Final Report

17th of April 2015

2IO70

Version 2

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

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:	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:	1. The machine finishes sorting the disks currently in the machine 2. 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:	1. An unsorted disk is moved to the colour detector
	2. The machine decides to which of the two containers the disk needs to be moved
	3. The machine moves the disk to the designated container
	4. The machine repeats step 2 through 4 until all disks have been sorted
	5. 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:	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:	1. Connect the PP2-board to the pc.
	2. Plug the pp2-board in to the power socket.
	3. Start the debugger
	4. 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:	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.
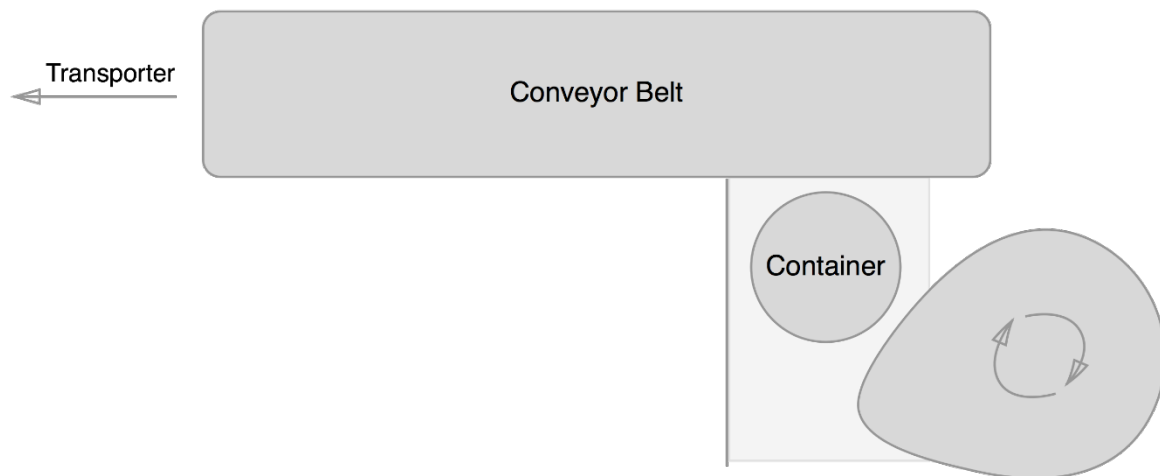
The first design for the feeder consist of 3 important parts. First you have the container. The container drops a disk, which is then pushed onto the conveyor belt using a cam. A wall to the left of the container makes sure the disk is pushed up and not to the left.

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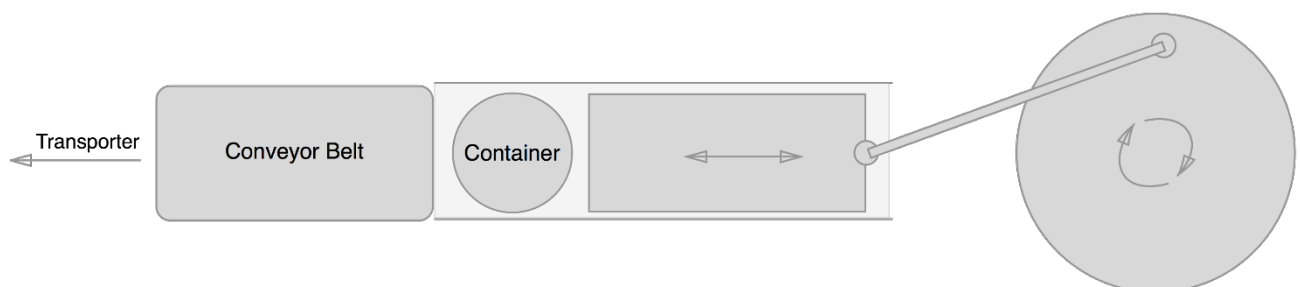
built both and tested them. They scored the same on almost all top priorities. They were

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.
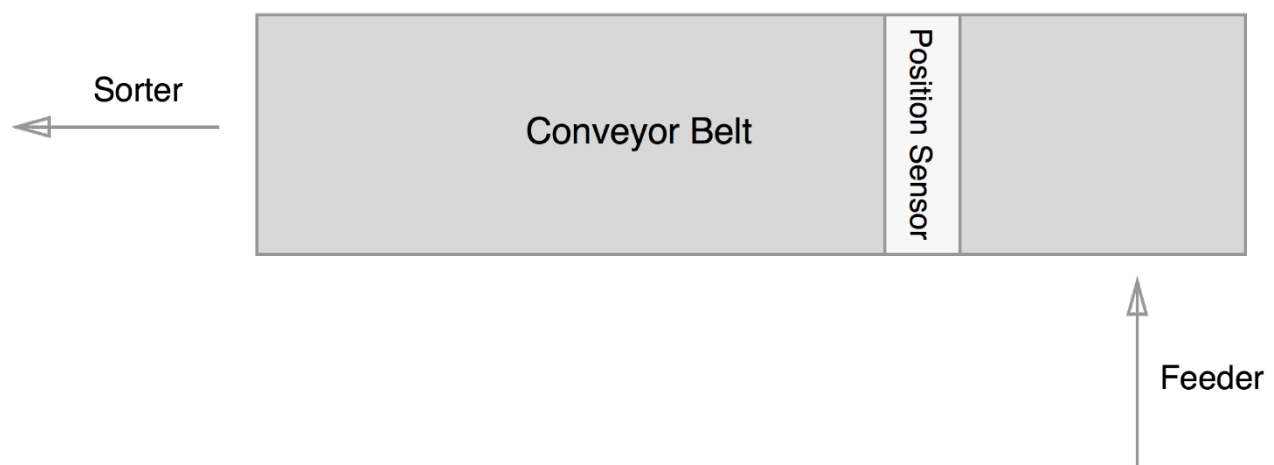
We were capable of realizing the of the long conveyor belt. But during the build of the conveyor belt we noticed that it would not be tight enough around the gears. Thus we tried to remove a small part of the belt. But this still didn't have to effect we hoped for. So we added a third gear in the middle 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.
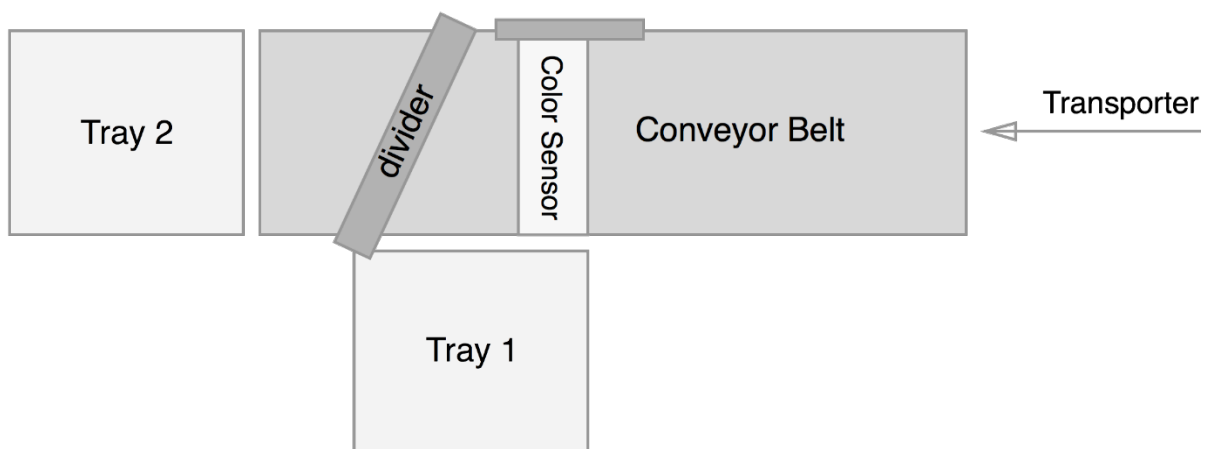
While the use of a seesaw sped up the sorting process, it also took a lot more space, so we went back to the drawing board and discarded this idea. Instead coming up with a wedge that would be slide onto the conveyor belt from the side whenever a disk of a certain colour is detected. This would then 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 us 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 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 conveyor 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
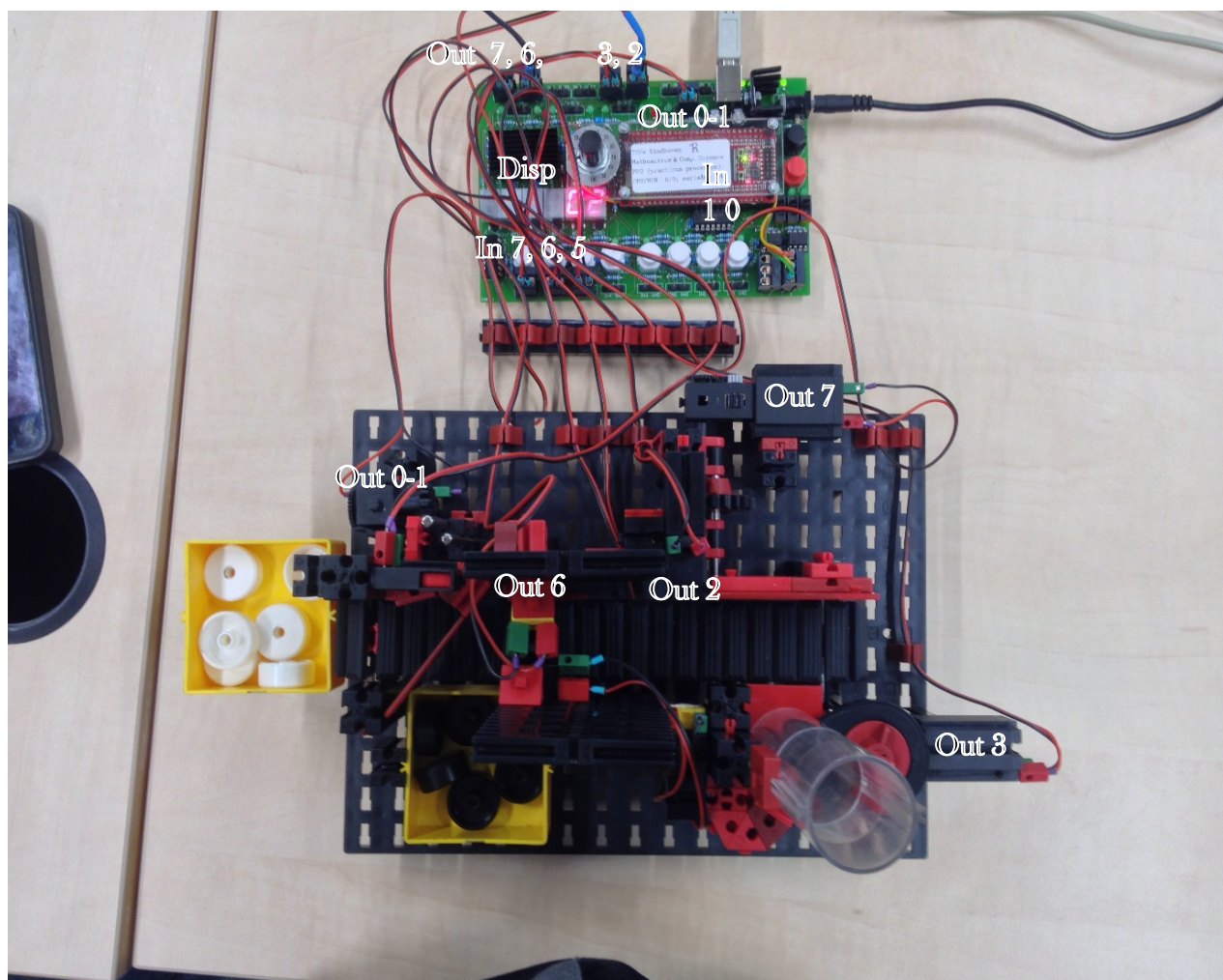
System Validation and Testing
Validate High level specifications
Our high level specifications are correct, because in the exercise it is said that a sorting machine for black and white discs should be made. And it also is said that we need at least one conveyer belt.

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 buttons 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 12s 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.
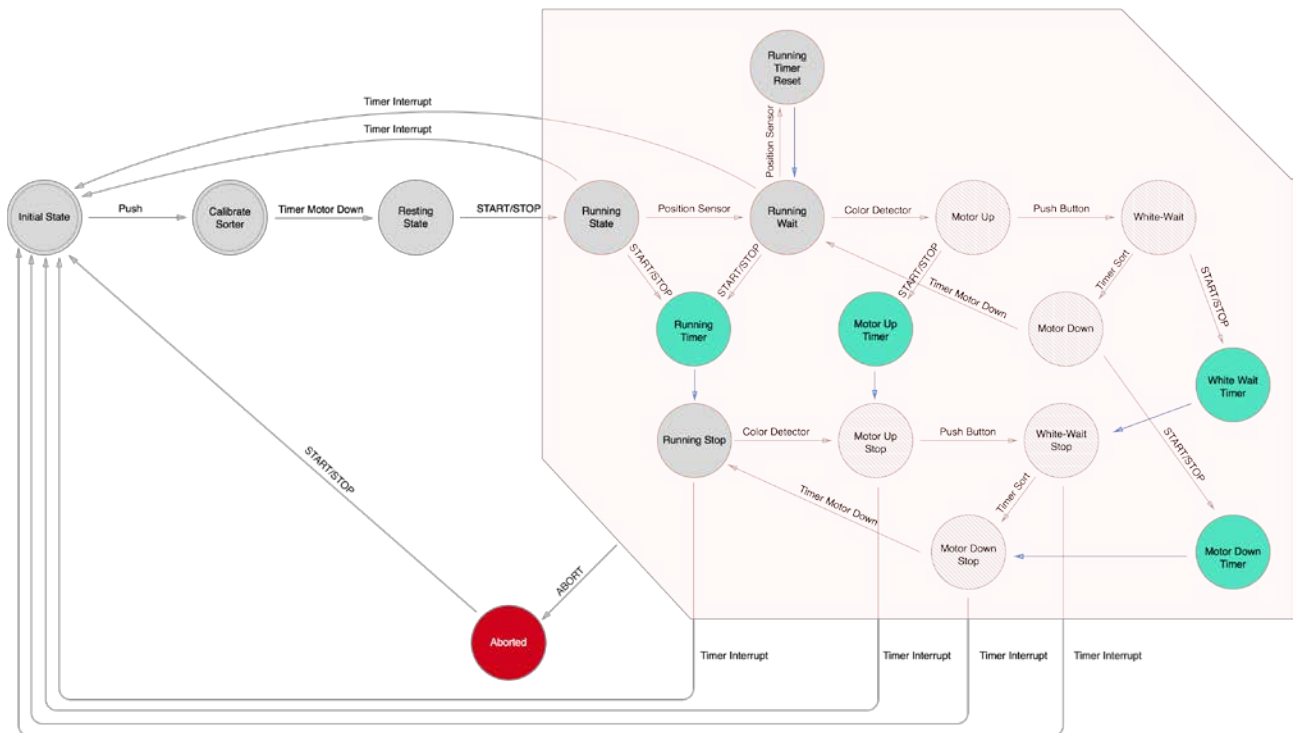
Tests done

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 form 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 where 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>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($state);
217
218         runningTimer();
219
220      }
221      unset($startStop);
222
223      //check if a disk is at the position detector
224      $position = getButtonPressed(7);
225      if ($position == 0) {
226         //reset the countdown, a disk was detected
227         setCountdown(COUNTDOWN);
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 }
```

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 //state 3	
198 function running()	
199 {	
200 timerManage();	265 running: BRS timerManage
201	
202 //check if we need to pause	
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 //stop the feeder engine	
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 //exit after 1 rotation of the belt	
	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 //update the state	
214 $state = 9;//TODO: echte state	302 LOAD R4 9 ;$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 //check if a disk is at the position detector	
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 //reset the countdown, because a disk was just detected	
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 //update the state	
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 //loop	
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- 38) 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 then 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 trough 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 conveyer 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.

Validation Work Plan

Evaluation time planned and spent

Task	Time spent ([hh]:mm)	Time planned ([hh]:mm)	Total work planned ([hh]:mm)	Overworked ([hh]:mm)	Overworked (%)	Planned difference ([hh]:mm)
Wp	26:45	22:00	20:00	04:45	21.59%	02:00
Wp.Df	00:00	02:00		-02:00	-100.00%	
Wp.Tt	00:00	19:00		-19:00	-100.00%	
Wp.L	00:00	01:00		-01:00	-100.00%	
Md	19:55	16:00	20:00	03:55	24.48%	-04:00
Md.Mi	04:55	06:00		-01:05	-18.06%	
Md.Tc	06:15	07:00		-00:45	-10.71%	
Md.L	00:55	03:00		-02:05	-69.44%	
Md.Cr	01:00	00:00		01:00	∞	
Ss	74:53	145:00	135:00	-70:06	-48.35%	10:00
Ss.In	02:43	10:00		-07:16	-72.71%	
Ss.Ot	01:25	15:00		-13:35	-90.56%	
Ss.Dio	01:25	12:00		-10:35	-88.19%	
Ss.Ias	12:58	09:00		03:58	44.21%	
Ss.Sc	04:38	06:00		-01:21	-22.57%	
Ss.UPP	08:50	66:00		-57:10	-86.62%	
Ss.L	08:00	03:00		05:00	166.67%	
Ss.Cr	00:15	24:00		-23:45	-98.96%	
Sd	52:20	97:00	100:00	-44:40	-46.05%	-03:00
Sd.I/o	00:00	06:00		-06:00	-100.00%	
Sd.Fe	00:00	42:00		-42:00	-100.00%	
Sd.Ec	01:15	17:00		-15:45	-92.65%	
Sd.Dd	01:00	15:00		-14:00	-93.33%	
Sd.L	00:00	04:00		-04:00	-100.00%	
Sd.Cr	00:45	13:00		-12:15	-94.23%	
Si	58:05	56:00	60:00	02:05	3.72%	-04:00
Si.Cs	00:00	30:00		-30:00	-100.00%	
Si.Fa	00:00	21:00		-21:00	-100.00%	
Si.L	00:15	01:00		-00:45	-75.00%	
Si.Cr	00:00	04:00		-04:00	-100.00%	
VaT	14:25	54:00	85:00	-39:35	-73.30%	-31:00
VaT.Tc	00:00	00:00		00:00	∞	
VaT.SLR	02:45	00:00		02:45	∞	
VaT.Uf	00:00	00:00		00:00	∞	
VaT.Dv	00:20	00:00		00:20	∞	
VaT.Co	00:00	12:00		-12:00	-100.00%	
VaT.Pr	00:00	10:00		-10:00	-100.00%	
VaT.L	00:00	16:00		-16:00	-100.00%	
VaT.Cr	00:00	16:00		-16:00	-100.00%	
FR	43:04	42:00	45:00	01:04	2.54%	-03:00
FR.L	04:40	42:00		-37:20	-88.89%	
FR.Cr	00:00	00:00		00:00	∞	
Pr	33:12	27:00	45:00	06:12	22.99%	-18:00
Pr.Mp	18:07	18:00		00:07	0.69%	
Pr.P	03:12	09:00		-05:47	-64.35%	
Undefined	71:13	00:00	00:00	71:13	∞	00:00
Total	**393:54**	**459:00**	**510:00**	**-65:06**	**-14.18%**	**-51:00**

Table 1 Overview of amount of time planned and spent in weeks 4 to 8. We start with week 4 because then our Work Plan started and ended in week 8, because in week 9 the calculations were finished. Time spent is derived from the logbooks. Time planned is derived from the weekly tables of the Work Plan. Total work planned is derived from the Total work section of the Work Plan. Overworked is based on Time spent and Time planned. Planned difference is Time planned – Total work planned. Tasks in bold actually are phases.

Logbooks

Next time we keep logbooks we should stick to the task description in the Work Plan instead of making up a description that often differs per person and differs in detail to the Work Plan. This way it saves time to do the calculations, whereas now we had to come up with the task description consistent with the Work Plan for the descriptions in the logbook. Another thing that could be improved is the way we keep the logbook. Instead of having a

file where we can keep logbooks we should design the file as such that the calculations can be done way easier. It takes some more time in the beginning of the project but saves a lot of time at the end.

Work Plan

The Work Plan lacked some tasks. For instance the FR phase lacked the whole process part documentation.

The weekly tables had 1 row per week that had a time duration of 30 minutes. This too is very inconvenient in calculating. We couldn't figure out, on a quick enough notice, how to deal with this. Therefore these half hours are counted as whole hours. This is luckily a small part of the planning. Another thing that's off with the tables is that it has a different amount of time planned than the scheme in the section Total work of the Work Plan. This is due to that we wanted to finish the Work Plan quickly. Another fault due to this reason is that some subtasks didn't get planned, like the first 4 subtasks of Validation and Testing.

Planned difference

The column labelled "Planned difference" shows per phase and in total how much time is falsely planned due to the reason given above. Because of this false planning we base the rest of the numbers on the actual planning, which is in the Timetables section of the Work Plan. We chose this as the actual planning because it's the most detailed one.

Time planned

We set ourselves the goal to spend 500 hours on the project, from week 4 to week 8, collectively. We got this number from the information from our tutor that we have to think about spending 700 hours collectively on the project. We started the planning from week 4, so it should be less. We also decided that we wanted to be done before the exam weeks, week 9 and 10. Per week, the time planned is more than the indicated time divided by 8 weeks. This is because we thought that we spent too little time in those first 3 weeks. Why it says 510 in the table, however, is because there were some late changes. Some tasks were expected to take a few hours longer.

Overworking

As you can see in Table 1, we underworked quite a bit. This may be partially explained by not being finished at the end of week 8. We will be spending some more time in weeks 9 and 10. Another reason might be that we work more efficient, but this is hard to show with these intermediate data.

Phases

The time spent on phases is not just simply the summation of the subtasks. We did this so we could add spent time to the phase if the subtask wasn't specified.

Work Plan (Simultaneously with Machine Design)
It seems that we underestimated the time needed to finish the Work Plan. This may as well explain why we rushed the weekly tables a bit. Due to bad logging it isn't clear on what subtask the time was spent.

Machine Design
We spent almost 25% more time on this. However the overworked time is just about 4 hours, which is small compared to the total length of the project.

Software Specification
We clearly overestimated the time needed for this phase. This is partially due to our tutor predicting that this phase will probably take the most time.

Software Design
Apparently we didn't follow the Work Plan well enough on this one. Except from "Compiling and defining layout of the document" (Sd.L) the two other tasks weren't executed in the first 8 weeks. This may be explained by how we work, described in the Workday section. There it isn't mentioned that we look at the Work Plan for tasks to be done.

Software Implementation and Integration
The overall planning and execution of this phase is quite well.

Validation and Testing
We don't think that we overestimated the time needed for Validation and Testing but that we didn't log this when we did it. This may have to do with our VaT being done and documented throughout the span of the project which may have caused that this got logged into another phase. This implies that the subtasks weren't logged either.

Final Report
We kind of underestimated the work needed to finish the project. We thought that we only had to put all our documents together and write a conclusion for the Validation and Testing. Though, there's the process part that needs to be documented in this and also this document needs an introduction or preface and a conclusion.

Presentation
We underestimated the presentations. You already see that for the first 8 weeks we 'overworked'. This is only the time spent on the mid-term presentation. Despite the fact that we had to redo it, this isn't a valid reason that it took us more time. If we prepared better for our first attempt, which we lacked in shown by talking hesitantly and softly, we wouldn't have to do it a second time. Then we didn't even discuss the final presentation we started working on in week 9.

Undefined
Due to inconsistent logging as discussed in the Logbook section above there's a lot of time in the logbooks that we couldn't add to a certain phase for sure.

Peaks

To find peaks to discuss here we didn't only look at the overworked percentage but also the overworked time. If the planned time is small, bigger differences between Time spent and Time planned are forgiven more easily. We guess it's harder to plan the time needed for a small task and small tasks have less impact on the overall project.

Ss.In, Ss.Ot and Ss.Dio
Apparently we thought that these tasks would be more complicated and take more time, but in practice this was not the case.

Ss.UPP
We didn't have that much testing, because we had a time shortage, and learning how to validate it cost even more time. Thus, it was easier for us to test the machine in practice2.

Ss.Cr and Sd.Cr
We neglected this task. When we finished these phases we were more excited about having it finished and delivering it than investing more time to go through it and enhance it.

Sd.I/o, Sd.Fe and Sd.L
See section Machine Design above.

Sd.Ec and Sd.Dd
We think this underworking is due to bad logging, putting less effort into it and overestimating the time needed for the task. If you look at the outcome of these tasks in the Software Design document we wouldn't say we just spent about 1 hour on these tasks, but we probably didn't spent much more either. They could have been more extensive but there may be some lack of motivation caused by the team members working on the technical part of this phase were also expected to write the documentation and this may have been too much.

Si.Cs, Si.Fa and Si.Cr
In the Software Implementation and Integration section above it is mentioned that this phase went quite well, but if you look into the subtasks you might state quite the opposite. This is due to bad logging. Probably after being busy with the project for a few weeks the logbooks got less attention and therefore were less detailed. This is why the subtasks appear to have no time spent on them.

VaT.Co, VaT.Pr, VaT.L and VaT.Cr
See Validation and Testing section above.

FR.L
The Final Report was not finished at the end of week 8. Therefore the "Compiling and defining layout of the document" couldn't be done yet.

Pr.P
If you look at the time missing for this subtask and the time overworked in the presentation phase they cancel out. Therefore we think that this missing time is due to bad logging.

Evaluation Team Roles

	Dat	Rolf	Stefan	Tudor	Wigger	Maarten	
Week 4	Q	-	-	S	-	P, M	
Reality	x	-	S(1)	S(2)	-	P, M	
Week 5	Q	-	S	P	-	M	
Reality	x	-	S(1), x(2)	P(1), x(2)	-	x	
Week 6	S, Q	-	P	-	-	M	
Reality	x	-	x	-	-	x	
Week 7	-	-	S	-	P	Q, M	P = President
Reality	-	-	x	-	x	x	S = Secretary
Week 8	-	P	-	-	-	S, Q, M	Q = Quality assurance manager
Reality	-	x	-	-	-	x	M = Materials manager

Table 2 Overview of roles assigned by the Work Plan and the reality checked by the Minutes. x's mean that the minutes didn't report if the role was performed by the right person. (#) expresses at what meeting this was the case. A hyphen means that this person had no role that week.

Minutes

The minutes need to be improved to let this validation succeed. Only 2 minutes provided who was president and who was secretary, therefore the xs in Table 2.

Results

There's little to say about if the roles were executed at the correct time and everyone has been president and secretary once and the materials manager didn't change and the quality assurance manager changed according to the requirements. There's very little information about the reality. Where there's information about it, 3 out of 4 roles were executed at the right time by the right person. Once Stefan took Tudor's role of secretary, because he was too late for the meeting.

Being Late

Excesses

There were cases where group members were very late or absent without notifying anyone. One of those instances Maarten was during a whole day and therefore missed the meeting with the tutor too. Maarten explains that this was because of oversleeping due to depression and not daring to come afterwards.

Another time Tudor was very late for our personal meeting and working on the project. He didn't want to talk about why but mentioned something about dying and said it surprised him that he came at the end.

Also Dat has managed to oversleep and he did this big time. About 11.00 or 12.00 hours we received a message that he just woke up. He explained that he was watching 2 movies till 3 AM. Because of this oversleeping he missed a meeting with our tutor.

Meetings

Because the minutes of each meeting states if people were late and how late, we're able to check how late every group member was and you can see the result in Table 3.

	STEFAN	WIGGER	MAARTEN	TUDOR	DAT	ROLF
LATE (M)	0	21	15	45	5	0

Table 3 How many minutes late was each group member for the tutor meetings? NB: These are only the minutes where there was no good excuse for being late.

Conclusion

It is clear from these numbers that we so far overestimated, or under-reported, the amount of work needed for each part of the project. It must however be said that the table above only includes work towards the major deliverables, this means that work on individual assignments (such as the abstract and maintaining the logbooks) and group talks (to resolve issues) have not been included, leading to an underestimation of the amount of time spent on the project. Either way, it seems that some members have contributed significantly less than others. This has now been addressed in the group however, and going forward we will try to create a more even workload for all.

As a result of the problems addressed above, 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.

Group reflection

This was our first Design Based Learning (DBL) course and therefore new to all of us. Never have we experienced group work so intensively and we felt that it was getting more serious. We underestimated the documentation and process part a lot.

It wasn't always easy to start working or to get everyone to face the same direction but after a few weeks of struggling we got the hang of it and felt like we could do this. We found out that it was best for our group to write the tasks on the whiteboard and assign people to it. However, we think that more communication and involvement is needed in the further projects to make working as a team more pleasant.

We also experienced this project as very time consuming and worked long hours on it at TU/e.

The actual machine of the project draws a lot of our attention and interest and it was a good visual of our hard work, all of us were pleased with the overall performance that we obtained in the end.

But as mentioned earlier, a lot of work was put in the project, leaving some of us with a mixed feeling, that the amount of work sometimes overshadowed the fun we had and the knowledge and experience we acquired during this time.

The group had to present the content of the project in two separate presentation, while the final presentation was carried out with no problem, the mid-term presentation was obliged to be redone and it was opted to be under a video-presentation format, the approach was

completely different from the original presentation, offered more creative freedom, but maintain the style and organisation of the old one and added more content unavailable in the mid-term period.

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
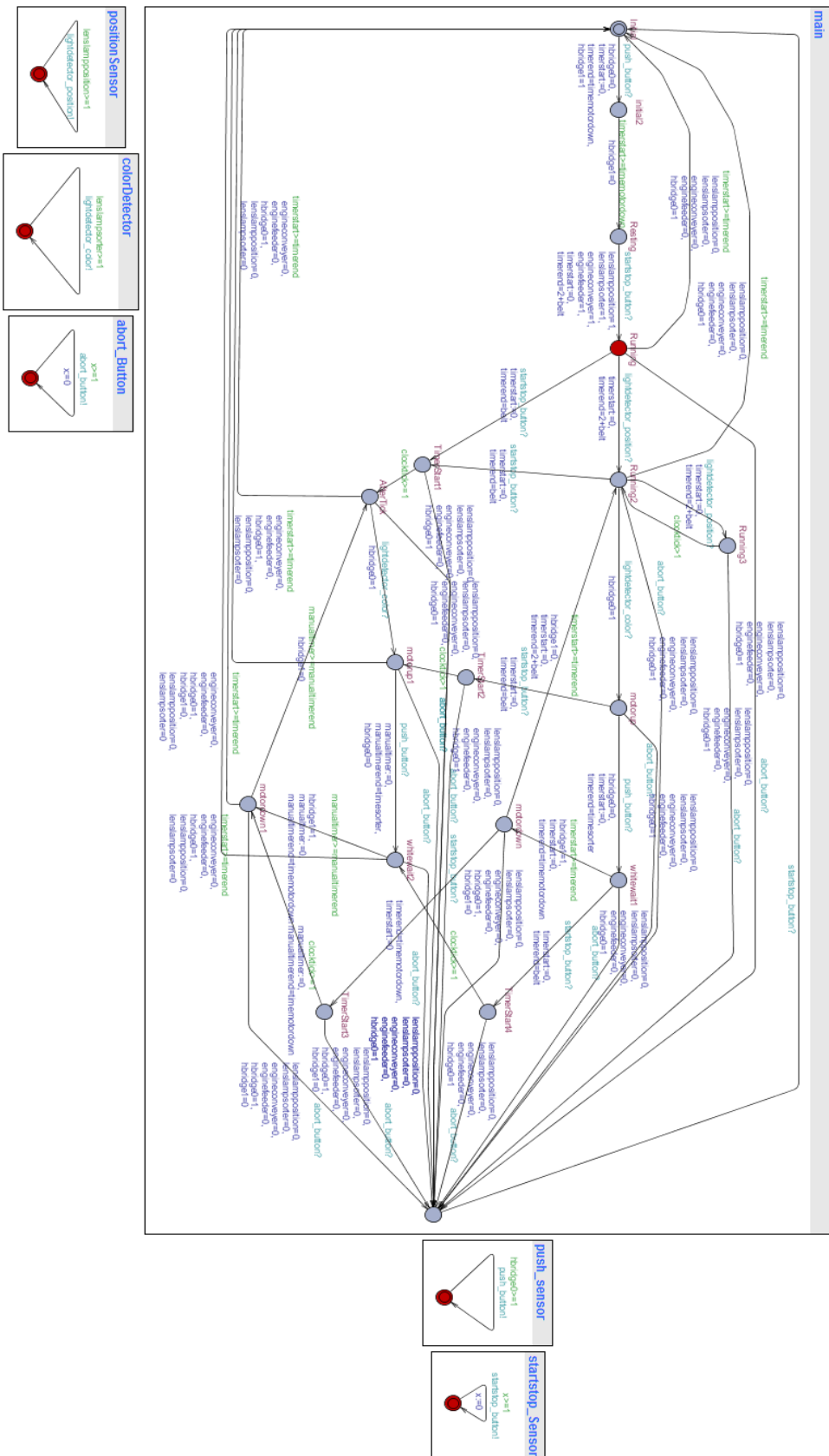
Appendix 2: Table of the display of states

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

```java
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 LENSLAMPPOSITION = 5,
36             LENSLAMPSORTER = 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 programm,this
55         // is used in the interrupt
56         SoftwareDesign.storeData(startofthecode,
57                         "offset", 0);
58
59         //store the vlue of the stackpointer,so
60         // we can clear the stack
61         // easily
62         SoftwareDesign.storeData(SP,
63                         "stackpointer",
64                         0);
65
66         $counter = 0;
67
68
69         //reset outputs
70         SoftwareDesign.storeData(0, "outputs",
71                         SoftwareDesign
72                         .HBRIDGE1);
73         SoftwareDesign.storeData(0, "outputs",
74                         SoftwareDesign
75                         .LENSLAMPPOSITION);
76         SoftwareDesign.storeData(0, "outputs",
77                         SoftwareDesign
78                         .LENSLAMPSORTER);
79         SoftwareDesign.storeData(0, "outputs",
80                         SoftwareDesign
81                         .LEDSTATEINDICATOR);
82         SoftwareDesign.storeData(0, "outputs",
83                         SoftwareDesign
84                         .DISPLAY);
85         SoftwareDesign.storeData(0, "outputs",
86                         SoftwareDesign
87                         .CONVEYORBELT);
88         SoftwareDesign.storeData(0, "outputs",
89                         SoftwareDesign
90                         .FEEDERENGINE);
91
92         //start moving the sorter up
93         SoftwareDesign.storeData(9, "outputs",
94                         SoftwareDesign
95                         .HBRIDGE0);
96
97         //go to the first state and set the
98         // value for the display
99         SoftwareDesign.$state = 0;
100        SoftwareDesign.initial();
101    }
102
103    //state0
104    void initial() {
105        setStackPointer(
106                getData("stackpointer", 0));
107        timerManage();
108        //check if the sorter push button is
109        // pressed
110        $push = getButtonPressed(5);
111        if ($push == 1) {
112            //move the sorter down
113            storeData(0, "outputs", HBRIDGE0);
114            storeData(9, "outputs", HBRIDGE1);
115            //update the state
116            $state = 1;
117            //reset sleep for the next function
118            $sleep = 0;
119            calibrateSorter();
120
121        }
122        //loop
123        initial();
124    }
125
126    //state 1
127    void calibrateSorter() {
128        timerManage();
129        //the sorter is now moving down,
130        //and we're waitng for it to reach the
131        // bottom
132        if ($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        timerManage();
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            // immediatly
155            sleep(2000);
156            //power up the lights
157            storeData(12, "outputs",
158                    LENSLAMPPOSITION);
159            storeData(12, "outputs",
160                    LENSLAMPSORTER);
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        timerManage();
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        timerManage();
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      timerManage();
245      //update the state
246      $state = 5;
247      runningWait();
248  }
249
250  //state 6
251  void motorUp() {
252      timerManage();
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          $state = 7;
271          whiteWait();
272      }
273      //loop
274      motorUp();
275  }
276
277  //state 7
278  void whiteWait() {
279      timerManage();
280      //we are waiting for the white disk to
281      // be sorted
282      if ($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      $sleep++;
305      whiteWait();
306  }
307
308  //state 8
309  void motorDown() {
310      timerManage();
311      //the sorter is moving down
312      if ($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
336  //state 9
337  void runningTimer() {
338      timerManage();
339      //update state
340      $state = 13;
341      runningStop();
342  }
343
344  //state 10
345  void motorUpTimer() {
346      timerManage();
347      //update state
348      $state = 14;
349      motorUpStop();
350  }
351
352  //state 11
353  void whiteWaitTimer() {
354      timerManage();
355      //update state
356      $state = 15;
357      whiteWaitStop();
358  }
359
360  //state 12
361  void motorDownTimer() {
362      timerManage();
363      //update state
364      $state = 16;
365      motorDownStop();
366  }
367
368  //state 13
369  void runningStop() {
370      timerManage();
371      //check if a white disk is at the
372      // colour detector
373      $colour = getButtonPressed(6);
374      if ($colour == 1) {
375          //move the sorter engine up
376          storeData(9, "outputs", HBRIDGE0);
377          //update the state
378          $state = 10;
379          motorUpStop();
380      }
381      //loop
382      runningStop();
383  }
384
385  //state 14
386  void motorUpStop() {
387      timerManage();
388      //check if the sorter push button is
389      // pressed
390      $push = getButtonPressed(5);
391      if ($push == 1) {
392          //stop the engien for the sorter
393          storeData(0, "outputs", HBRIDGE0);
394          //update the state
395          $state = 11;
396          whiteWaitStop();
397      }
398      motorUpStop();
399  }
400
401  //state 15
402  void whiteWaitStop() {
403      timerManage();
404      //check if the white disk has been sorted
405      if ($sleep == SORT * 1000) {
406          //start moving the sorter down
407          storeData(9, "outputs", HBRIDGE1);
408          //update the state
409          $state = 12;
410          //reset the sleep for the next
411          // function
412          $sleep = 0;
413          motorDown();
414      }
415      //loop
416      $sleep++;
417      whiteWaitStop();
418  }
419
420  //state 16
421  void motorDownStop() {
422      timerManage();
423      //check if the sorter has moved down
424      if ($sleep == TIMEMOTORDOWN) {
425          //stop the engine of the sorter
426          storeData(0, "outputs", HBRIDGE1);
427          //update the state
428          $state = 9;
429          //reset sleep for the next function
430          $sleep = 0;
```

```
431         runningWait();
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", LENSLAMPPOSITION);
447     storeData(0, "outputs", LENSLAMPSORTER);
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     // immediatly
455     timerManage();
456     //set the display to the state of initial
457     $state = 0;
458
459     initial();
460
461 }
462
463 void abort() {
464     //stop all outputs
465     storeData(0, "outputs", HBRIDGE0);
466     storeData(0, "outputs", HBRIDGE1);
467     storeData(0, "outputs", LENSLAMPPOSITION);
468     storeData(0, "outputs", LENSLAMPSORTER);
469     storeData(0, "outputs",
470             LEDSTATEINDICATOR);
471     storeData(0, "outputs", DISPLAY);
472     storeData(0, "outputs", CONVEYORBELT);
473     storeData(0, "outputs", FEEDERENGINE);
474     //make sure the outputs stop immediatly
475     timerManage();
476     //update the state to be correct in
477     // aborted
478     $state = 17;
479     aborted();
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, $voltage);
513     }
514     //check if we are in a new itteration
515     if ($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     // itteration
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 "condtionali" 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.

$variable	The variable where the pulled variable is put into
return	void

setCountdown($countdown)

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

$countdown	How long the countdown should wait, in timer ticks
return	void

getData($location, $offset)

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

$location	The location where the variable is stored
$offset	The offset of the location
return	The value of the data segment

storeData($variable, $location, $offset)

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

$variable	The variable to store
$location	The name of the location where the variable is stored
$offset	The offset of the location
return	void

sleep($howLong)

Pause the program.

$howLong	How long to sleep in clockticks
return	void

initVar($variable,$places)

Initialize a variable that is used in that data segment.

$variable	The name of the variable
$places	How long the array is
return	void

branch($branchTO)

Branch to a function.

$branchTO	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

```php
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 Technik in order to sort black
and white discs.
6   * @team Group 16
7   * @author Stefan van den Berg
8   * @author Rolf Verschuuren
9   * @author Wigger Boelens
10  * @since 13/3/2015
11  */
12 include 'functions.php';
13 //**COMPILER**
14 moveFunction('timerInterrupt', 1);
15 moveFunction('timerManage', 50);
16
17 //**DATA**
18 initVar('offset', 1);
19 initVar('stackPointer', 1);
20 initVar('outputs', 12);
21 initVar('state', 1);
22
23 //**CODE**
24 define('TIMEMOTORDOWN', 150); //how long the sorter takes to move down
25 define('BELT', 2000);
26 define('BELTROUND', 2000);//Time for the belt to make a rotation
27 define('SORT', 200);//Clockticks to make a rotation
28 define('COUNTDOWN', 30000);
29 //outputs
30 define('LENSLAMPPOSITION', 2);
31 define('LENSLAMPSORTER', 6);
32 define('HBRIDGE0', 0);
33 define('HBRIDGE1', 1);
34 define('CONVEYORBELT', 7);
35 define('FEEDERENGINE', 3);
36 define('DISPLAY', 8);
37 define('LEDSTATEINDICATOR', 9);
38
39 //not a state
40 function main()
41 {
42     global $counter, $location;
43
44     //store the offset of the program, this is used in the interrupt
45     storeData(R5, 'offset', 0);
46     //install the countdown
47     installCountdown('timerInterrupt');
48
49     //save the location of the stackPointer, so we can clear the stack
50     storeData(SP, 'stackPointer', 0);
51
52     //the variables that are the same throughout the program:
53     $counter = 0;
54     $location = 0;
55     $sleep = 0;
56
57
58     //stop everything
59     $temp = 0;
60     storeData($temp, 'outputs', HBRIDGE1);
61     storeData($temp, 'outputs', LENSLAMPPOSITION);
62     storeData($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     //sh0w 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        $sleep = 0;
110        calibrateSorter();
111
112    }
113    unset($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
158         //power up the lamps
159         $temp = 12;
160         storeData($temp, 'outputs', LENSLAMPPOSITION);
161         unset($temp);
162         timerManage();
163         sleep(1000);
164         $temp = 12;
165         storeData($temp, 'outputs', LENSLAMPSORTER);
166         unset($temp);
167         timerManage();
168         sleep(2000);
169
170
171         //start up the belt and feeder
172         $temp = 9;
173         storeData($temp, 'outputs', CONVEYORBELT);
174         $temp = 9;
175         storeData($temp, 'outputs', FEEDERENGINE);
176         unset($temp);
177
178         //set and start the countdown for the moment there are no more disks
179         //this countdown will reset every time a disk is found
180         //when it triggers, timerInterrupt will be ran.
181         setCountdown(COUNTDOWN);
182         startCountdown();
183
184         //update the state
185         $state = 3;
186         storeData($state, 'state', 0);
187         unset($state);
188
189         running();
190     }
191     unset($startStop);
192
193     //loop
194     resting();
195 }
196
197 //state 3
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         $state = 9;//TODO: echte state
215         storeData($state, 'state', 0);
216         unset($state);
217
218         runningTimer();
219
220     }
221     unset($startStop);
222
223     //check if a disk is at the position detector
224     $position = getButtonPressed(7);
225     if ($position == 0) {
226         //reset the countdown, because a disk was just detected
227         setCountdown(COUNTDOWN);
228
229         //update the state
230         $state = 4;
231         storeData($state, 'state', 0);
232         unset($state);
```

```php
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($state);
261
262        runningTimer();
263
264    }
265    unset($startStop);
266
267    //check if a disk is at the position detector
268    $position = getButtonPressed(7);
269    if ($position == 0) {
270        //reset the countdown, because a disk was just detected
271        setCountdown(COUNTDOWN);
272
273        //update state
274        $state = 5;
275        storeData($state, 'state', 0);
276        unset($state);
277
278        runningTimerReset();
279
280    }
281    unset($position);
282
283    //check if a white disk is at the colour detector
284    $colour = getButtonPressed(6);
285    if ($colour == 1) {
286        //move the sorter up so the disk goes to the correct box
287        $temp = 10;
288        storeData($temp, 'outputs', HBRIDGE0);
289
290        //stop the feeder engine
291        $temp = 0;
292        storeData($temp, 'outputs', FEEDERENGINE);
293        unset($temp);
294
295        //update state
296        $state = 6;
297        storeData($state, 'state', 0);
298        unset($state);
299
300        motorUp();
301    }
302    unset($colour);
303
304    //loop
305    runningWait();
306 }
307
308 //state 5
309 function runningTimerReset()
310 {
311    timerManage();
312
313    //update state
314    $state = 4;
315    storeData($state, 'state', 0);
316    unset($state);
317
318    runningWait();
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($state);
342
343        motorUpTimer();
344
345    }
346    unset($startStop);
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
399    //check if we need to pause
400    $startStop = getButtonPressed(0);
401    if ($startStop == 1) {
402        //stop the feeder engine
403        $temp = 0;
404        storeData($temp, 'outputs', FEEDERENGINE);
405        unset($temp);
406
407        //exit after 1 rotation of the belt
408        setCountdown(BELT * 10);
409
410        //update the state
411        $state = 11;
412        storeData($state, 'state', 0);
413        unset($state);
414
415        whiteWaitTimer();
416    }
417    unset($startStop);
418
419    //loop
420    $sleep++;
421    whiteWait();
422 }
423
424 //state 8
425 function motorDown()
426 {
427    global $sleep;
428    timerManage();
429
430
431    //check if a white disk is at the colour detector
432    $colour = getButtonPressed(6);
433    if ($colour == 1) {
434        //move the sorter up so the disk goes to the correct box
435        $temp=0;
436        storeData($temp,'outputs',HBRIDGE1);
437        $temp = 10;
438        storeData($temp, 'outputs', HBRIDGE0);
439        unset($temp);
440
441        //update state
442        $state = 6;
443        storeData($state, 'state', 0);
444        $sleep=0;
445        unset($state);
446
447        motorUp();
448    }
449    unset($colour);
450
451
452    //the sorter is moving down, we are waiting for that to complete
453    if ($sleep == TIMEMOTORDOWN) {
454        //stop the sorter, its where it should be
455        $temp = 0;
456        storeData($temp, 'outputs', HBRIDGE1);
457        $temp = 7;
458        storeData($temp, 'outputs', FEEDERENGINE);
459        unset($temp);
460
461        //update state
462        $state = 4;
463        storeData($state, 'state', 0);
464        //reset sleep for the next function
465        $sleep = 0;
466        unset($state);
467
468        runningWait();
469    }
470
471    //check if we need to pause
472    $startStop = getButtonPressed(0);
473    if ($startStop == 1) {
474        //stop the feeder engine
475        $temp = 0;
476        storeData($temp, 'outputs', FEEDERENGINE);
477        unset($temp);
478
```

```
479        //exit after 1 rotation of the belt
480        setCountdown(BELT * 10);
481
482        //update the state
483        $state = 12;
484        storeData($state, 'state', 0);
485        unset($state);
486
487        motorDownTimer();
488    }
489    unset($startStop);
490
491    //loop
492    $sleep++;
493    motorDown();
494
495 }
496
497 //state 9
498 function runningTimer()
499 {
500    timerManage();
501
502    //update state
503    $state = 13;
504    storeData($state, 'state', 0);
505    unset($state);
506
507    runningStop();
508 }
509
510 //state 10
511 function motorUpTimer()
512 {
513    timerManage();
514
515    //update state
516    $state = 14;
517    storeData($state, 'state', 0);
518    unset($state);
519
520    motorUpStop();
521 }
522
523 //state 11
524 function whiteWaitTimer()
525 {
526    timerManage();
527
528    //update state
529    $state = 15;
530    storeData($state, 'state', 0);
531    unset($state);
532
533    whiteWaitStop();
534 }
535
536 //state 12
537 function motorDownTimer()
538 {
539    timerManage();
540
541    //update state
542    $state = 16;
543    storeData($state, 'state', 0);
544    unset($state);
545
546    motorDownStop();
547 }
548
549 //state 13
550 function runningStop()
551 {
552    timerManage();
553
554    //check if a white disk is at the colour detector
555    $colour = getButtonPressed(6);
556    if ($colour == 1) {
557        //stop the sorter engine, because its at its highest position
558        $temp = 10;
559        storeData($temp, 'outputs', HBRIDGE0);
560
561        //stop the feeder engine
562        $temp = 0;
563        storeData($temp, 'outputs', FEEDERENGINE);
564        unset($temp);
565
566        //update state
567        $state = 10;
568        storeData($state, 'state', 0);
569        unset($state);
570
571        motorUpStop();
572    }
573    unset($colour);
574
575    //loop
576    runningStop();
577 }
578
579 //state 14
580 function motorUpStop()
581 {
582    timerManage();
583
584    //check if the sorter push button is pressed
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 == TIMEMOTORDOWN) {
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', LENSLAMPPOSITION);
676    storeData($temp, 'outputs', LENSLAMPSORTER);
677    storeData($temp, 'outputs', LEDSTATEINDICATOR);
678    storeData($temp, 'outputs', DISPLAY);
679    storeData($temp, 'outputs', CONVEYORBELT);
680    storeData($temp, 'outputs', FEEDERENGINE);
681
682
683    //reset, because we will no longer be in timerInterrupt
684    display($temp, 'display');
685    unset($temp);
686
687    //go back to initial
688    $temp = getData('offset', 0);
689    $temp2 = getFuncLocation('initial');
690    $temp += $temp2;
691
692
693    addStackPointer(2);
694    pushStack($temp);
695    addStackPointer(-1);
696 }
697
698 //not a state
699 function abort()
700 {
701    //free some memory
702    unset($engines);
703
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', LENSLAMPPOSITION);
714    storeData($temp, 'outputs', LENSLAMPSORTER);
715    storeData($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
```

69

```
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($engines, 'leds');
790
791
792        unset($voltage);
793        //check if abort is pressed
794        $abort = getButtonPressed(1);
795        if ($abort == 1) {
796            abort();//STOP THE MACHINE!
797        }
798        unset($abort);
799
800        //check if we are in a new iteration
801        if ($counter == 6) {
802            //set the first part of the display
803            $temp = getData('state', 0);
804            mod(10, $temp);
805            display($temp, 'display', 1);
806            unset($temp);
807        }
808        //check if we are at the end of the iteration
809        if ($counter == 11) {
810            //set the second part of the display;
811            pushStack($sleep);
812
813            $temp = getData('state', 0);
814            //get the last digit of the state
815            //we have no variables left, so we use $sleep
816
817            $sleep = $temp;
818            mod(10, $sleep);
819            $temp -= $sleep;
820            $temp /= 10;
821            //display the last digit
822            display($temp, 'display', 2);
823
824            pullStack($sleep);
825            unset($temp);
826        }
827
828
829        //set the variables for the next run
830        $engines = 0;
831        $location = 0;
832        $counter++;
833
834        //and return to where we came from
835        return;
836    }
837
838    //loop
839    $location++;
840    branch('timerManage');
841 }
```

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                   LENSLAMPPOSITION EQU 2
15                   LENSLAMPSORTER 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                                            ;sleep
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]                    ; Compare the timer to 0
34                   BMI  _wait
35                   PULL R4
36                   RTS
37
38 _pressed:         PUSH R4                            ;make sure all vars are the same at the end
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                      ;the result
50                   ADD SP 1                     ;decrease the SP so we get the correct pulls
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                                            ;display
78 _Hex7Seg:         BRS _Hex7Seg_bgn                   ; push address(tbl) onto stack and proceed at bgn
79 _Hex7Seg_tbl:     CONS %01111110                     ; 7-segment pattern for '0'
80                   CONS %00110000                     ; 7-segment pattern for '1'
81                   CONS %01101101                     ; 7-segment pattern for '2'
82                   CONS %01111001                     ; 7-segment pattern for '3'
83                   CONS %00110011                     ; 7-segment pattern for '4'
84                   CONS %01011011                     ; 7-segment pattern for '5'
85                   CONS %01011111                     ; 7-segment pattern for '6'
86                   CONS %01110000                     ; 7-segment pattern for '7'
87                   CONS %01111111                     ; 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                                         ; Set the timer to 0
110                 LOAD R0 0
111                 SUB R0 [R5+13]
112                 STOR R0 [R5+13]
113                 STOR SP [GB +stackPointer + 0]             ;storeData(SP, 'stackPointer', 0)
114                 LOAD R0 0                            ;$counter = 0
115                 LOAD R1 0                            ;$location = 0
116                 LOAD R2 0                            ;$sleep = 0
117                 LOAD R3 0                            ;$temp = 0
118                 STOR R3 [GB +outputs + HBRIDGE1]            ;storeData($temp, 'outputs', HBRIDGE1)
119                 STOR R3 [GB +outputs + LENSLAMPPOSITION]      ;storeData($temp, 'outputs', LENSLAMPPOSITION)
120                 STOR R3 [GB +outputs + LENSLAMPSORTER]       ;storeData($temp, 'outputs', LENSLAMPSORTER)
121                 STOR R3 [GB +outputs + LEDSTATEINDICATOR]    ;storeData($temp, 'outputs', LEDSTATEINDICATOR)
122                 STOR R3 [GB +outputs + DISPLAY]            ;storeData($temp, 'outputs', DISPLAY)
123                 STOR R3 [GB +outputs + CONVEYORBELT]         ;storeData($temp, 'outputs', CONVEYORBELT)
124                 STOR R3 [GB +outputs + FEEDERENGINE]         ;storeData($temp, 'outputs', FEEDERENGINE)
125                 LOAD R4 0                            ;$state = 0
126                 STOR R4 [GB +state + 0]                   ;storeData($state, 'state', 0)
127                 LOAD R3 10                           ;$temp = 10
128                 STOR R3 [GB +outputs + HBRIDGE0]            ;storeData($temp, 'outputs', HBRIDGE0)
129                                         ;unset($temp, $state)
130                 BRA initial                          ;initial()
131
132 timerInterrupt:      BRS timerManage                          ;timerManage()
133                 LOAD R3 5                            ;$temp = 5
134                 PUSH R5                             ;display($temp, 'display')
135                 PUSH R4
136                 LOAD R5 R3
137                 BRS _Hex7Seg
138                 LOAD R4 %0000001
139                 STOR R4 [R5+9]
140                 PULL R4
141                 PULL R5
142                 LOAD R3 10                           ;$temp = 10
143                 STOR R3 [GB +outputs + HBRIDGE0]            ;storeData($temp, 'outputs', HBRIDGE0)
144                 LOAD R3 0                            ;$temp = 0
145                 STOR R3 [GB +outputs + LENSLAMPPOSITION]      ;storeData($temp, 'outputs', LENSLAMPPOSITION)
146                 STOR R3 [GB +outputs + LENSLAMPSORTER]       ;storeData($temp, 'outputs', LENSLAMPSORTER)
147                 STOR R3 [GB +outputs + LEDSTATEINDICATOR]    ;storeData($temp, 'outputs', LEDSTATEINDICATOR)
148                 STOR R3 [GB +outputs + DISPLAY]            ;storeData($temp, 'outputs', DISPLAY)
149                 STOR R3 [GB +outputs + CONVEYORBELT]         ;storeData($temp, 'outputs', CONVEYORBELT)
150                 STOR R3 [GB +outputs + FEEDERENGINE]         ;storeData($temp, 'outputs', FEEDERENGINE)
151                 PUSH R5                             ;display($temp, 'display')
152                 PUSH R4
153                 LOAD R5 R3
154                 BRS _Hex7Seg
155                 LOAD R4 %0000001
156                 STOR R4 [R5+9]
157                 PULL R4
158                 PULL R5
159                                         ;unset($temp)
160                 LOAD R3 [ GB + offset + 0 ]                ;$temp = getData('offset', 0)
161                 LOAD R4 initial                       ;$temp2 = getFuncLocation('initial')
162                 ADD R3 R4                            ;$temp += $temp2
163                 ADD SP 2                            ;addStackPointer(2)
164                 PUSH R3                             ;pushStack($temp)
165                 ADD SP -1                           ;addStackPointer(-1)
166                 RTE
167
168 initial:            LOAD R3 0                            ;$temp = 0
169                 PUSH R5                             ;display($temp, 'leds2')
170                 LOAD R5 -16
171                 STOR R3 [R5+10]
172                 PULL R5
173                 LOAD R3 [ GB + stackPointer + 0 ]           ;$temp = getData('stackPointer', 0)
174                 LOAD SP R3                           ;setStackPointer($temp)
175                 BRS timerManage                        ;timerManage()
176                 PUSH R3                             ;$push = getButtonPressed(5)
177                 LOAD R3 5
178                 BRS _pressed
179                 PULL R3
180                 SUB SP 5
181                 PULL R4
182                 ADD SP 4
```

72

```
183                CMP R4 1                              ;if ($push == 1) {
184                BEQ conditional0
185 return0:                                            ;unset($push)
186                BRA initial                           ;initial()
187
188                                                      ;if ($push == 1) {
189 conditional0:      LOAD R3 0                          ;$temp = 0
190                STOR R3 [GB +outputs + HBRIDGE0]       ;storeData($temp, 'outputs', HBRIDGE0)
191                LOAD R3 10                             ;$temp = 10
192                STOR R3 [GB +outputs + HBRIDGE1]       ;storeData($temp, 'outputs', HBRIDGE1)
193                LOAD R3 1                              ;$temp = 1
194                STOR R3 [GB +state + 0]                ;storeData($temp, 'state', 0)
195                                                      ;unset($temp)
196                LOAD R2 0                              ;$sleep = 0
197                BRA calibrateSorter                    ;calibrateSorter()
198
199 calibrateSorter:    BRS timerManage                   ;timerManage()
200                CMP R2 TIMEMOTORDOWN                   ;if ($sleep == TIMEMOTORDOWN) {
201                BEQ conditional1
202 return1:          ADD R2 1                            ;$sleep+=1
203                BRA calibrateSorter                    ;calibrateSorter()
204
205                                                      ;if ($sleep == TIMEMOTORDOWN) {
206 conditional1:      LOAD R3 0                          ;$temp = 0
207                STOR R3 [GB +outputs + HBRIDGE1]       ;storeData($temp, 'outputs', HBRIDGE1)
208                LOAD R4 2                              ;$state = 2
209                STOR R4 [GB +state + 0]                ;storeData($state, 'state', 0)
210                                                      ;unset($state)
211                LOAD R2 0                              ;$sleep = 0
212                BRA resting                            ;resting()
213
214 resting:          BRS timerManage                    ;timerManage()
215                PUSH R3                                ;$startStop = getButtonPressed(0)
216                LOAD R3 0
217                BRS _pressed
218                PULL R3
219                SUB SP 5
220                PULL R4
221                ADD SP 4
222                CMP R4 1                               ;if ($startStop == 1) {
223                BEQ conditional2
224 return2:                                             ;unset($startStop)
225                BRA resting                            ;resting()
226
227                                                      ;if ($startStop == 1) {
228 conditional2:      LOAD R3 12                         ;$temp = 12
229                STOR R3 [GB +outputs + LENSLAMPPOSITION]     ;storeData($temp, 'outputs', LENSLAMPPOSITION)
230                                                      ;unset($temp)
231                BRS timerManage                        ;timerManage()
232                PUSH R5                                ;sleep(1000)
233                LOAD R5 1000
234                BRS _timer
235                PULL R5
236                LOAD R3 12                             ;$temp = 12
237                STOR R3 [GB +outputs + LENSLAMPSORTER]       ;storeData($temp, 'outputs', LENSLAMPSORTER)
238                                                      ;unset($temp)
239                BRS timerManage                        ;timerManage()
240                PUSH R5                                ;sleep(2000)
241                LOAD R5 2000
242                BRS _timer
243                PULL R5
244                LOAD R3 9                              ;$temp = 9
245                STOR R3 [GB +outputs + CONVEYORBELT]         ;storeData($temp, 'outputs', CONVEYORBELT)
246                LOAD R3 9                              ;$temp = 9
247                STOR R3 [GB +outputs + FEEDERENGINE]         ;storeData($temp, 'outputs', FEEDERENGINE)
248                                                      ;unset($temp)
249                PUSH R5 ;reset timer                   ;setCountdown(COUNTDOWN)
250                PUSH R4
251                LOAD R5 -16
252                LOAD R4 0
253                SUB R4 [R5+13]
254                STOR R4 [R5+13]                        ;set timer
255                LOAD R4 COUNTDOWN
256                STOR R4 [R5+13]
257                PULL R4
258                PULL R5
259                SETI 8                                 ;startCountdown()
260                LOAD R3 3                              ;$state = 3
261                STOR R3 [GB +state + 0]                ;storeData($state, 'state', 0)
262                                                      ;unset($state)
263                BRA running                            ;running()
264
265 running:          BRS timerManage                     ;timerManage()
266                PUSH R3                                ;$startStop = getButtonPressed(0)
267                LOAD R3 0
268                BRS _pressed
269                PULL R3
270                SUB SP 5
271                PULL R3
272                ADD SP 4
273                CMP R3 1                               ;if ($startStop == 1) {
274                BEQ conditional3
275 return3:                                             ;unset($startStop)
276                PUSH R3                                ;$position = getButtonPressed(7)
```

73

```
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                              ;if ($position == 1) {
284              BEQ conditional4
285 return4:                                          ;unset($position)
286              BRA running                           ;running()
287
288                                                    ;if ($startStop == 1) {
289 conditional3:      LOAD R4 0                              ;$temp = 0
290              STOR R4 [GB +outputs + FEEDERENGINE]       ;storeData($temp, 'outputs', FEEDERENGINE)
291                                                    ;unset($temp)
292              PUSH R5 ;reset timer                         ;setCountdown(BELT * 10)
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
301              PULL R5
302              LOAD R4 9                                  ;$state = 9
303              STOR R4 [GB +state + 0]                    ;storeData($state, 'state', 0)
304                                                    ;unset($state)
305              BRA runningTimer                           ;runningTimer()
306
307                                                    ;if ($position == 1) {
308 conditional4:      PUSH R5 ;reset timer                         ;setCountdown(COUNTDOWN)
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
318              LOAD R4 4                                  ;$state = 4
319              STOR R4 [GB +state + 0]                    ;storeData($state, 'state', 0)
320                                                    ;unset($state)
321              BRA runningWait                            ;runningWait()
322
323 runningWait:       BRS timerManage                            ;timerManage()
324              PUSH R3                                    ;$startStop = getButtonPressed(0)
325              LOAD R3 0
326              BRS _pressed
327              PULL R3
328              SUB SP 5
329              PULL R3
330              ADD SP 4
331             CMP R3 1                                    ;if ($startStop == 1) {
332              BEQ conditional5
333 return5:                                          ;unset($startStop)
334              PUSH R3                                    ;$position = getButtonPressed(7)
335              LOAD R3 7
336              BRS _pressed
337              PULL R3
338              SUB SP 5
339              PULL R3
340              ADD SP 4
341             CMP R3 0                                    ;if ($position == 0) {
342              BEQ conditional6
343 return6:                                          ;unset($position)
344              PUSH R3                                    ;$colour = getButtonPressed(6)
345              LOAD R3 6
346              BRS _pressed
347              PULL R3
348              SUB SP 5
349              PULL R3
350              ADD SP 4
351             CMP R3 1                                    ;if ($colour == 1) {
352              BEQ conditional7
353 return7:                                          ;unset($colour)
354              BRA runningWait                            ;runningWait()
355
356                                                    ;if ($startStop == 1) {
357 conditional5:      LOAD R4 0                              ;$temp = 0
358              STOR R4 [GB +outputs + FEEDERENGINE]       ;storeData($temp, 'outputs', FEEDERENGINE)
359                                                    ;unset($temp)
360              PUSH R5 ;reset timer                         ;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                                  ;$state = 9
```

```
371                 STOR R4 [GB +state + 0]                    ;storeData($state, 'state', 0)
372                                              ;unset($state)
373                 BRA runningTimer                           ;runningTimer()
374
375                                              ;if ($position == 0) {
376 conditional6:       PUSH R5 ;reset timer                   ;setCountdown(COUNTDOWN)
377                 PUSH R4
378                 LOAD R5 -16
379                 LOAD R4 0
380                 SUB R4 [R5+13]
381                 STOR R4 [R5+13]                            ;set timer
382                 LOAD R4 COUNTDOWN
383                 STOR R4 [R5+13]
384                 PULL R4
385                 PULL R5
386                 LOAD R4 5                                  ;$state = 5
387                 STOR R4 [GB +state + 0]                    ;storeData($state, 'state', 0)
388                                              ;unset($state)
389                 BRA runningTimerReset                      ;runningTimerReset()
390
391                                              ;if ($colour == 1) {
392 conditional7:       LOAD R4 10                             ;$temp = 10
393                 STOR R4 [GB +outputs + HBRIDGE0]           ;storeData($temp, 'outputs', HBRIDGE0)
394                 LOAD R4 0                                  ;$temp = 0
395                 STOR R4 [GB +outputs + FEEDERENGINE]       ;storeData($temp, 'outputs', FEEDERENGINE)
396                                              ;unset($temp)
397                 LOAD R4 6                                  ;$state = 6
398                 STOR R4 [GB +state + 0]                    ;storeData($state, 'state', 0)
399                                              ;unset($state)
400                 BRA motorUp                                ;motorUp()
401
402 motorUp:            BRS timerManage                        ;timerManage()
403                 PUSH R3                                    ;$startStop = getButtonPressed(0)
404                 LOAD R3 0
405                 BRS _pressed
406                 PULL R3
407                 SUB SP 5
408                 PULL R3
409                 ADD SP 4
410                 CMP R3 1                                   ;if ($startStop == 1) {
411                 BEQ conditional8
412 return8:                                              ;unset($startStop)
413                 PUSH R3                                    ;$push = getButtonPressed(5)
414                 LOAD R3 5
415                 BRS _pressed
416                 PULL R3
417                 SUB SP 5
418                 PULL R3
419                 ADD SP 4
420                 CMP R3 1                                   ;if ($push == 1) {
421                 BEQ conditional9
422 return9:                                              ;unset($push)
423                 BRA motorUp                                ;motorUp()
424
425                                              ;if ($startStop == 1) {
426 conditional8:       LOAD R4 0                              ;$temp = 0
427                 STOR R4 [GB +outputs + FEEDERENGINE]       ;storeData($temp, 'outputs', FEEDERENGINE)
428                                              ;unset($temp)
429                 PUSH R5 ;reset timer                       ;setCountdown(BELT * 10)
430                 PUSH R4
431                 LOAD R5 -16
432                 LOAD R4 0
433                 SUB R4 [R5+13]
434                 STOR R4 [R5+13]                            ;set timer
435                 LOAD R4 BELT * 10
436                 STOR R4 [R5+13]
437                 PULL R4
438                 PULL R5
439                 LOAD R4 10                                 ;$state = 10
440                 STOR R4 [GB +state + 0]                    ;storeData($state, 'state', 0)
441                                              ;unset($state)
442                 BRA motorUpTimer                           ;motorUpTimer()
443
444                                              ;if ($push == 1) {
445 conditional9:       LOAD R4 0                              ;$temp = 0
446                 STOR R4 [GB +outputs + HBRIDGE0]           ;storeData($temp, 'outputs', HBRIDGE0)
447                                              ;unset($temp)
448                 LOAD R4 7                                  ;$state = 7
449                 STOR R4 [GB +state + 0]                    ;storeData($state, 'state', 0)
450                                              ;unset($state)
451                 LOAD R2 0                                  ;$sleep = 0
452                 BRA whiteWait                              ;whiteWait()
453
454 whiteWait:          BRS timerManage                        ;timerManage()
455                 CMP R2 SORT                                ;if ($sleep == SORT) {
456                 BEQ conditional10
457 return10:           PUSH R3                                ;$startStop = getButtonPressed(0)
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                                   ;if ($startStop == 1) {
```

```
465                     BEQ conditional11
466 return11:                                      ;unset($startStop)
467                     ADD R2 1                          ;$sleep+=1
468                     BRA whiteWait                          ;whiteWait()
469
470                                              ;if ($sleep == SORT) {
471 conditional10:      LOAD R3 10                          ;$temp = 10
472                     STOR R3 [GB +outputs + HBRIDGE1]          ;storeData($temp, 'outputs', HBRIDGE1)
473                                              ;unset($temp)
474                     PUSH R5 ;reset timer                    ;setCountdown(COUNTDOWN)
475                     PUSH R4
476                     LOAD R5 -16
477                     LOAD R4 0
478                     SUB R4 [R5+13]
479                     STOR R4 [R5+13]                          ;set timer
480                     LOAD R4 COUNTDOWN
481                     STOR R4 [R5+13]
482                     PULL R4
483                     PULL R5
484                     LOAD R3 8                                ;$state = 8
485                     STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
486                                              ;unset($state)
487                     LOAD R2 0                          ;$sleep = 0
488                     BRA motorDown                          ;motorDown()
489
490                                              ;if ($startStop == 1) {
491 conditional11:      LOAD R4 0                          ;$temp = 0
492                     STOR R4 [GB +outputs + FEEDERENGINE]        ;storeData($temp, 'outputs', FEEDERENGINE)
493                                              ;unset($temp)
494                     PUSH R5 ;reset timer                    ;setCountdown(BELT * 10)
495                     PUSH R4
496                     LOAD R5 -16
497                     LOAD R4 0
498                     SUB R4 [R5+13]
499                     STOR R4 [R5+13]                          ;set timer
500                     LOAD R4 BELT * 10
501                     STOR R4 [R5+13]
502                     PULL R4
503                     PULL R5
504                     LOAD R4 11                               ;$state = 11
505                     STOR R4 [GB +state + 0]                  ;storeData($state, 'state', 0)
506                                              ;unset($state)
507                     BRA whiteWaitTimer                          ;whiteWaitTimer()
508
509 whiteWaitTimer:      BRS timerManage                              ;timerManage()
510                     LOAD R3 15                                ;$state = 15
511                     STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
512                                              ;unset($state)
513                     BRA whiteWaitStop                          ;whiteWaitStop()
514
515 whiteWaitStop:       BRS timerManage                              ;timerManage()
516                     CMP R2 SORT                          ;if ($sleep == SORT) {
517                     BEQ conditional12
518 return12:           ADD R2 1                          ;$sleep+=1
519                     BRA whiteWaitStop                          ;whiteWaitStop()
520
521                                              ;if ($sleep == SORT) {
522 conditional12:      LOAD R3 10                          ;$temp = 10
523                     STOR R3 [GB +outputs + HBRIDGE1]            ;storeData($temp, 'outputs', HBRIDGE1)
524                     LOAD R3 0                          ;$temp = 0
525                     STOR R3 [GB +outputs + FEEDERENGINE]        ;storeData($temp, 'outputs', FEEDERENGINE)
526                                              ;unset($temp)
527                     LOAD R3 12                          ;$state = 12
528                     STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
529                                              ;unset($state)
530                     LOAD R2 0                          ;$sleep = 0
531                     BRA motorDownStop                          ;motorDownStop()
532
533 motorDownStop:       BRS timerManage                              ;timerManage()
534                     CMP R2 TIMEMOTORDOWN                      ;if ($sleep == TIMEMOTORDOWN) {
535                     BEQ conditional13
536 return13:           ADD R2 1                          ;$sleep+=1
537                     BRA motorDownStop                          ;motorDownStop()
538
539                                              ;if ($sleep == TIMEMOTORDOWN) {
540 conditional13:      LOAD R3 0                          ;$temp = 0
541                     STOR R3 [GB +outputs + HBRIDGE1]          ;storeData($temp, 'outputs', HBRIDGE1)
542                                              ;unset($temp)
543                     LOAD R3 9                          ;$state = 9
544                     STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
545                                              ;unset($state)
546                     LOAD R2 0                          ;$sleep = 0
547                     BRA runningStop                          ;runningStop()
548
549 runningStop:         BRS timerManage                              ;timerManage()
550                     PUSH R3                          ;$colour = getButtonPressed(6)
551                     LOAD R3 6
552                     BRS _pressed
553                     PULL R3
554                     SUB SP 5
555                     PULL R3
556                     ADD SP 4
557                     CMP R3 1                                ;if ($colour == 1) {
558                     BEQ conditional14
```

```
559 return14:                               ;unset($colour)
560            BRA runningStop                ;runningStop()
561
562                                           ;if ($colour == 1) {
563 conditional14:     LOAD R4 10             ;$temp = 10
564            STOR R4 [GB +outputs + HBRIDGE0]        ;storeData($temp, 'outputs', HBRIDGE0)
565            LOAD R4 0                       ;$temp = 0
566            STOR R4 [GB +outputs + FEEDERENGINE]    ;storeData($temp, 'outputs', FEEDERENGINE)
567                                           ;unset($temp)
568            LOAD R4 10                      ;$state = 10
569            STOR R4 [GB +state + 0]         ;storeData($state, 'state', 0)
570                                           ;unset($state)
571            BRA motorUpStop                 ;motorUpStop()
572
573 motorUpStop:       BRS timerManage                ;timerManage()
574            PUSH R3                         ;$push = getButtonPressed(5)
575            LOAD R3 5
576            BRS _pressed
577            PULL R3
578            SUB SP 5
579            PULL R3
580            ADD SP 4
581            CMP R3 1                        ;if ($push == 1) {
582            BEQ conditional15
583 return15:                                 ;unset($push)
584            BRA motorUpStop                 ;motorUpStop()
585
586                                           ;if ($push == 1) {
587 conditional15:     LOAD R4 0              ;$temp = 0
588            STOR R4 [GB +outputs + HBRIDGE0]        ;storeData($temp, 'outputs', HBRIDGE0)
589                                           ;unset($temp)
590            LOAD R4 11                      ;$state = 11
591            STOR R4 [GB +state + 0]         ;storeData($state, 'state', 0)
592                                           ;unset($state)
593            BRA whiteWaitStop               ;whiteWaitStop()
594
595 motorDown:         BRS timerManage                ;timerManage()
596            PUSH R3                         ;$colour = getButtonPressed(6)
597            LOAD R3 6
598            BRS _pressed
599            PULL R3
600            SUB SP 5
601            PULL R3
602            ADD SP 4
603            CMP R3 1                        ;if ($colour == 1) {
604            BEQ conditional16
605 return16:                                 ;unset($colour)
606            CMP R2 TIMEMOTORDOWN            ;if ($sleep == TIMEMOTORDOWN) {
607            BEQ conditional17
608 return17:      PUSH R3                     ;$startStop = getButtonPressed(0)
609            LOAD R3 0
610            BRS _pressed
611            PULL R3
612            SUB SP 5
613            PULL R3
614            ADD SP 4
615            CMP R3 1                        ;if ($startStop == 1) {
616            BEQ conditional18
617 return18:                                 ;unset($startStop)
618            ADD R2 1                        ;$sleep+=1
619            BRA motorDown                   ;motorDown()
620
621                                           ;if ($colour == 1) {
622 conditional16:     LOAD R4 0              ;$temp=0
623            STOR R4 [GB +outputs + HBRIDGE1]        ;storeData($temp,'outputs',HBRIDGE1)
624            LOAD R4 10                      ;$temp = 10
625            STOR R4 [GB +outputs + HBRIDGE0]        ;storeData($temp, 'outputs', HBRIDGE0)
626                                           ;unset($temp)
627            LOAD R4 6                        ;$state = 6
628            STOR R4 [GB +state + 0]         ;storeData($state, 'state', 0)
629            LOAD R2 0                        ;$sleep=0
630                                           ;unset($state)
631            BRA motorUp                      ;motorUp()
632
633                                           ;if ($sleep == TIMEMOTORDOWN) {
634 conditional17:     LOAD R3 0              ;$temp = 0
635            STOR R3 [GB +outputs + HBRIDGE1]        ;storeData($temp, 'outputs', HBRIDGE1)
636            LOAD R3 7                        ;$temp = 7
637            STOR R3 [GB +outputs + FEEDERENGINE]    ;storeData($temp, 'outputs', FEEDERENGINE)
638                                           ;unset($temp)
639            LOAD R3 4                        ;$state = 4
640            STOR R3 [GB +state + 0]         ;storeData($state, 'state', 0)
641            LOAD R2 0                        ;$sleep = 0
642                                           ;unset($state)
643            BRA runningWait                  ;runningWait()
644
645                                           ;if ($startStop == 1) {
646 conditional18:     LOAD R4 0              ;$temp = 0
647            STOR R4 [GB +outputs + FEEDERENGINE]    ;storeData($temp, 'outputs', FEEDERENGINE)
648                                           ;unset($temp)
649            PUSH R5 ;reset timer             ;setCountdown(BELT * 10)
650            PUSH R4
651            LOAD R5 -16
652            LOAD R4 0
```

```
653                    SUB R4 [R5+13]
654                    STOR R4 [R5+13]                          ;set timer
655                    LOAD R4 BELT * 10
656                    STOR R4 [R5+13]
657                    PULL R4
658                    PULL R5
659                    LOAD R4 12                          ;$state = 12
660                    STOR R4 [GB +state + 0]                  ;storeData($state, 'state', 0)
661                                                     ;unset($state)
662                    BRA motorDownTimer                       ;motorDownTimer()
663
664 motorDownTimer:      BRS timerManage                             ;timerManage()
665                    LOAD R3 16                          ;$state = 16
666                    STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
667                                                     ;unset($state)
668                    BRA motorDownStop                        ;motorDownStop()
669
670 motorUpTimer:        BRS timerManage                             ;timerManage()
671                    LOAD R3 14                          ;$state = 14
672                    STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
673                                                     ;unset($state)
674                    BRA motorUpStop                        ;motorUpStop()
675
676 runningTimerReset:   BRS timerManage                             ;timerManage()
677                    LOAD R3 4                          ;$state = 4
678                    STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
679                                                     ;unset($state)
680                    BRA runningWait                        ;runningWait()
681
682 runningTimer:        BRS timerManage                             ;timerManage()
683                    LOAD R3 13                          ;$state = 13
684                    STOR R3 [GB +state + 0]                  ;storeData($state, 'state', 0)
685                                                     ;unset($state)
686                    BRA runningStop                        ;runningStop()
687
688                                                     ;if ($location == 0) {
689 conditional19:       LOAD R3 0                               ;$engines = 0
690                    BRA return19                          ;}
691
692                                                     ;if ($voltage > $counter) {
693 conditional20:       LOAD R4 R1                               ;$voltage = $location
694                    PUSH R5                               ;$voltage = pow(2, $voltage)
695                    LOAD R5 2
696                    BRS _pow
697                    LOAD R4 R5
698                    PULL R5
699                    ADD R3 R4                               ;$engines += $voltage
700                    BRA return20                          ;}
701
702                                                     ;if ($location == 7) {
703 conditional21:       PUSH R5                               ;sleep(1)
704                    LOAD R5 1
705                    BRS _timer
706                    PULL R5
707                    PUSH R5                               ;display($engines, 'leds')
708                    LOAD R5 -16
709                    STOR R3 [R5+11]
710                    PULL R5
711                                                     ;unset($voltage)
712                    PUSH R3                               ;$abort = getButtonPressed(1)
713                    LOAD R3 1
714                    BRS _pressed
715                    PULL R3
716                    SUB SP 5
717                    PULL R4
718                    ADD SP 4
719                    CMP R4 1                               ;if ($abort == 1) {
720                    BEQ conditional22
721 return22:                                            ;unset($abort)
722                    CMP R0 6                               ;if ($counter == 6) {
723                    BEQ conditional23
724 return23:            CMP R0 11                               ;if ($counter == 11) {
725                    BEQ conditional24
726 return24:            LOAD R3 0                               ;$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($temp, 'outputs', HBRIDGE0)
785             STOR R3 [GB +outputs + LENSLAMPPOSITION]     ;storeData($temp, 'outputs', LENSLAMPPOSITION)
786             STOR R3 [GB +outputs + LENSLAMPSORTER]       ;storeData($temp, 'outputs', LENSLAMPSORTER)
787             STOR R3 [GB +outputs + LEDSTATEINDICATOR]     ;storeData($temp, 'outputs', LEDSTATEINDICATOR)
788             STOR R3 [GB +outputs + DISPLAY]            ;storeData($temp, 'outputs', DISPLAY)
789             STOR R3 [GB +outputs + CONVEYORBELT]         ;storeData($temp, 'outputs', CONVEYORBELT)
790             STOR R3 [GB +outputs + FEEDERENGINE]         ;storeData($temp, 'outputs', FEEDERENGINE)
791                                             ;unset($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
```

79

```
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
```