep

; Compare the timer to 0

;make sure all vars are the same at the end

;the result
;decrease the SP so we get the correct pulls

;display
; push address(tbl) onto stack and proceed at bg
; 7-segment pattern for '0'
; 7-segment pattern for '1'
; 7-segment pattern for '2'
; 7-segment pattern for '3'
; 7-segment pattern for '4'
; 7-segment pattern for '5'
; 7-segment pattern for '6'
; 7-segment pattern for '7'
; 7-segment pattern for '8'


```

88      CONS %01111011      ; 7-segment pattern for '9'
89      CONS %01110111      ; 7-segment pattern for 'A'
90      CONS %00011111      ; 7-segment pattern for 'b'
91      CONS %01001110      ; 7-segment pattern for 'C'
92      CONS %00111101      ; 7-segment pattern for 'd'
93      CONS %01001111      ; 7-segment pattern for 'E'
94      CONS %01000111      ; 7-segment pattern for 'F'
95      _Hex7Seg_bgn:      AND R5 %01111      ; R0 = R0 MOD 16 , just to be safe...
96      LOAD R4 [SP++]      ; R4 = address(tbl) (retrieve from stack)
97      LOAD R4 [R4+R5]      ; R4 = tbl[R0]
98      LOAD R5 -16
99      STOR R4 [R5+8]      ; and place this in the Display Element
100     RTS
101     main:      STOR R5 [GB +offset + 0]      ;storeData(R5, 'offset', 0)
102      LOAD R0 timerInterrupt      ;installCountdown('timerInterrupt')
103      ADD R0 R5
104      LOAD R1 16
105      STOR R0 [R1]
106
107      LOAD R5 -16
108
109
110
111      LOAD R0 0      ; Set the timer to 0
112      SUB R0 [R5+13]
113      STOR R0 [R5+13]
114      STOR SP [GB +stackPointer + 0]      ;storeData(SP, 'stackPointer', 0)
115      LOAD R0 0      ;$counter = 0
116      LOAD R1 0      ;$location = 0
117      LOAD R2 0      ;$sleep = 0
118      LOAD R3 0      ;$temp = 0
119      STOR R3 [GB +outputs + HBRIDGE1]      ;storeData($temp, 'outputs', HBRIDGE1)
120      STOR R3 [GB +outputs + LENSAMPPOSITION]      ;storeData($temp, 'outputs', LENSAMPPOSITION)
121      STOR R3 [GB +outputs + LENSAMPSORTER]      ;storeData($temp, 'outputs', LENSAMPSORTER)
122      STOR R3 [GB +outputs + LEDSTATEINDICATOR]      ;storeData($temp, 'outputs', LEDSTATEINDICATOR)
123      STOR R3 [GB +outputs + DISPLAY]      ;storeData($temp, 'outputs', DISPLAY)
124      STOR R3 [GB +outputs + CONVEYORBELT]      ;storeData($temp, 'outputs', CONVEYORBELT)
125      STOR R3 [GB +outputs + FEEDERENGINE]      ;storeData($temp, 'outputs', FEEDERENGINE)
126      LOAD R4 0      ;$state = 0
127      STOR R4 [GB +state + 0]      ;storeData($state, 'state', 0)
128      LOAD R3 10      ;$temp = 10
129      STOR R3 [GB +outputs + HBRIDGE0]      ;storeData($temp, 'outputs', HBRIDGE0)
130
131      BRA initial      ;unset($temp, $state)
132                                     ;initial()
133     timerInterrupt:      BRS timerManage      ;timerManage()
134      LOAD R3 5      ;$temp = 5
135      PUSH R5      ;display($temp, 'display')
136      PUSH R4
137      LOAD R5 R3
138      BRS _Hex7Seg
139      LOAD R4 %0000001
140      STOR R4 [R5+9]
141      PULL R4
142      PULL R5
143      LOAD R3 10      ;$temp = 10
144      STOR R3 [GB +outputs + HBRIDGE0]      ;storeData($temp, 'outputs', HBRIDGE0)
145      LOAD R3 0      ;$temp = 0
146      STOR R3 [GB +outputs + LENSAMPPOSITION]      ;storeData($temp, 'outputs', LENSAMPPOSITION)
147      STOR R3 [GB +outputs + LENSAMPSORTER]      ;storeData($temp, 'outputs', LENSAMPSORTER)
148      STOR R3 [GB +outputs + LEDSTATEINDICATOR]      ;storeData($temp, 'outputs', LEDSTATEINDICATOR)
149      STOR R3 [GB +outputs + DISPLAY]      ;storeData($temp, 'outputs', DISPLAY)
150      STOR R3 [GB +outputs + CONVEYORBELT]      ;storeData($temp, 'outputs', CONVEYORBELT)
151      STOR R3 [GB +outputs + FEEDERENGINE]      ;storeData($temp, 'outputs', FEEDERENGINE)
152      PUSH R5      ;display($temp, 'display')
153      PUSH R4
154      LOAD R5 R3
155      BRS _Hex7Seg
156      LOAD R4 %0000001
157      STOR R4 [R5+9]
158      PULL R4
159      PULL R5
160
161      LOAD R3 [ GB + offset + 0 ]      ;unset($temp)
162      LOAD R4 initial      ;$temp = getData('offset', 0)
163      ADD R3 R4      ;$temp2 = getFuncLocation('initial')
164      ADD SP 2      ;$temp += $temp2
165      PUSH R3      ;addStackPointer(2)
166      ADD SP -1      ;pushStack($temp)
167      RTE      ;addStackPointer(-1)
168
169     initial:      LOAD R3 0      ;$temp = 0
170      PUSH R5      ;display($temp, 'Leds2')
171      LOAD R5 -16
172      STOR R3 [R5+10]
173      PULL R5
174      LOAD R3 [ GB + stackPointer + 0 ]      ;$temp = getData('stackPointer', 0)
175      LOAD SP R3      ;setStackPointer($temp)
176      BRS timerManage      ;timerManage()
177      PUSH R3      ;$push = getButtonPressed(5)
178      LOAD R3 5
179      BRS _pressed

```

```

179          PULL R3
180          SUB SP 5
181          PULL R4
182          ADD SP 4
183          CMP R4 1
184          BEQ conditional0
185      return0:
186          BRA initial
187
188
189      conditional0:
190          LOAD R3 0
191          STOR R3 [GB +outputs + HBRIDGE0]
192          LOAD R3 10
193          STOR R3 [GB +outputs + HBRIDGE1]
194          LOAD R3 1
195          STOR R3 [GB +state + 0]
196
197          LOAD R2 0
198          BRA calibrateSorter
199      calibrateSorter:
200          BRS timerManage
201          CMP R2 TIMEMOTORDOWN
202          BEQ conditional1
203      return1:
204          ADD R2 1
205          BRA calibrateSorter
206
207      conditional1:
208          LOAD R3 0
209          STOR R3 [GB +outputs + HBRIDGE1]
210          LOAD R4 2
211          STOR R4 [GB +state + 0]
212
213          LOAD R2 0
214          BRA resting
215
216      resting:
217          BRS timerManage
218          PUSH R3
219          LOAD R3 0
220          BRS _pressed
221          PULL R3
222          SUB SP 5
223          PULL R4
224          ADD SP 4
225          CMP R4 1
226          BEQ conditional2
227      return2:
228          BRA resting
229
230      conditional2:
231          LOAD R3 12
232          STOR R3 [GB +outputs + LENSAMPPOSITION]
233
234          BRS timerManage
235          PUSH R5
236          LOAD R5 1000
237          BRS _timer
238          PULL R5
239          LOAD R3 12
240          STOR R3 [GB +outputs + LENSAMPSORTER]
241
242          BRS timerManage
243          PUSH R5
244          LOAD R5 2000
245          BRS _timer
246          PULL R5
247          LOAD R3 9
248          STOR R3 [GB +outputs + CONVEYORBELT]
249          LOAD R3 9
250          STOR R3 [GB +outputs + FEEDERENGINE]
251
252          PUSH R5 ;reset timer
253          PUSH R4
254          LOAD R5 -16
255          LOAD R4 0
256          SUB R4 [R5+13]
257          STOR R4 [R5+13]
258          LOAD R4 COUNTDOWN
259          STOR R4 [R5+13]
260          PULL R4
261          PULL R5
262          SETI 8
263          LOAD R3 3
264          STOR R3 [GB +state + 0]
265
266          BRA running
267
268      running:
269          BRS timerManage
270          PUSH R3
271          LOAD R3 0
272          BRS _pressed
273          PULL R3

```

```

; if ($push == 1) {
; unset($push)
; initial()

; if ($push == 1) {
; $temp = 0
; storeData($temp, 'outputs', HBRIDGE0)
; $temp = 10
; storeData($temp, 'outputs', HBRIDGE1)
; $temp = 1
; storeData($temp, 'state', 0)
; unset($temp)
; $sleep = 0
; calibrateSorter()

; timerManage()
; if ($sleep == TIMEMOTORDOWN) {
; $sleep+=1
; calibrateSorter()

; if ($sleep == TIMEMOTORDOWN) {
; $temp = 0
; storeData($temp, 'outputs', HBRIDGE1)
; $state = 2
; storeData($state, 'state', 0)
; unset($state)
; $sleep = 0
; resting()

; timerManage()
; $startStop = getButtonPressed(0)

; if ($startStop == 1) {
; unset($startStop)
; resting()

; if ($startStop == 1) {
; $temp = 12
; storeData($temp, 'outputs', LENSAMPPOSITION)
; unset($temp)
; timerManage()
; sleep(1000)

; $temp = 12
; storeData($temp, 'outputs', LENSAMPSORTER)
; unset($temp)
; timerManage()
; sleep(2000)

; $temp = 9
; storeData($temp, 'outputs', CONVEYORBELT)
; $temp = 9
; storeData($temp, 'outputs', FEEDERENGINE)
; unset($temp)
; setCountdown(COUNTDOWN)

; set timer

; startCountdown()
; $state = 3
; storeData($state, 'state', 0)
; unset($state)
; running()

; timerManage()
; $startStop = getButtonPressed(0)

```

```

270 SUB SP 5
271 PULL R3
272 ADD SP 4
273 CMP R3 1
274 BEQ conditional3
275 return3:
276 PUSH R3
277 LOAD R3 7
278 BRS _pressed
279 PULL R3
280 SUB SP 5
281 PULL R3
282 ADD SP 4
283 CMP R3 1
284 BEQ conditional4
285 return4:
286 BRA running
287
288
289 conditional3:
290 LOAD R4 0
291 STOR R4 [GB +outputs + FEEDERENGINE]
292
293 PUSH R5 ;reset timer
294 PUSH R4
295 LOAD R5 -16
296 LOAD R4 0
297 SUB R4 [R5+13]
298 STOR R4 [R5+13]
299 LOAD R4 BELT * 10
300 STOR R4 [R5+13]
301 PULL R4
302 PULL R5
303 LOAD R4 9
304 STOR R4 [GB +state + 0]
305 BRA runningTimer
306
307
308 conditional4:
309 PUSH R5 ;reset timer
310 PUSH R4
311 LOAD R5 -16
312 LOAD R4 0
313 SUB R4 [R5+13]
314 STOR R4 [R5+13]
315 LOAD R4 COUNTDOWN
316 STOR R4 [R5+13]
317 PULL R4
318 PULL R5
319 LOAD R4 4
320 STOR R4 [GB +state + 0]
321 BRA runningWait
322
323 runningWait:
324 BRS timerManage
325 PUSH R3
326 LOAD R3 0
327 BRS _pressed
328 PULL R3
329 SUB SP 5
330 PULL R3
331 ADD SP 4
332 CMP R3 1
333 BEQ conditional5
334 return5:
335 PUSH R3
336 LOAD R3 7
337 BRS _pressed
338 PULL R3
339 SUB SP 5
340 PULL R3
341 ADD SP 4
342 CMP R3 0
343 BEQ conditional6
344 return6:
345 PUSH R3
346 LOAD R3 6
347 BRS _pressed
348 PULL R3
349 SUB SP 5
350 PULL R3
351 ADD SP 4
352 CMP R3 1
353 BEQ conditional7
354 return7:
355 BRA runningWait
356
357 conditional5:
358 LOAD R4 0
359 STOR R4 [GB +outputs + FEEDERENGINE]
360 PUSH R5 ;reset timer

```

```

; if ($startStop == 1) {
;unset($startStop)
;$position = getButtonPressed(7)

; if ($position == 1) {
;unset($position)
;running()

; if ($startStop == 1) {
;$temp = 0
;storeData($temp, 'outputs', FEEDERENGINE)
;unset($temp)
;setCountdown(BELT * 10)

;set timer

;$state = 9
;storeData($state, 'state', 0)
;unset($state)
;runningTimer()

; if ($position == 1) {
;setCountdown(COUNTDOWN)

;set timer

;$state = 4
;storeData($state, 'state', 0)
;unset($state)
;runningWait()

;timerManage()
;$startStop = getButtonPressed(0)

; if ($startStop == 1) {
;unset($startStop)
;$position = getButtonPressed(7)

; if ($position == 0) {
;unset($position)
;$colour = getButtonPressed(6)

; if ($colour == 1) {
;unset($colour)
;runningWait()

; if ($startStop == 1) {
;$temp = 0
;storeData($temp, 'outputs', FEEDERENGINE)
;unset($temp)
;setCountdown(BELT * 10)

```

```

361      PUSH R4
362      LOAD R5 -16
363      LOAD R4 0
364      SUB R4 [R5+13]
365      STOR R4 [R5+13]          ;set timer
366      LOAD R4 BELT * 10
367      STOR R4 [R5+13]
368      PULL R4
369      PULL R5
370      LOAD R4 9
371      STOR R4 [GB +state + 0]  ;$state = 9
372                                  ;storeData($state, 'state', 0)
373      BRA runningTimer        ;unset($state)
374                                  ;runningTimer()
375
376 conditional6:      PUSH R5 ;reset timer
377      PUSH R4
378      LOAD R5 -16
379      LOAD R4 0
380      SUB R4 [R5+13]
381      STOR R4 [R5+13]          ;set timer
382      LOAD R4 COUNTDOWN
383      STOR R4 [R5+13]
384      PULL R4
385      PULL R5
386      LOAD R4 5
387      STOR R4 [GB +state + 0]  ;$state = 5
388                                  ;storeData($state, 'state', 0)
389      BRA runningTimerReset   ;unset($state)
390                                  ;runningTimerReset()
391
392 conditional7:      LOAD R4 10
393      STOR R4 [GB +outputs + HBRIDGE0]
394      LOAD R4 0
395      STOR R4 [GB +outputs + FEEDERENGINE]
396                                  ;if ($colour == 1) {
397                                  ;$temp = 10
398                                  ;storeData($temp, 'outputs', HBRIDGE0)
399                                  ;$temp = 0
400                                  ;storeData($temp, 'outputs', FEEDERENGINE)
401                                  ;unset($temp)
402                                  ;$state = 6
403                                  ;storeData($state, 'state', 0)
404                                  ;unset($state)
405      BRA motorUp            ;motorUp()
406
407 motorUp:          BRS timerManage
408                  PUSH R3
409                  LOAD R3 0
410                  BRS _pressed
411                  PULL R3
412                  SUB SP 5
413                  PULL R3
414                  ADD SP 4
415                  CMP R3 1
416                  BEQ conditional8
417
418 return8:          PUSH R3
419                  LOAD R3 5
420                  BRS _pressed
421                  PULL R3
422                  SUB SP 5
423                  PULL R3
424                  ADD SP 4
425                  CMP R3 1
426                  BEQ conditional9
427
428 return9:          BRA motorUp
429
430 conditional8:      LOAD R4 0
431      STOR R4 [GB +outputs + FEEDERENGINE]
432
433      PUSH R5 ;reset timer
434      PUSH R4
435      LOAD R5 -16
436      LOAD R4 0
437      SUB R4 [R5+13]
438      STOR R4 [R5+13]          ;set timer
439      LOAD R4 BELT * 10
440      STOR R4 [R5+13]
441      PULL R4
442      PULL R5
443      LOAD R4 10
444      STOR R4 [GB +state + 0]  ;$state = 10
445                                  ;storeData($state, 'state', 0)
446      BRA motorUpTimer       ;unset($state)
447                                  ;motorUpTimer()
448
449 conditional9:      LOAD R4 0
450      STOR R4 [GB +outputs + HBRIDGE0]
451                                  ;if ($push == 1) {
452                                  ;$temp = 0
453                                  ;storeData($temp, 'outputs', HBRIDGE0)
454                                  ;unset($temp)
455                                  ;$state = 7
456                                  ;storeData($state, 'state', 0)
457                                  ;unset($state)
458                                  ;$sleep = 0

```

```

452             BRA whiteWait                               ;whiteWait()
453
454 whiteWait:      BRS timerManage                         ;timerManage()
455                 CMP R2 SORT                             ;if ($sleep == SORT) {
456                 BEQ conditional10                       ;$startStop = getButtonPressed(0)
457 return10:       PUSH R3
458                 LOAD R3 0
459                 BRS _pressed
460                 PULL R3
461                 SUB SP 5
462                 PULL R3
463                 ADD SP 4
464                 CMP R3 1
465                 BEQ conditional11                       ;if ($startStop == 1) {
466 return11:       ADD R2 1
467                 BRA whiteWait                           ;unset($startStop)
468                                                         ;$sleep+=1
469                                                         ;whiteWait()
470
471 conditional10:   LOAD R3 10
472                 STOR R3 [GB +outputs + HBRIDGE1]        ;if ($sleep == SORT) {
473                                                         ;$temp = 10
474                 PUSH R5 ;reset timer                    ;storeData($temp, 'outputs', HBRIDGE1)
475                 PUSH R4
476                 LOAD R5 -16
477                 LOAD R4 0
478                 SUB R4 [R5+13]
479                 STOR R4 [R5+13]                         ;unset($temp)
480                 LOAD R4 COUNTDOWN                      ;setCountdown(COUNTDOWN)
481                 STOR R4 [R5+13]
482                 PULL R4
483                 PULL R5
484                 LOAD R3 8
485                 STOR R3 [GB +state + 0]                 ;$state = 8
486                                                         ;storeData($state, 'state', 0)
487                                                         ;unset($state)
488                 LOAD R2 0
489                 BRA motorDown                           ;$sleep = 0
490                                                         ;motorDown()
491
492 conditional11:   LOAD R4 0
493                 STOR R4 [GB +outputs + FEEDERENGINE]    ;if ($startStop == 1) {
494                                                         ;$temp = 0
495                 PUSH R5 ;reset timer                    ;storeData($temp, 'outputs', FEEDERENGINE)
496                 PUSH R4
497                 LOAD R5 -16
498                 LOAD R4 0
499                 SUB R4 [R5+13]
500                 STOR R4 [R5+13]                         ;unset($temp)
501                 LOAD R4 BELT * 10
502                 STOR R4 [R5+13]                         ;setCountdown(BELT * 10)
503                 PULL R4
504                 PULL R5
505                 LOAD R4 11
506                 STOR R4 [GB +state + 0]                 ;$state = 11
507                                                         ;storeData($state, 'state', 0)
508                                                         ;unset($state)
509                 BRA whiteWaitTimer                     ;whiteWaitTimer()
510
511 whiteWaitTimer: BRS timerManage                         ;timerManage()
512                 LOAD R3 15
513                 STOR R3 [GB +state + 0]                 ;$state = 15
514                                                         ;storeData($state, 'state', 0)
515                                                         ;unset($state)
516 whiteWaitStop:  BRS timerManage                         ;timerManage()
517                 CMP R2 SORT                             ;if ($sleep == SORT) {
518                 BEQ conditional12                       ;$startStop = getButtonPressed(0)
519 return12:       ADD R2 1
520                 BRA whiteWaitStop                       ;unset($startStop)
521                                                         ;$sleep+=1
522                                                         ;whiteWaitStop()
523
524 conditional12:   LOAD R3 10
525                 STOR R3 [GB +outputs + HBRIDGE1]        ;if ($sleep == SORT) {
526                 LOAD R3 0
527                 STOR R3 [GB +outputs + FEEDERENGINE]    ;$temp = 10
528                                                         ;storeData($temp, 'outputs', HBRIDGE1)
529                                                         ;$temp = 0
530                                                         ;storeData($temp, 'outputs', FEEDERENGINE)
531                                                         ;unset($temp)
532                                                         ;$state = 12
533                                                         ;storeData($state, 'state', 0)
534                                                         ;unset($state)
535                                                         ;$sleep = 0
536                                                         ;motorDownStop()
537
538 motorDownStop:  BRS timerManage                         ;timerManage()
539                 CMP R2 TIMEMOTORDOWN                   ;if ($sleep == TIMEMOTORDOWN) {
540                 BEQ conditional13                       ;$startStop = getButtonPressed(0)
541 return13:       ADD R2 1
542                 BRA motorDownStop                       ;unset($startStop)
543                                                         ;$sleep+=1
544                                                         ;motorDownStop()
545
546 conditional13:   LOAD R3 0
547                 STOR R3 [GB +outputs + HBRIDGE1]        ;if ($sleep == TIMEMOTORDOWN) {
548                                                         ;$temp = 0
549                                                         ;storeData($temp, 'outputs', HBRIDGE1)
550                                                         ;unset($temp)

```

```

543          LOAD R3 9
544          STOR R3 [GB +state + 0]
545
546          LOAD R2 0
547          BRA runningStop
548
549 runningStop: BRS timerManage
550             PUSH R3
551             LOAD R3 6
552             BRS _pressed
553             PULL R3
554             SUB SP 5
555             PULL R3
556             ADD SP 4
557             CMP R3 1
558             BEQ conditional14
559 return14:
560             BRA runningStop
561
562
563 conditional14: LOAD R4 10
564                STOR R4 [GB +outputs + HBRIDGE0]
565                LOAD R4 0
566                STOR R4 [GB +outputs + FEEDERENGINE]
567
568                LOAD R4 10
569                STOR R4 [GB +state + 0]
570
571                BRA motorUpStop
572
573 motorUpStop: BRS timerManage
574              PUSH R3
575              LOAD R3 5
576              BRS _pressed
577              PULL R3
578              SUB SP 5
579              PULL R3
580              ADD SP 4
581              CMP R3 1
582              BEQ conditional15
583 return15:
584              BRA motorUpStop
585
586
587 conditional15: LOAD R4 0
588                STOR R4 [GB +outputs + HBRIDGE0]
589
590                LOAD R4 11
591                STOR R4 [GB +state + 0]
592
593                BRA whiteWaitStop
594
595 motorDown: BRS timerManage
596            PUSH R3
597            LOAD R3 6
598            BRS _pressed
599            PULL R3
600            SUB SP 5
601            PULL R3
602            ADD SP 4
603            CMP R3 1
604            BEQ conditional16
605 return16:
606            CMP R2 TIMEMOTORDOWN
607            BEQ conditional17
608 return17: PUSH R3
609            LOAD R3 0
610            BRS _pressed
611            PULL R3
612            SUB SP 5
613            PULL R3
614            ADD SP 4
615            CMP R3 1
616            BEQ conditional18
617 return18:
618            ADD R2 1
619            BRA motorDown
620
621
622 conditional16: LOAD R4 0
623                STOR R4 [GB +outputs + HBRIDGE1]
624                LOAD R4 10
625                STOR R4 [GB +outputs + HBRIDGE0]
626
627                LOAD R4 6
628                STOR R4 [GB +state + 0]
629                LOAD R2 0
630
631                BRA motorUp
632
633

```

```

;state = 9
;storeData($state, 'state', 0)
;unset($state)
;$sleep = 0
;runningStop()

;timerManage()
;$colour = getButtonPressed(6)

;if ($colour == 1) {
;unset($colour)
;runningStop()

;if ($colour == 1) {
;$temp = 10
;storeData($temp, 'outputs', HBRIDGE0)
;$temp = 0
;storeData($temp, 'outputs', FEEDERENGINE)
;unset($temp)
;$state = 10
;storeData($state, 'state', 0)
;unset($state)
;motorUpStop()

;timerManage()
;$push = getButtonPressed(5)

;if ($push == 1) {
;unset($push)
;motorUpStop()

;if ($push == 1) {
;$temp = 0
;storeData($temp, 'outputs', HBRIDGE0)
;unset($temp)
;$state = 11
;storeData($state, 'state', 0)
;unset($state)
;whiteWaitStop()

;timerManage()
;$colour = getButtonPressed(6)

;if ($colour == 1) {
;unset($colour)
;if ($sleep == TIMEMOTORDOWN) {
;$startStop = getButtonPressed(0)

;if ($startStop == 1) {
;unset($startStop)
;$sleep+=1
;motorDown()

;if ($colour == 1) {
;$temp=0
;storeData($temp,'outputs',HBRIDGE1)
;$temp = 10
;storeData($temp, 'outputs', HBRIDGE0)
;unset($temp)
;$state = 6
;storeData($state, 'state', 0)
;$sleep=0
;unset($state)
;motorUp()

;if ($sleep == TIMEMOTORDOWN) {

```

```

634 conditional17:    LOAD R3 0
635                  STOR R3 [GB +outputs + HBRIDGE1]
636                  LOAD R3 7
637                  STOR R3 [GB +outputs + FEEDERENGINE]
638
639                  LOAD R3 4
640                  STOR R3 [GB +state + 0]
641                  LOAD R2 0
642
643                  BRA runningWait
644
645
646 conditional18:    LOAD R4 0
647                  STOR R4 [GB +outputs + FEEDERENGINE]
648
649                  PUSH R5 ;reset timer
650                  PUSH R4
651                  LOAD R5 -16
652                  LOAD R4 0
653                  SUB R4 [R5+13]
654                  STOR R4 [R5+13]
655                  LOAD R4 BELT * 10
656                  STOR R4 [R5+13]
657                  PULL R4
658                  PULL R5
659                  LOAD R4 12
660                  STOR R4 [GB +state + 0]
661
662                  BRA motorDownTimer
663
664 motorDownTimer:   BRS timerManage
665                  LOAD R3 16
666                  STOR R3 [GB +state + 0]
667
668                  BRA motorDownStop
669
670 motorUpTimer:     BRS timerManage
671                  LOAD R3 14
672                  STOR R3 [GB +state + 0]
673
674                  BRA motorUpStop
675
676 runningTimerReset: BRS timerManage
677                  LOAD R3 4
678                  STOR R3 [GB +state + 0]
679
680                  BRA runningWait
681
682 runningTimer:     BRS timerManage
683                  LOAD R3 13
684                  STOR R3 [GB +state + 0]
685
686                  BRA runningStop
687
688
689 conditional19:    LOAD R3 0
690                  BRA return19
691
692
693 conditional20:    LOAD R4 R1
694                  PUSH R5
695                  LOAD R5 2
696                  BRS _pow
697                  LOAD R4 R5
698                  PULL R5
699                  ADD R3 R4
700                  BRA return20
701
702
703 conditional21:    PUSH R5
704                  LOAD R5 1
705                  BRS _timer
706                  PULL R5
707                  PUSH R5
708                  LOAD R5 -16
709                  STOR R3 [R5+11]
710                  PULL R5
711
712                  PUSH R3
713                  LOAD R3 1
714                  BRS _pressed
715                  PULL R3
716                  SUB SP 5
717                  PULL R4
718                  ADD SP 4
719                  CMP R4 1
720                  BEQ conditional22
721 return22:         CMP R0 6
722                  BEQ conditional23
723 return23:         CMP R0 11

```

```

; $temp = 0
; storeData($temp, 'outputs', HBRIDGE1)
; $temp = 7
; storeData($temp, 'outputs', FEEDERENGINE)
; unset($temp)
; $state = 4
; storeData($state, 'state', 0)
; $sleep = 0
; unset($state)
; runningWait()

; if ($startStop == 1) {
; $temp = 0
; storeData($temp, 'outputs', FEEDERENGINE)
; unset($temp)
; setCountdown(BELT * 10)

; set timer

; $state = 12
; storeData($state, 'state', 0)
; unset($state)
; motorDownTimer()

; timerManage()
; $state = 16
; storeData($state, 'state', 0)
; unset($state)
; motorDownStop()

; timerManage()
; $state = 14
; storeData($state, 'state', 0)
; unset($state)
; motorUpStop()

; timerManage()
; $state = 4
; storeData($state, 'state', 0)
; unset($state)
; runningWait()

; timerManage()
; $state = 13
; storeData($state, 'state', 0)
; unset($state)
; runningStop()

; if ($location == 0) {
; $engines = 0
; }

; if ($voltage > $counter) {
; $voltage = $location
; $voltage = pow(2, $voltage)

; $engines += $voltage
; }

; if ($location == 7) {
; sleep(1)

; display($engines, 'Leds')

; unset($voltage)
; $abort = getButtonPressed(1)

; if ($abort == 1) {

; unset($abort)
; if ($counter == 6) {

; if ($counter == 11) {

```

725	BEQ conditional24	
726	LOAD R3 0	;engines = 0
727	LOAD R1 0	;location = 0
728	ADD R0 1	;counter+=1
729	RTS	;return
730	BRA return21	;
731		
732		;if (\$abort == 1) {
733	conditional22: BRA abort	;abort()
734	BRA return22	;
735		
736		;if (\$counter == 6) {
737	conditional23: LOAD R4 [GB + state + 0]	;temp = getData('state', 0)
738	MOD R4 10	;mod(10, \$temp)
739	PUSH R5	;display(\$temp, 'display', 1)
740	PUSH R4	
741	LOAD R5 R4	
742	BRS _Hex7Seg	
743	LOAD R4 %0000001	
744	STOR R4 [R5+9]	
745	PULL R4	
746	PULL R5	
747		;unset(\$temp)
748	BRA return23	;
749		
750		;if (\$counter == 11) {
751	conditional24: PUSH R2	;pushStack(\$sleep)
752	LOAD R4 [GB + state + 0]	;temp = getData('state', 0)
753	LOAD R2 R4	;sleep = \$temp
754	MOD R2 10	;mod(10, \$sleep)
755	SUB R4 R2	;temp -= \$sleep
756	DIV R4 10	;temp /= 10
757	PUSH R5	;display(\$temp, 'display', 2)
758	PUSH R4	
759	LOAD R5 R4	
760	BRS _Hex7Seg	
761	LOAD R4 %0000010	
762	STOR R4 [R5+9]	
763	PULL R4	
764	PULL R5	
765	PULL R2	;pullStack(\$sleep)
766		;unset(\$temp)
767	BRA return24	;
768		
769	abort:	;unset(\$engines)
770	PUSH R5 ;reset timer	;setCountdown(1000)
771	PUSH R4	
772	LOAD R5 -16	
773	LOAD R4 0	
774	SUB R4 [R5+13]	
775	STOR R4 [R5+13]	;set timer
776	LOAD R4 1000	
777	STOR R4 [R5+13]	
778	PULL R4	
779	PULL R5	
780	LOAD R3 [GB + stackPointer + 0]	;temp = getData('stackPointer', 0)
781	LOAD SP R3	;setStackPointer(\$temp)
782	LOAD R3 0	;temp = 0
783	STOR R3 [GB +outputs + HBRIDGE1]	;storeData(\$temp, 'outputs', HBRIDGE1)
784	STOR R3 [GB +outputs + HBRIDGE0]	;storeData(\$temp, 'outputs', HBRIDGE0)
785	STOR R3 [GB +outputs + LENS LAMP POSITION]	;storeData(\$temp, 'outputs', LENS LAMP POSITION)
786	STOR R3 [GB +outputs + LENS LAMP SORTER]	;storeData(\$temp, 'outputs', LENS LAMP SORTER)
787	STOR R3 [GB +outputs + LED STATE INDICATOR]	;storeData(\$temp, 'outputs', LED STATE INDICATOR)
788	STOR R3 [GB +outputs + DISPLAY]	;storeData(\$temp, 'outputs', DISPLAY)
789	STOR R3 [GB +outputs + CONVEYOR BELT]	;storeData(\$temp, 'outputs', CONVEYOR BELT)
790	STOR R3 [GB +outputs + FEEDER ENGINE]	;storeData(\$temp, 'outputs', FEEDER ENGINE)
791		;unset(\$temp)
792	BRS timerManage	;timerManage()
793	LOAD R3 17	;state = 17
794	STOR R3 [GB +state + 0]	;storeData(\$state, 'state', 0)
795	LOAD R3 7	;state = 7
796	PUSH R5	;display(\$state, 'Leds2', 0)
797	LOAD R5 -16	
798	STOR R3 [R5+10]	
799	PULL R5	
800		;unset(\$state)
801	BRA aborted	;aborted()
802		
803	aborted: PUSH R5 ;reset timer	;setCountdown(1000)
804	PUSH R4	
805	LOAD R5 -16	
806	LOAD R4 0	
807	SUB R4 [R5+13]	
808	STOR R4 [R5+13]	;set timer
809	LOAD R4 1000	
810	STOR R4 [R5+13]	
811	PULL R4	
812	PULL R5	
813	BRS timerManage	;timerManage()
814	PUSH R3	;startStop = getButtonPressed(0)
815	LOAD R3 0	

816	BRS _pressed	
817	PULL R3	
818	SUB SP 5	
819	PULL R3	
820	ADD SP 4	
821	CMP R3 1	;if (\$startStop == 1) {
822	BEQ conditional25	
823 return25:		;unset(\$startStop)
824	BRA aborted	;aborted()
825		
826		;if (\$startStop == 1) {
827 conditional25:	LOAD R4 10	;\$temp = 10
828	STOR R4 [GB +outputs + HBRIDGE0]	;storeData(\$temp, 'outputs', HBRIDGE0)
829		;unset(\$temp)
830	LOAD R4 0	;\$state = 0
831	STOR R4 [GB +state + 0]	;storeData(\$state, 'state', 0)
832		;unset(\$state)
833	BRA initial	;initial()
834		
835 timerManage:	CMP R1 0	;if (\$location == 0) {
836	BEQ conditional19	
837 return19:	MOD R0 12	;mod(12, \$counter)
838	ADD R1 outputs	;\$voltage = getData('outputs', \$location)
839	LOAD R4 [GB + R1]	
840	SUB R1 outputs	
841	CMP R4 R0	;if (\$voltage > \$counter) {
842	BGT conditional20	
843 return20:	CMP R1 7	;if (\$location == 7) {
844	BEQ conditional21	
845 return21:	ADD R1 1	;\$location+=1
846	BRA timerManage	;branch('timerManage')
847		
848	@END	