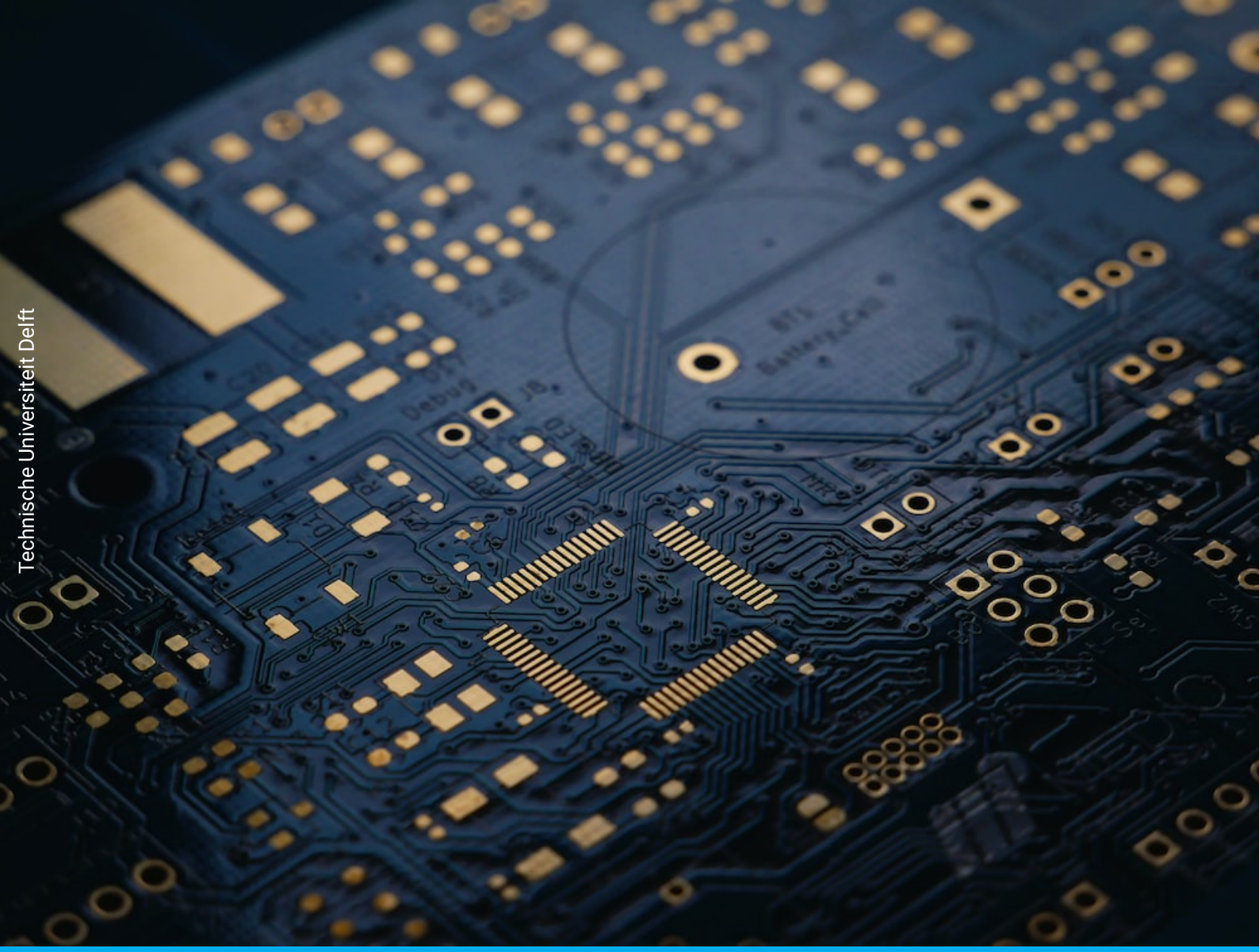

Technische Universiteit Delft

EPO 2 Final Report

Chantal Chen
Thijs van Esch
Eojin Lim
Pieter Olyslaegers
Kevin Pang
Darren van de Pol
Wilson Rong
Maximiliaan Sobry
Wiktor Tomanek

Supervisor: Dr. ir. Arjan van Genderen
TA: Nassim Beladel, Ian van Ingen, Daniel Paassen

Electrical Engineering BSc
June 16th, 2023

TU Delft

Contents

1 Introduction

For EPO-2 the goal is to develop hardware and software modules to implement a self-operating robot. This report details the design of a robot that needs to execute three challenges on a challenge grid, also called a maze, where small metal disks called 'mines' were added. An overview of the maze is described in Figure 1.1.

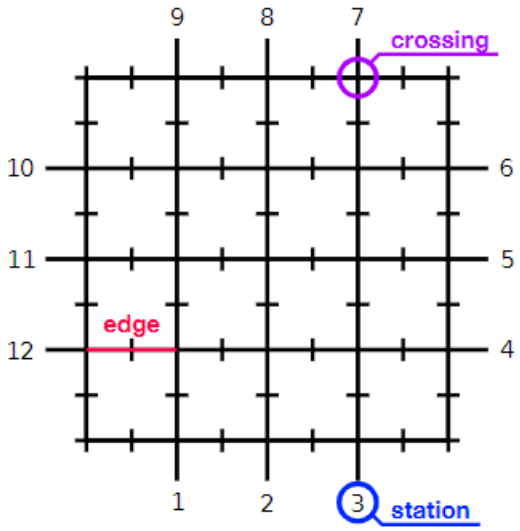

Figure 1.1: The challenge grid

The challenges that the robot must face are the following:

A The robot is given a sequence of stations it has to visit in order.

B The same as challenge A, but with mines added to the board at the midpoint of edges. The robot has to detect and avoid these mines.

C The robot has to map the entire grid and all the mines contained within. When it is finished an extra mine is added, which also needs to be found by the robot.

At the symposium, these challenges need to be completed as quick as possible by the robot. In order to do so, the robot should have the following requirements. It should detect the mines, follow and navigate on the grid (Figure 1.1) and memorize the path it took so that it knows its location all the time. The navigation and location was done by a program written in C. This program was also deciding the path the robot takes.

For the program to be able to decide the path of the robot. It has to be able to communicate with the robot. This communication will be done using the UART protocol and two Xbee modules, one attached to

the robot and the other connected to the laptop. To detect the mines from challenges B and C, a mine sensor will be made using an oscillator and a coil. In the end, all of these systems will be integrated together.

A schematic overview of the EPO-2 robot has been provided in Figure 1.2. In this overview, every module has been assigned a different colour. The corresponding chapters and the components of each module are also indicated.

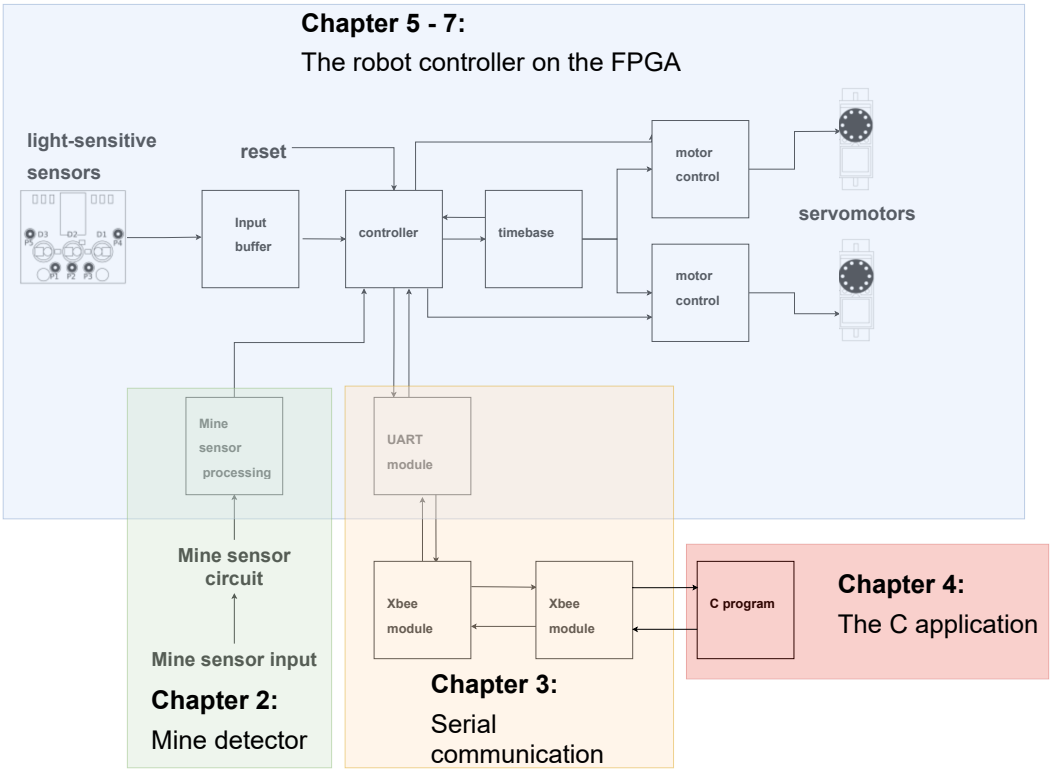

Figure 1.2: Schematic overview of the EPO-2 robot

Four different modules need to be designed and executed.

- Mine Detector: a mine-detecting sensor should be designed and created that is able to detect mines without touching them and allowing robot to avoid them. The mine-detecting sensor will be discussed in Chapter 2.
- The C application: a program with an interface and an algorithm for computing the route should be created in C. The C application will be discussed in Chapter 4
- Serial communication: an interface should be created that can guide the robot via the PC. Serial communication will be discussed in Chapter 3.
- The robot controller on the FPGA: a final state machine should be designed and written in VHDL to be able to follow the line, detect mines and execute the commands of the PC if needed. In Chapter 5, the line follower will be discussed that is part of the robot controller on the FPGA. The robot controller on the FPGA itself will be discussed in Chapters 6 and 7.

2

Mine sensor

2.1. Introduction

In EPO-2 the robot must be able to complete three different challenges. Two of these challenges are involved with obstacles in the robot's path, so-called mines. The robot must be able to detect these mines in its path and avoid them by determining a different path that can be used to reach its destination. To detect these mines in the first place the robot must be equipped with a sensor that is able to detect these mines. In this chapter, the design, testing, assembly, and implementation of such sensor will be explained.

2.2. The requirements

The 'mines' on the field are made of metal disks. Thus the first and most important requirement is that the sensor must be able to sense metal objects.
In order to sense a metal object the circuit must be built with an LC oscillator, which is another requirement. Next to only sensing a mine, the raw input signal has to be processed using code into an actual bit output which is '1' for mine and '0' for no mine.

2.3. The followed design methodology

While designing the mine detector module, the following design methodology was proceeded:

1. The circuit based on the Schmitt Trigger oscillator was designed. This oscillator was introduced in the lab assignment of the Amplifiers and Instrumentation course.
2. The oscillator was tested and the variation in the frequency was measured.
3. VHDL circuit based on the time base counter and a simple finite state machine was designed, written, and tested with the analogue circuitry. The purpose of this program is to count how long the oscillator stays in a certain state, and therefore to measure the frequency of oscillations.

2.3.1. Design

The decision to use a design based on varying inductance was made since this approach is significantly more stable and less dependent on environmental conditions than a design based on varying capacitance.

Figure 2.1 represents the final design of the circuit.

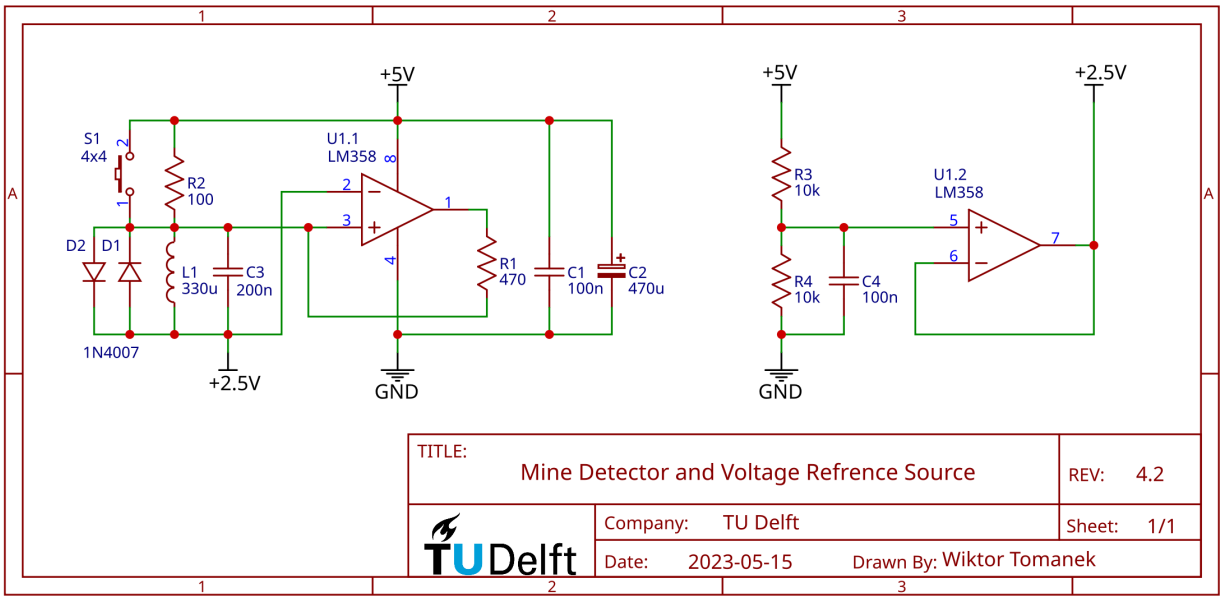

Figure 2.1: Circuit Design

The first step of designing the circuit was to choose coil L1. The goal was to choose the one that shows the biggest change of inductance when the mine is placed below it. Since that depends on the geometry of the coil and the number of possible coils was small and finite, the decision was made to choose it empirically by testing all possible coils with an RLC meter.

The value of the inductance was the less important parameter than the intensity of its variation.

The coil with 330µH was chosen. Capacitor C3 creates an LC oscillator with it. Its value was not a very important factor in the system since it only defines the oscillation frequency of the LC tank. This factor is only roughly limited by the fact that very low frequencies (<1kHz) will cause the robot to have a delay to its response and too high frequencies (>1MHz) will be affected by the variations of the environment capacitance and will therefore be unstable and hard to accurately measure. However, the ESR and ESI of the capacitor C3 play an important role in the performance of the circuit, therefore a few configurations were tested and the one with two 100nF capacitors in parallel was chosen since this reduces the ESR and ESI by half and gives a reasonable frequency of around 10kHz.

The next step was to choose resistor R1. A too-small value would cause the circuit to become very saturated and distort the wave. As given in the datasheet of LM358 (LM358 was the only op-amp that was available) - high values of gain cause high harmonic distortions. A too-big value would cause the system to not

have enough gain to overcome the dampening effects and the system would never oscillate. This component was chosen empirically to be of a value of 470 Ohm.

Switch S1 is required because sometimes the system is stable enough that it does not start to oscillate. In that case, S1 is used to "kick-start" it.

Originally S1 was used in series with a 100 Ohm resistor to limit the current however it was noticed that keeping this resistor at all times gives a much smoother waveform. This means that the circuit was still saturating a little too much, so a bit more additional dampening was required to limit the swing of the LC tank. The simplest solution in this case was to just leave this resistor connected at all times and this is the story of the resistor R2 (it is just an additional dampening).

Diodes D1 and D2 were added to make sure that the LC tank will never swing above 0.7V which is a safe peak amplitude before the circuit becomes very saturated and causes significant harmonic distortions.

Lastly, capacitors C1 and C2 provide power filtering. C2 acts as a so-called reservoir that gives high amounts of current when needed, however, it is limited in the reaction time due to the relatively high ESR and ESI of the electrolytic capacitors. C1 corrects for that, thus filtering the high-frequency noise.

One side of the oscillator tank is connected to the output of the op-amp and the other one requires to be connected to the bias voltage of the circuit which ideally is in the middle of the power rails therefore 2.5V in this case, to provide the highest possible output swing. LM358 is a dual op-amp IC, thus the unused amplifier was utilized as a buffer for the reference voltage created with a voltage divider consisting of R3 and R4. Capacitor C4 was just added for additional filtering of any potential noise at the output of the voltage divider.

All capacitors were chosen to have a voltage rating higher than 5V, so they are safe to use in this circuit. Power ratings of the resistors do not play an important role in the design, because currents in the circuit are very low, thus all resistors have 1/4W ratings. Tolerances of the components do not matter because the entire system can be calibrated in VHDL. (Assuming that they are in a range of modern standards.)

2.3.2. Simulations

The designed circuit was first simulated using the LTSpice. It was tested whenever the output of the circuit is indeed as expected, and whenever the frequency of the square wave output is indeed changing along with the change of the inductance, which corresponds to the sensor sensing a metal object.

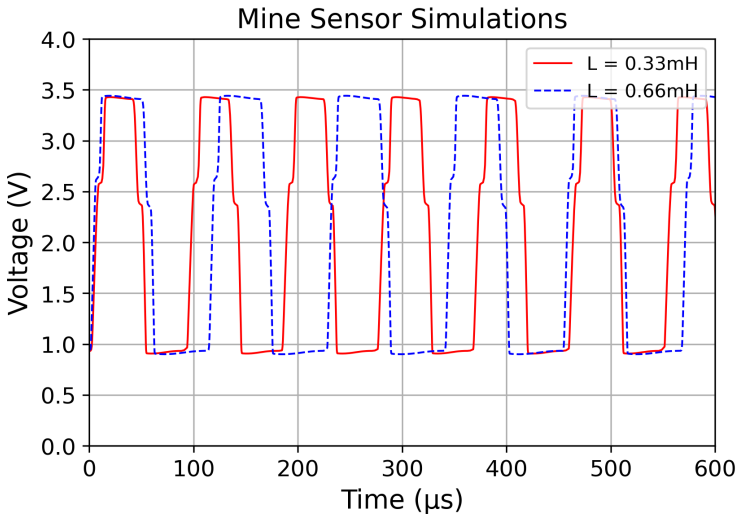

Figure 2.2: Circuit Simulations

Results of the simulations have shown the output of the circuit is indeed a square wave and the frequency of this square wave changes with a changing inductance value. In this case, with a doubled inductance value from $L = 0.33mH$ to $L = 0.66mH$ it is clearly shown in Figure 2.2 that the frequency of the square waves changes as expected. In the presence of the metal plate, the inductance of the inductor increases, giving a bigger period for the square wave. In reality, the increase of inductance in the presence of the 'mine' is around +10%, this was measured using the RLC meter.
In this simulation, $L = 0.36mH$ is not given in the graph since the change in frequency is small and hardly noticeable. Nevertheless, it doesn't matter that this value is not simulated since the main behaviour of; a changing period in the presence of a changing inductance is the most important aspect, which can also be simulated using $L = 0.66mH$.
To sum up, the simulated output has a square wave function in which the period changes along a changing inductance. The simulated circuit works as expected.

2.4. Implementation

2.4.1. Code

After the sensor has been designed and simulated it will be built and integrated with the robot. However, simply connecting the output of the sensor to the FPGA board is not yet enough. The output of this sensor still has to be processed to determine if a mine is detected or not.
With the use of some VHDL code the square wave output can be processed into determining whether there is a mine detected or not. As can be seen in the simulations, in Figure 2.2, the output of the sensor is a square wave function. The wave essentially has a high voltage and a low voltage, which the FPGA board reads as a '1' for high and '0' for low voltage.
Since the '1' and '0' keeps their value for a few microseconds it is possible to count the amount of ones or zeros in 1 period. In the presence of a metal mine, the inductance will increase which causes the period to increase, which means that the amount of time the wave is '1' or '0' holds in a period increases. Essentially, if it is known how many ones or zeros fit in one period without a mine, that value can be compared with the number of ones or zeros in a period with a mine.

Therefore the implemented code has a counter which in this case counts the amounts of zeros. In reality, it does not matter if the ones or zeros are counted since the change in count due to a mine is the important factor. The code is implemented using an FSM. It has 4 stages: a reset stage, a counting stage, a compare stage, and an output stage. In communication with the other EPO-2 groups the output stage, which is the stage that sends a '0' or '1' as output signalling that a mine had been detected, keeps on being on this stage for 10 milliseconds. This means that besides the FSM, there is also another code with a time counter which sends a reset '1' signal to the FSM after the 10 milliseconds have passed. This results in the FSM going back into its first stage, essentially resetting itself to start counting and detecting mines again. In the compare state, the number of zeros counted in one period gets compared to a set threshold. If it exceeds the threshold a '1' as output is given in the detected state to the rest of the system, signalling a mine has been detected. The threshold for the counting in the code will be determined in the testing phase.

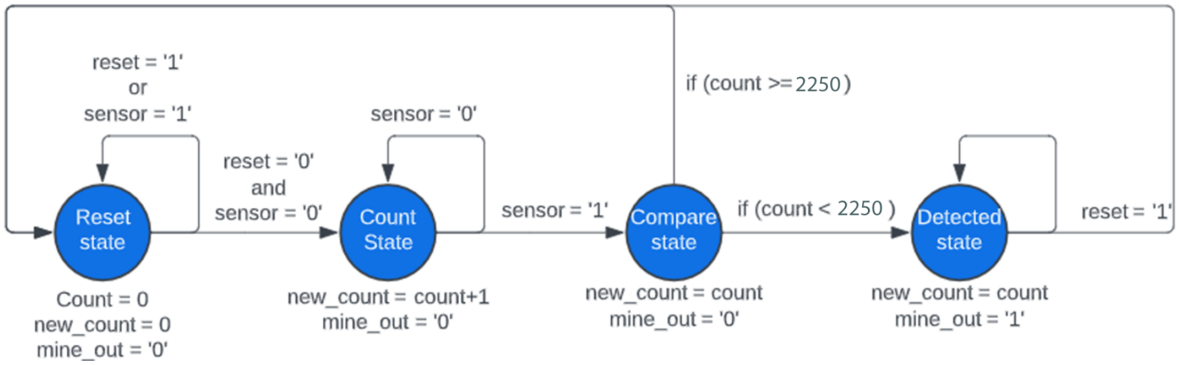

Figure 2.3: Finite State Diagram of the Sensor

In Figure 2.3 above a Finite State Diagram is shown of the code. And in Appendix E the full VHDL code for the mine sensor can be found.

2.4.2. Physical layout

The sensor was assembled on a separate prototyping PCB that connects to the robot as a module, as seen in the Figures 2.4 and 2.5. This solution allowed the circuit to be easily removed and tested separately or modified if needed.

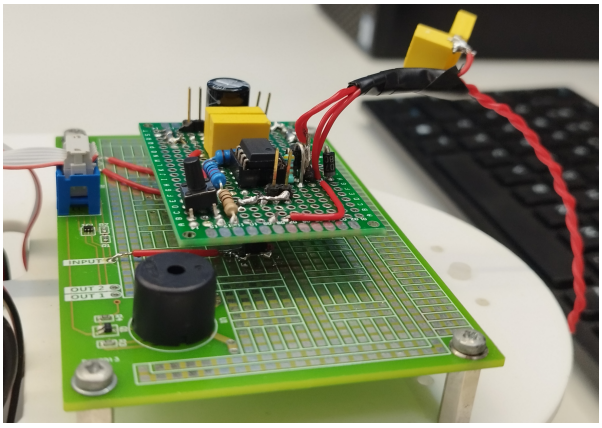

Figure 2.4: Side view

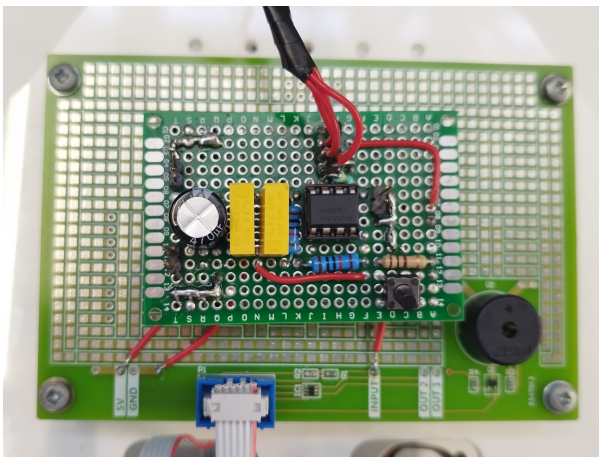

Figure 2.5: Top view

For low ESR applications, foil capacitors were chosen since they were available with low voltage specifications, thus they had compact dimensions.
The last step was to mount the coil using nylon struts and screws as seen in the Figure 2.6. Nylon was chosen since it is a nonmagnetic material and does not interfere with the operation of the oscillator. The spacing between the coil and the table was precisely adjusted with small washers and nuts. The goal was to have the coil as close to the table and the mine as possible for the highest sensitivity, but not too close, since that might cause the robot to push the mine away.

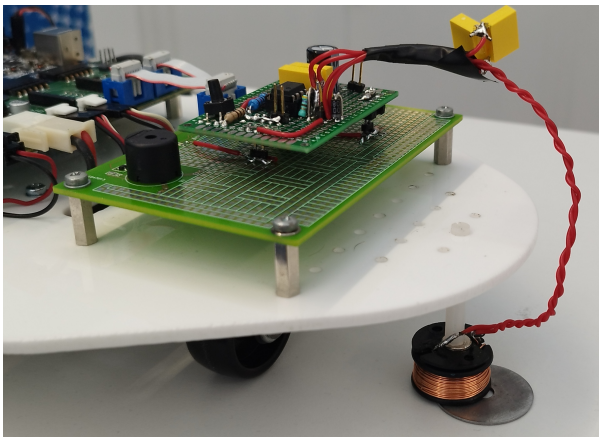

Figure 2.6: Coil view

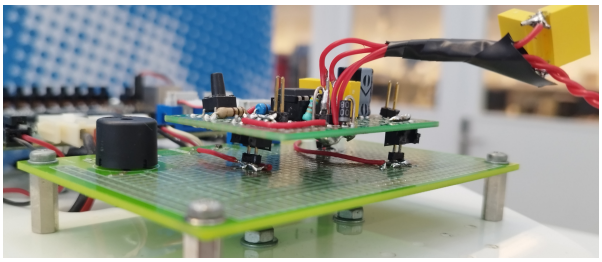

Figure 2.7: Front view

2.5. The testing phase

2.5.1. Physical circuit testing

After assembling the sensor circuit, it was first tested whether the circuit output gives a correct square wave output as seen in the simulations.
Using a probe connected to the output, the output was measured.

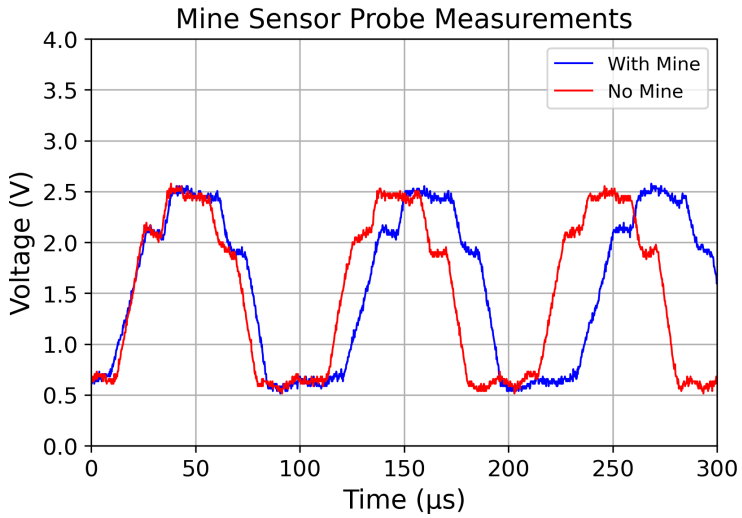

Figure 2.8: Circuit Probe Measurements

As seen in Figure 2.8 the output is mostly a square wave function. The FPGA board considers voltages above 1.2V as a '1'. So the two 'steps' are seen as a '1'. Essentially when looking carefully at the measurements, the amount of time that the signal is a '1' is longer than it is '0' in a period. This is not problematic since the change in the period under the influence of a mine is important, therefore the shape of the 'square' wave signal is fine to work with.
In Figure 2.8 the square wave is plotted with and without a mine. The graph shows a clear change in frequency and with a mine, the period is bigger. Which is expected from the simulations. From this, it can be concluded the sensor sufficiently senses the presence of a mine.

After testing if the built circuit worked as intended. The next step was to determine how many times '0' was counted in 1 period. With this information, a threshold value can be set in the VHDL code.

The FPGA boards run on a 50Mhz clock which has a 20 nanosecond period. It must be determined how many of these periods fit into the '0' state of the output square wave signal of 1 period. The time the signal is below 1.2V could be measured with an oscilloscope. However, the decision was made not to proceed with this approach and instead use the FPGA board directly to count the '0' state of the square wave. In the unlikely event that the circuit may behave a bit differently on the FPGA board. The value counted by the FPGA board is more accurate than the one with a probe-determined count. This method also compensates for the fact that FPGA is still counting while the signal is on one of the edges since the voltage has to cross the threshold.

2.5.2. Sensor FPGA board implementation testing

A simple VHDL code that counts how many periods of 20 nanoseconds fit into the low state of the square wave for 1 period was written, with a bit vector as output. This output was displayed on the FPGA board using the led lights on the board. This gave 2112 counts without a mine and 2400 counts with a mine. After further testing with the code and taking into account the psychical placement of the sensor, the final threshold value of 2250 counts was chosen.
After a threshold was chosen, the sensor with the VHDL code could be tested. After the VHDL Code was implemented on the FPGA board. Two states, the counting stage, and the compare state were checked whether they work as intended, by designating an LED to each state and connecting the probe to them to see the states on the oscilloscope.

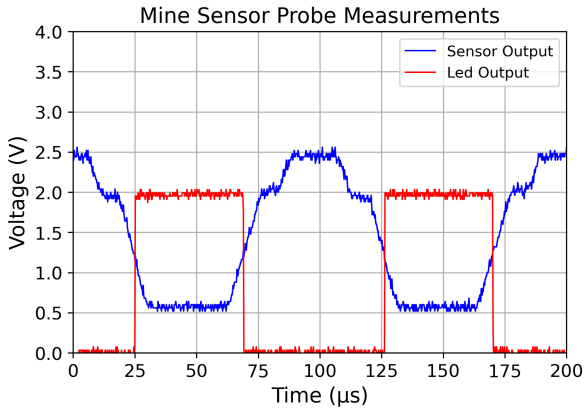

Figure 2.9: Count state

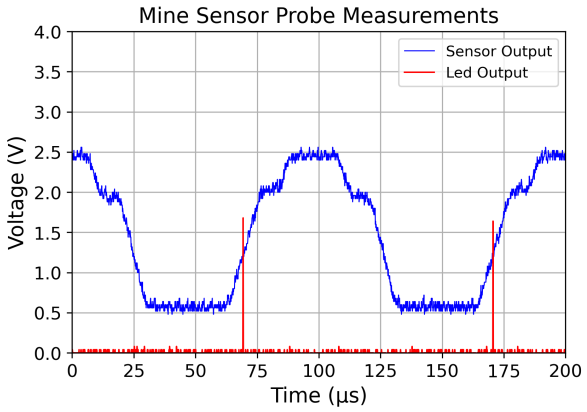

Figure 2.10: Compare state

As can be seen in Figures 2.9 and 2.10 the counting state happens when the square wave signal is '0' and the compare state at the end of the '0' signal when the square wave jumps to a higher voltage. This shows the VHDL Code is indeed counting when the square wave signal is low and after counting immediately goes to the compare state. Out of visual observations where the output in the output state is assigned to a LED. It could be seen that the LED lights up when a metal object is placed below it and stays off when there was no metal mine below the sensor. From these observations and measurements, it can be said confidentially that the sensor and the code work as intended.

3

Communication

The controller must be able to communicate with the robot via a wireless connection. It was decided that this communication be done using *XBee* modules. Such modules use their own protocol to communicate with each other while using the well-established UART protocol to communicate with external hardware. This means that both the controller, which runs on a desktop computer, and the robot must implement the UART protocol to communicate with each other. How this was achieved, is described in this chapter.

3.1. The requirements

This section describes the requirements relating to the communication for both the robot and the controller.

3.1.1. The robot

In the manual, it was given that the robot has to use XBee modules with IEEE 802.15.4 standards. These modules support the needs of low-cost and low-power wireless sensor networks. This means that the XBee modules are able to provide reliable delivery of data between devices with minimal power required. They operate within the ISM 2.4 GHZ frequency band and have an indoor range of 30 meters.

With these XBee modules, the robot was required to read incoming data from the PC and write specific data to the PC. In order to let the robot communicate with the XBee module, it will need a VHDL code for a UART transceiver. This code was given on Brightspace and only needed to be implemented with the VHDL Robot top entity.

3.1.2. The controller

If an XBee module is connected to a desktop computer via a USB cable, it appears as a serial device. This means that the controller must have access to the following functions.

- A function to initialize the serial interface.
- A function to write a single byte to the serial interface, after this interface has been initialized.
- A function to read a single byte to the serial interface, after this interface has been initialized.
- A function to properly close the connection to a serial device, again after the interface has been initialized.

These functions must be implemented for the MacOS (BSD) and Linux platforms (both of which luckily use the same interface, because Unix is the best OS, allowing the code for the serial communication to be platform-independent for these two platforms) using the C programming language. It is most certainly not a requirement for this code to function when ran on the Microsoft Windows operating system.

3.1.3. Communication protocol

For the robot and the controller, only having the ability to read and write data does not allow them to communicate. It must be clear what received data actually implies for both the sender and the receiver. This means that a proper communication protocol must be designed. This protocol must meet the following requirements.

- It specifies when incoming data should be expected and when data should be sent for both the robot and the controller.
- It specifies what the robot should do with the data it receives from the controller.
- It specifies what the controller should do with the data it receives from the controller.

3.2. The followed design methodology

This section describes how the robot's UART module and the serial interface were designed.

3.2.1. The robot

The VHDL code that directly operates the XBee modules was already provided. To implement and use this code, the following flags and signals had to be used.

- Outputs:
  - `data_out`: This output is a vector of 8 bits and contains the data that has to be sent.
  - `write_data`: This is the flag that has to be set high to write data.
  - `read_data`: Using this output, data can be read.
- Inputs:
  - `data_ready`: This flag is set high by the UART code when it has successfully received data.
  - `data_in`: This byte-long vector contains the data sent by the laptop.

Next to these signals the RX and TX signals from the UART file also had to be properly integrated with the top entity. By assigning these signals and flags the robot can use the UART code to communicate with the laptop.

3.2.2. The controller

Since reading and writing data to a serial port is not such a nonstandard thing to do, code to achieve this could be found online fairly easily. Code snippets found online should be referenced from where within the application code they were used. These code snippets were modified to obtain the four functions that were required for the controller to have access to. These functions would be named `init_sio`, `write_byte`, `read_byte` and `close_sio` respectively.

3.3. Implementation

This section describes how the robot was programmed to use the UART module, how the controller implemented the serial interface, and what communication protocol would be used for communication between the robot and the controller.

3.3.1. The UART module for the robot

Serial communication via an XBee module is done by the robot using the UART (universal asynchronous receiver/transmitter) protocol. It was decided to make use of serial communication with the following properties.

- The rate at which information is transferred is 9600 baud.
- Each frame contains 8 data bits.
- No parity bits are used.
- Each frame ends after a single stop bit.

3.3.2. Asynchronous serial communication

The serial communication is asynchronous, which means that the clocks of the transmitting and receiving systems are not synchronized. This leads to a problem that the receiving system does not know exactly when a bit of information is arriving over the RX signal. To solve this problem, there will be two additional bits placed which are called the 'Start bit' and the 'Stop bit'. The Start bit will be placed in front of the 8 bits data with useful information and the stop bit will be placed at the end of the 8 bits data. These signals do not carry important data but allow the receiver to synchronise the 8 bits of data. Exchanging data with UART modules goes in several steps. These steps go as follows;

1. With no incoming data or outgoing data, the communication line is in 'BREAK' state with signal level '0'
2. if the communication starts, a moment before the communication line will go into 'IDLE' state with signal level '1'
3. The communication starts with the synchronization of the UART receiver. It synchronises itself to the falling edge of the start bit which has a signal level '0'. Afterwards, the receiver will start to measure the amount of time elapsed to sample every data bit. Due to the transmission speed being known beforehand, the receiver knows how long each data bit lasts, which allows sampling. To sample accurately, the sampling is done in the middle of each bit.
4. The communication ends with a STOP bit. Its length is one to two times the length of a data bit and can be used as a check for signal integrity. after the stop bit an 'IDLE' signal will occur of any length. The process can now start over again.

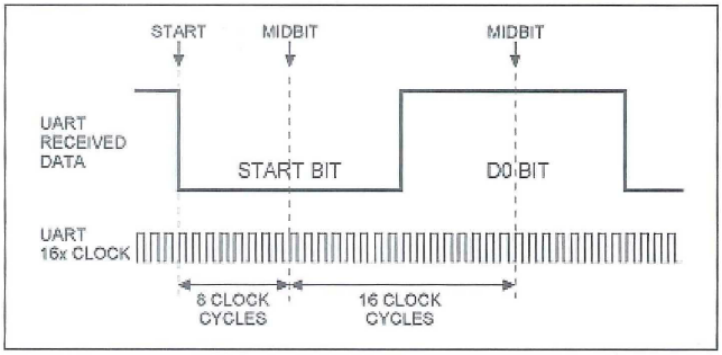

Figure 3.1: frame synchronisation and data bit sampling points

3.3.3. Testing and setting up the robot's ability to communicate

To be able to write between two XBee modules without interference, the modules had to be linked. This can be done by specifying the ID, channel, local and destination address. The software XCTU can be used to configure these devices. When the link is established and the XBee modules are connected to XCTU on two different laptops, data packets could be sent between two laptops.

This method was used every time a problem was encountered. By setting up one XBee with the XCTU and the other with either the robot or the C application one could check whether communication worked manually.

3.3.4. The serial interface

The implementation of the serial interface can be found in Appendix A.3.

3.3.5. The communication protocol

It was decided that the robot should send only send data when this data is needed by the controller. This data could be needed by the controller to calculate the most optimal path, update the current robot status or other things. There are three possible situations the robot can be in causing it to send data to the controller.

1. The robot has encountered an intersection. This would cause the robot to send a byte with value 2.

2. The robot has encountered a dead end. This always means that the robot has 'entered' a station. This would cause the robot to send a byte with value 3.
3. The robot has encountered a mine. This would cause the robot to send a byte with value 4.

The controller will only send data to the robot after it has received a single byte from the robot. This means that the robot may expect to receive a single byte for every status update it sends the controller. Each byte that the controller writes to the serial port is an instruction for the robot. There are four possible instructions that can be sent to the robot. One of these is named `ROBOT_UTURN`. An instruction to make a U-turn will be sent if the robot has just encountered a dead end or if the robot has encountered a mine. A different instruction is `ROBOT_STOP`. The controller should only send this instruction if the robot can stop doing everything it is doing. The robot should then loop in the same state until it is reset.

All other instructions exist to allow the controller to tell the robot to go forward, to the left or the right when it is at an intersection. These instructions are listed in the following table.

Opcode	Bit value	Description
`ROBOT_LEFT`	`0b001`	Go left at the next intersection.
`ROBOT_RIGHT`	`0b010`	Go right at the next intersection.
`ROBOT_FORWARD`	`0b011`	Continue driving forward at the next intersection.
`ROBOT_STOP`	`0b100`	The robot should stop immediately.
`ROBOT_UTURN`	`0b101`	The robot should make a U-turn immediately.

Table 3.1: Opcode with their bit value and description

3.4. The testing phase

To check whether the ZigBee modules were functional and ready to be used it was first tested if the code written to send and receive bytes from the controllers side actually worked. Then, it was verified that the robot was able to use the modules to communicate with the controller.

3.4.1. Testing the serial interface

A test program was written for a desktop computer in order to test if the controller is able to send and receive bytes using an XBee module.

This test program did two things after the serial interface has been initialized. First, it writes each lowercase letter in the alphabet to the serial port. Then, it keeps reading bytes from the serial port until the program is interrupted or some large number of attempts to read a byte have been made. By connecting one XBee module to the test program and connecting to another module using the GNU 'screen' utility it can quickly be established whether the test program does what it is supposed to do.

The program can be found in Appendix B.2.

4

The C application

The goal of the project is for the robot to be able to complete all three assignments. This goal is to be achieved by integrating many different subsystems. One of these subsystems is an application, written using the C programming language, running on a desktop computer.

This program makes use of several algorithms and data that were received from the robot to determine instructions that the robot should execute. In this chapter, the manner in which the application was written, designed and tested will be discussed.

4.1. The requirements

The application has to send the robot to where it needs to go. It does this by writing data (instructions) to a serial port. These instructions will then be sent wirelessly to the robot.

From the viewpoint of the robot, the application requirements boil down to the controller having to send an instruction each time the robot sends data about its state. The application may instruct the robot to make a U-turn or it may direct the robot to go left, right, or forward when it reaches the next intersection.

The application can determine where to send the robot from the arguments given to the program. The program must be able to read key-value pairs and name-only values given to it via the command line. It must also minimize its reading of user input via standard input. This gives the user of the program the ability to programmatically determine the desired arguments instead of having to manually enter them each time, improving the usability of the program.

Another requirement is that the program must have an exit status that either indicates that the program was run successfully or indicates exactly what caused the program to fail. This makes it easier to find errors and troubleshoot them.

Everyone writing code for the program must also make sure that all code that has been written is formatted similarly.

4.2. The followed design methodology

This section describes how the program was written, how the application was structured, and what algorithms were used for determining the most optimal path for the robot to take.

4.2.1. Code styling

Instead of just focusing on a working application, much attention was also given to the quality of the software that was written. This is because software of higher quality is generally easier to debug and maintain. This means that even though it may take longer before the application can be tested, debugging afterwards will be done much quicker. To increase the quality of the written software, the following things were done.

- It was made sure the code style was consistent. This was achieved with the use of the `clang-format` utility, which is included by default with the Clang compiler. This utility allows the specifying of code formatting rules in a file, ensuring that all written code is formatted the same. This `.clang-format` can be found in Appendix D.
- Writing tests that can be used separately from the main application.
- Providing error codes where appropriate, so if something fails we can more easily see what went wrong.
- Splitting the code for the application across multiple files, each containing a set of related functions and other definitions, only including other definitions when required.

4.2.2. Build system

For compiling the code written for the application the open-source build environment *CMake*. CMake is not a compiler. Rather, it is a system that generates another system's build files.

CMake can be configured by writing directives in the build script named `CMakeLists.txt`. This file can be found in Appendix C. This file specifies some build options, sets the compiler flags that enable all warnings and the flags that turn these warnings into errors. Each executable (one for the main application and one for each application test) is specified in this file separately.

4.2.3. The algorithm

For challenges A and B the path that the robot should take is simply the shortest path between its current location and its destination. To determine this path, the Lee algorithm was used.

The Lee algorithm can be used to determine the shortest path between two cells of a two-dimensional matrix. For our robot, this matrix is an array containing 13 rows and 13 columns. Each cell where the robot may not go initially gets assigned a value of -1, while cells, where the robot is allowed to go, are assigned a value of 0. What follows is an in-depth description of how this algorithm functions.

The Lee algorithm works on a 2-dimensional array. For the board the robot will be driving on this array will be a matrix containing 13 rows and 13 columns. Each cell represents either a station, a crossing, or an edge. The remaining cells will have the value -1.

The first thing that will be done by the algorithm is to set the value of the destination cell to 1. Then, the neighboring cell will be assigned the value of the destination cell incremented by one. This process will be repeated for each cell: for each cell with the value i each of its neighboring cells, if no value has been assigned to this cell and the robot could possibly move to that cell, will be assigned the value $i+1$.

For challenge C the Lee algorithm wasn't optimal, because it is used for finding the shortest path. And for the last challenge every mine cell needs to be visited exactly once to find all the mines in the maze. Therefore it was decided to use a Hamiltonian Cycle.
This algorithm represents a closed loop in the maze where every node(vertex) is exactly visited once. The cells where a mine could potentially be found are replaced by a vertex. See Figure 4.4
For the use of the Hamiltonian cycle, an adjacency matrix needs to be created for the maze. This adjacency matrix is represented in Figure 4.2.
The adjacency matrix indicates with the Boolean value 1 if there is a direct path between two vertices and the Boolean value 0 indicates that there is no direct path. see Adjacent Matrix F.1
For finding the shortest path, we start by calculating the route from the closest vertex (mine cell) to the starting point in the maze. The next vertex is a vertex that is not already taken and has an edge between both vertices. This can be checked by the value 1 in the adjacency matrix. The process is repeated until a cycle is found and the last vertex is the starting vertex.
Finally, the robot starts tracing back the Hamiltonian cycle and visits all the vertices (mine cells) exactly once. Every time a mine is found the Hamiltonian cycle is recalculated with the remaining vertices.

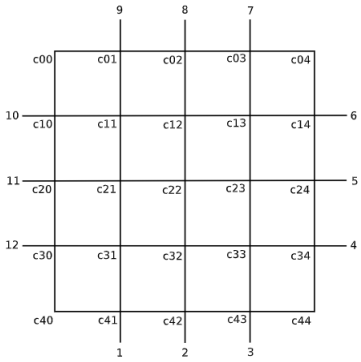

Figure 4.1: 13 by 13 array

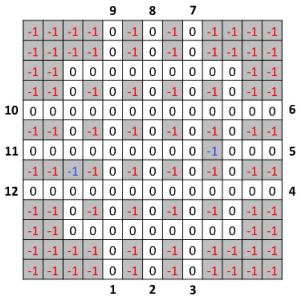

Figure 4.2: maze field

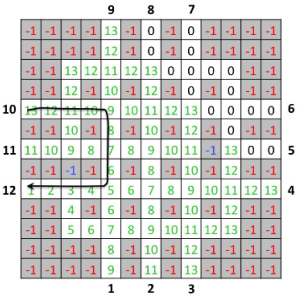

Figure 4.3: filled maze field

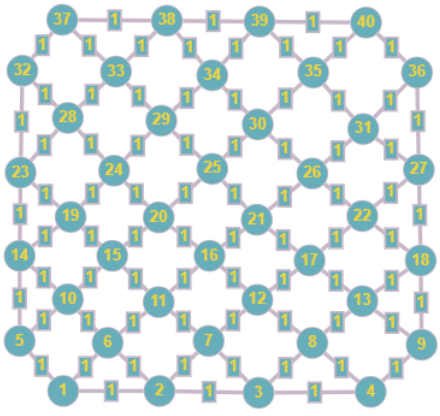

Figure 4.4: Hamiltonian cycle [2]

4.3. Implementation

In this section, it is discussed how the previously discussed design requirements are met.

4.3.1. Application structure

The application has been divided into several modules. Each of these modules is listed separately as a section of Appendix A. Figure 4.5 provides an overview of how data flows to and from each module in the application.

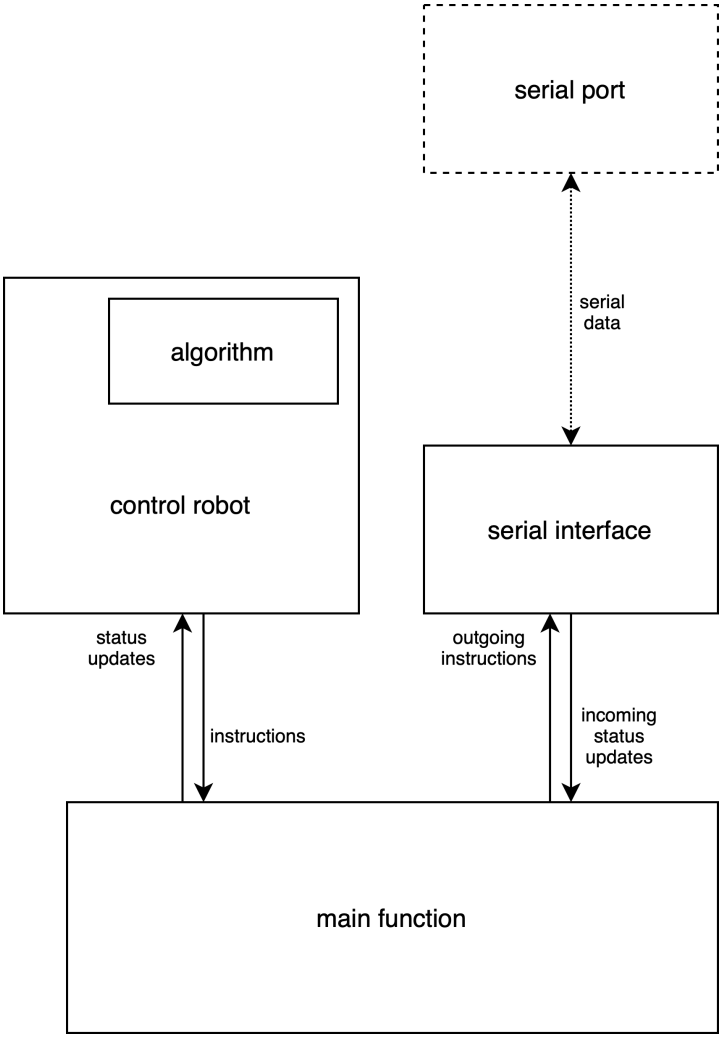

Figure 4.5: Flow of data within the application

4.3.2. Pathfinding

This section covers how the Lee algorithm was implemented.

A basic queue system was implemented in C. This system was used to queue the cells that have neighboring cells that were not given a value yet. Then, the Lee algorithm was implemented as the `calc_path` function.

Due to a lack of time, the algorithm for challenge C could not be implemented. It is expected that any implementation would function as described in the requirements section.

4.3.3. Robot control

The robot control is used directly from the file containing the application's main function. Like most other modules in the application, it has an initialization function that initializes the module. The implementation

can be found in Appendix A.

4.4. Testing

The testing that was done for the application can be divided into two categories.

- Tests without the robot.
- Tests with the robot.

4.4.1. Application testing

Some parts of the application could be tested without the robot.

- The queue system that was used by the algorithms.
- The algorithm(s) that calculate the most optimal path for the robot to take.
- The functions that parse the arguments given to the application.

A test program was written for each of these points. It used the interface it was testing in the way it was intended to check whether it functions correctly. If an unexpected value was encountered in any of the tests, the program would print what value it expected and the actual value along with an error message before exiting with an error code. All test programs can be found in Appendix B.

Each of these tests used the same functions for testing values. These functions were declared in a header file that was included by each source file containing a test.

A shell script was written to run all of these tests automatically. If any of the test programs exited with an error code the script return with a non-zero code, meaning it failed. Using this script it was easy to check whether changes made to the application caused errors in parts of the application that was expected to function correctly. This shell script can be found in Appendix B.4.

The application currently functions as expected. This is indicated by the aforementioned script not indicating that an error occurred.

4.4.2. Robot testing

The application as a whole is difficult to test without actually having the robot connected. Because of this, testing was also done with the robot connected.

Once testing with the robot started many errors were encountered. Some of these errors were caused by the application receiving data from the robot it did not expect. This was however not an error caused by the application code, but by the robot. Once some problems relating to the robot were fixed, there was not enough time left to debug the C application.

5

The line follower

5.1. Introduction

In Chapters 5, 6 and 7 the whole process of designing and testing the VHDL code of the robot will be discussed. However, in order to be able to write a VHDL code for the robot, some basic understanding of the physical parts of the robot as well as some knowledge about a line follower is needed. Since the physical components of the robot (except for the mine-sensor) are given and have been discussed in Chapter 2 of the line follower manual [3] in the course EE1D21 of TU Delft, the physical parts of the robot will not be elaborated further upon in this report. On the other hand, the line follower and its code will be elaborated on, since the VHDL code of the robot is an extension of the line follower.

The general idea of the to-be-designed line follower is to enable a robot to generate the correct signals to control its motors based on the current input of a light-sensitive sensor system, thus enabling it to follow a line. In order to do this several components have to be designed in VHDL. These components are a time base, an input buffer, the motor controls and a (general) controller. In this chapter, the reasons behind the importance of these components and their functionality will be described.

5.2. The requirements

When designing the components, the specifications of the physical parts of the robot have to be taken into account. Therefore, components that have to be designed need to meet several requirements. While there are many specifications of the physical parts of the robot, the most important specifications to keep in mind when designing the VHDL code are that the robot runs on a 50 MHz system clock, the light-sensitive sensors are constantly sending data and that the motors are controlled by pulse width modulation (PWM). The motors that are used are controlled by a pulse sent every 20 ms. [3] With these specifications in mind, the requirements for the components of the VHDL code can be defined. In this section, those requirements will be briefly addressed.

- Time base: should be able to count up to a period of a PWM pulse and get reset by a synchronous reset.
- Input buffer: should ensure that sensor values are not changing during a clock cycle.
- Motor controls: should send the correct PWM signal to control the motors.
- Controller: should determine the direction (and speed) the wheels are turning to, based on the sensor values.

These requirements are taken from the provided line follower manual [3] and should also be kept in mind when designing the VHDL code of the EPO2 robot, which will appear in Chapters 6 and 7.

5.3. Design and Implementation

In Figure 5.1 an overview of the line follower provided by the line follower manual [3] is given. Based on this overview and the given requirements, the VHDL code for line follower was designed and implemented.

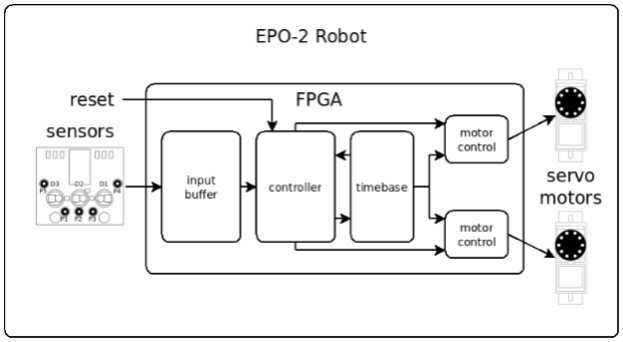

Figure 5.1: Overview of line follower that is used for EPO-2 [3]

It is noticeable that the sensor values of the light-sensitive sensors enter the robot via the input buffer, which then forwards those signals to the controller. The controller in turn sends signals to the time base and motor controls. Lastly, the motor controls generate the right PWM signal so the motors start turning. During this, the time base counts the time and its output is used in the controller and motor controls. In the following sections, an explanation of the light sensors and these four components will be given.

5.3.1. Light-sensitive sensors

Light-sensitive sensors are mounted on the bottom of the robot. They measure the light reflected from the ground and set the output to '0' or '1' when the surface beneath the sensors is black or white respectively [3].The sensor system consists of three sensors, sensor_l(eft), sensor_m(iddle), and sensor_r(ight), that are attached horizontally such that they are parallel with the rear wheel shaft.

The robot always wants to have the line in the middle, which means the sensor values for sensor_l, sensor_m, and sensor_r are '1', '0' and '1' respectively (short notation: '101'). When the sensor values deviate from these "optimal" values, the robot ought to steer itself back on track until these values are met again.

5.3.2. Time base

As mentioned, a time base needs to be created for the robot to measure the time within the period of a PWM pulse. In order to achieve this, the time base will be implemented with a counter. The whole time base is shown in Figure 5.2, taken from the line follower manual[3]. It is visible that the time base is implemented with two process blocks. It repeatedly counts till 20 ms (the period of a PWM pulse). This is realised by using a register which can store the counted value for more than one clock period. The 'add_1' block adds a value 1 to that counted value after each clock cycle, given the time base is not reset.

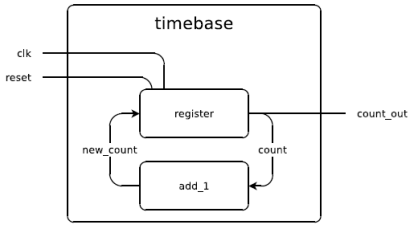

Figure 5.2: Schematic representation of the time base [3]

Since the time base adds a value of '1' each clock cycle, the length of the bit-vectors for the signals, new_count, count and count_out is obtained as follows: The frequency of the global clock is 50MHz, which means that the period is $\frac{1}{50\times10^6}$ = 20 ns. This means that in order to know when a 20 ms period has period has passed, the counter needs to count up to at least one million ($\frac{20\times10^{-3}}{20\times10^{-9}} = 1\times10^6$). From this, it is decided that

a 20-bit signal should be used because 2^{20} has a value that is just above one million. Along with the VHDL knowledge from the line follower manual, the VHDL code of the time base can be written. The VHDL code of the time base can be found in Appendix G.1.

The counter is an essential part of the line follower, as the duration of the PWM signal generated from the motor control gives the direction of the wheels. More about that will be explained in the subsection Motor Control.

5.3.3. Input buffer

The input signals should not change during a clock cycle. Therefore, to ensure that the input signals for the FSM are stable and an input buffer has been made. This input buffer consists of two 3-bit flip flop, which is placed between the sensors and the controller see Figure 5.1.

The sensor_l_in, sensor_m_in and sensor_r_in are used as inputs of the input buffer and then the 3-bit registers use this input and transfer it to the controller as can be seen in 5.1 and 5.3. This does mean that the output values of the input buffer are not the real-time values, but rather the sensor values from two clock cycles earlier. However, this means the controller component will use data from two clock cycles (=40 ns) earlier. This will cause no problems since 40 ns is way too fast for the motors to react to. This means that in 40 ns the robot will not have changed its position such that its sensor values differ from the sensor values that would have been observed 40 ns earlier.

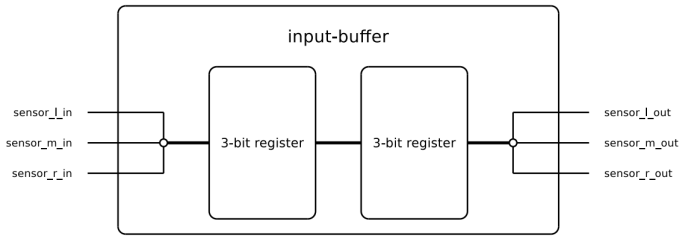

Figure 5.3: Schematic representation of the input buffer[3]

As mentioned, the input buffer is implemented with two 3-bit flipflops. For which the VHDL code can be found in Appendix G.7. Combining two of these components will result in the input buffer described above. The code for that can be found in Appendix G.5.

5.3.4. Motor control

The motor controls are responsible for making the motors turn the wheels clockwise (cw), counterclockwise (ccw) and turning the motors on or off. As described earlier, the motors are controlled by a PWM signal. This signal must have a period of 20 ms and a duty cycle between 5% and 10% [3]. This means that the pulse must be high at least 5 to 10% of the period time if you want the motors to drive the wheels. This knowledge combined with the given notion that a pulse shorter than 1.5 ms will cause the motors to turn counterclockwise and a pulse longer than 1.5 ms will cause the motors to turn clockwise, enables the reader to decide on the duration of the signals the motors should output in order to drive the wheels. Namely, the durations of 1 ms and 2 ms have been chosen to drive the motors counterclockwise and clockwise respectively. These two values satisfy the duty cycle and differ enough from 1.5 ms, so that no ambiguity may occur. See Figure 5.4 below.

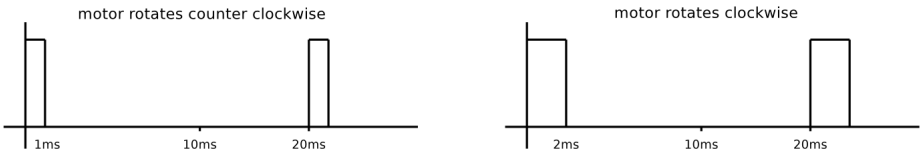

Figure 5.4: Length of pulse signals

The motor control component has been implemented in VHDL by two processes. One process to deter-

mine the next state and one to determine in which the output duration of pulse of a PWM signal, based on an input signal called 'direction'. This signal emerges from the controller component and is '0' if the wheels need to turn counterclockwise and '1' if the wheels need to turn clockwise. The motor control component will then make a pulse of 1 ms or 2 ms using the output signal of the time base as a reference. Lastly, after each period (20 ms), the motor controls are reset with a reset signal coming in from the time base. This way the robot can redo the whole process again. For a more extensive explanation of the motor control see the line follower manual [3]. If the above behaviour and description of the motor control component are implemented, one will obtain the VHDL code seen in Appendix G.3.

5.3.5. Controller

The controller is the central component of the line follower. It lets all the separate components work with each other and controls the behaviour of the line following based on its inputs. The architecture consists of two processes. The first process ensures that the controller entity works with a synchronous reset and functions on the rising edge of the clock. The second process coordinates the earlier discussed components to let the robot follow a black line. This is done by using a finite state machine which comprises six states. The reset state (state_r) resets the time base and motor direction signals and functions as a "read state" which reads the sensor signals constantly. Based on those sensor signals, the finite state machine will go to one of the other five states and stays in that state for 20 ms. These five states each create different directions signals for the motor controller components, therefore they steer the robot in different directions. The table below was used to connect the different sensor inputs to the correct direction state and thus provide the right output signals for the motor control. Afterwards, the finite state machine goes back to the reset state and repeats this process to coordinate the line following behaviour.

Table 5.5 is used to design the controller for the line follower.

Sensors			Wheels		Motor pulses		Direction
left	middle	right	left	right	left	right	
black	black	black	forward	forward	2 ms	1 ms	forward
black	black	white	stop	forward	0 ms	1 ms	gentle left
black	white	black	forward	forward	2 ms	1 ms	forward
black	white	white	reverse	forward	1 ms	1 ms	sharp left
white	black	black	forward	stop	2 ms	0 ms	gentle right
white	black	white	forward	forward	2 ms	1 ms	forward
white	white	black	forward	reverse	2 ms	2 ms	sharp right
white	white	white	forward	forward	2 ms	1 ms	forward

Figure 5.5: Relation between sensors, wheels, motor pulses and direction [3]

In Appendix G.8, the complete VHDL of the controller as described above can be found.

5.3.6. Robot top entity

This subsection concerns the integration of all the components described in the previous subsections. The robot top entity, as expected, has the sensor values as inputs and the PWM signals for the motors as output. For the integration of the components in VHDL, all the separate components need to be port mapped in the top entity file. This code can be found in Appendix G.10.

5.4. Testing phase

To check whether the line follower works as intended, a testbench for each component of the line follower was made and used to test the behaviour of the components. Using a testbench is essentially, feeding fixed values at the input of the component to be tested and then observing its outputs.

5.4.1. Time base testing

The testbench of the time base can be found in Appendix G.2. It simulates the counter with a clock period of 20 ns. From Figure G.1, it is evident that the counter counts up with one when the clock is on its rising clock edge.

5.4.2. Motor control testing
The testbench of the motor control provided by the line follower manual [3] can be found in Appendix G.4. This testbench simulates a period of 20 ms in clockwise (direction = 1) and counterclockwise direction (direction = 0)[3]. In Figure G.2 there are two noticeable peaks that transition from a low value to a high value and after some time transition back to a low value. The first peak is from 0 ms to 1 ms. During this peak the reset has the value '0' and direction '0' (counterclockwise direction). Since the motors need to turn counterclockwise, the pulse needs to last 1 ms. The second peak is from 20 ms to 22 ms. During this peak, the reset has the value '0' and direction '1' (clockwise direction). Since the motors need to turn clockwise, the pulse needs to last 2 ms. Which corresponds with what we expect and have described in section 5.3.4.

5.4.3. Input buffer testing
The testbench of the input buffer has been made that can be found in Appendix G.6. From Figure G.3, it is evident that the input signals are delayed with two clock cycles, given that no output signals have been initialized, hence the undefined output signals during the first 2 clock cycles.

5.4.4. Controller testing
The testbench of the controller has been made that can be found in Appendix G.9. From Figure G.4, it can be deduced that the motor left direction, motor right direction, motor left reset, motor right reset and the count reset has been set to the correct value with the corresponding sensor values according to Figure 5.5.

For example, when the sensor_l_in is '0', sensor_m_in is '1' and sensor_r_in is '1', which means it detects black white white, it should have a motor pulse of '1' for both left and right according to Figure 5.5. This occurs by setting the motor direction left and right to '1' as described in section 5.4.2. The motor reset signals and the count_reset signal should both be set to '0' since the motors ought not to be reset during the turn. Using the same reasoning as above it can be checked that, the output signals of the controller are indeed correct.

5.4.5. Robot top entity testing
The testbench of the robot top entity provided by the line follower manual [3] can be found in Appendix G.11. Simulating this testbench results in Figure G.5. It can be seen that the output PWM signals based on the input signals correspond with the values in Figure 5.5.

5.5. Conclusion
The line follower is of utter importance for EPO-2 because the VHDL code for the robot is its extension. The components of the line follower consist of a time base, an input buffer, motor controls, and a controller that have all been designed in VHDL code. The functionality of the line follower was checked by creating testbenches and testing whether the output signals correspond to the expected values.

6

Robot Top Entity

6.1. Introduction

As mentioned, for the EPO-2 project, the line follower assignment needs to be expanded. Besides following a black line, it needs to be able to do several more complicated tasks to complete challenges A, B, and C. This chapter will give an overview of what has been changed for the EPO-2 robot with respect to the line follower. For the most part, the components have stayed relatively the same, except for the controller. Furthermore, two more components have been added, which are the mine sensor component and the UART component. Since the controller has been changed drastically, a separate chapter (Chapter 7) has been dedicated to this component. The other components of the EPO-2 robot will be discussed in this chapter.

6.2. Requirements

In the EPO-2 manual, there were no specific requirements, but it rather gave a mission. Next to following a black line, the robot needs to be able to perform tasks based on the C program and detect mines in the maze. [1]

6.3. Design and implementation

In order to let the robot communicate with the C program and detect a mine, a mine sensor component, and a UART component need to be added to the robot entity. This led to new input and output signals going in and out of the controller which ultimately meant that the controller FSM in the line follower needed to be redesigned, as mentioned in the introduction. The EPO-2 manual proposed two different design approaches for a new (more advanced) finite state machine in the controller component. The first manner was to expand the existing controller FSM and the second opted for a multiplexer design. It was chosen to use the first design due to its simplicity in its structure and implementation. However, one large FSM would make it hard to retain a good overview of the structure. Therefore it was chosen to split the FSM into three smaller FSMs. The separation was based on the three different functionalities that needed to be implemented. Their functionalities will be discussed in the next chapter.

6.3.1. Robot top entity

Prior to addressing the new components and changes made to the different components, it is good to have an overview of the different components which the robot now consists of. In Figure 6.1 an overview of the robot is given.

The overview of the robot in Figure 6.1 shows a further developed version of the old line follower. The existing ports from the line follower from Chapter 5 have stayed the same. As for the new mine sensor and UART component new input and output ports have to be added to the controller entity. On top of that, internal adjustments have been made to the time base and motor control components. They will be discussed later on in this chapter.

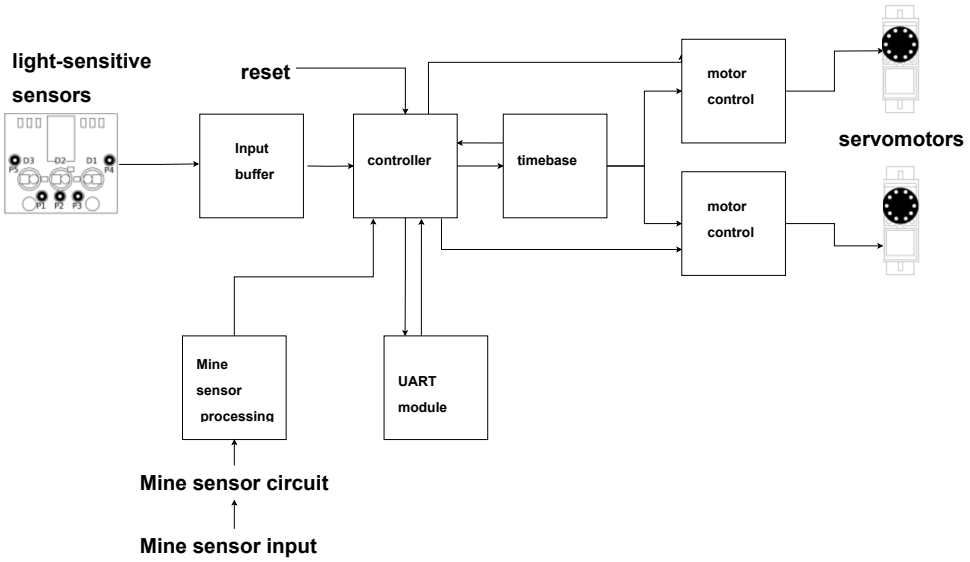

Figure 6.1: Overview of the robot top entity

The mine sensor component sends one output signal to the controller whose input is named mine_s. This input port is either zero or one. If the mine detector circuit detects a mine, then mine_s will be high. Otherwise, the signal is low. This way the robot knows whether there is a mine ahead. For more information about the mine sensor, refer back to Chapter 2.

The UART component uses RX and TX lines to read and write data to the robot. Therefore in the robot's top entity, the input port RX and output port TX are added. Furthermore, the UART sends a buffer_empty signal to the robot entity to let the robot know when the UART module is ready to accept new data for transmission. This signal has been added to the robot top entity named, buffer_empty. The code for the UART component was given, and simply portmapped in the top entity. For more information about the UART module, refer back to Chapter 3.

The full robot top entity VHDL code can be found in Appendix H.3.3.

6.3.2. Adjustments made to the time base

In the time base an extra output signal, reset_out, has been added. This signal will go high when either the global reset is high or when the time base counter has counted 20 milliseconds. Otherwise, the signal is low. The purpose of this extra signal is to use it as a periodic reset for the servo motors as they needed to be reset every 20 milliseconds. This implementation will be discussed in the next section. The code below shows what adjustment has been made. Only the part that differs from the time base in the line follower is shown. For the complete adjusted time base, see Appendix H.1.

```
process ( clk )
   begin
       if ( clk'event and clk ='1' ) then
                 if ( (reset = '1') or (unsigned(count) >= to_unsigned(1000000, 20))) then
                       count <= ( others => '0');
                       reset_out <= '1';                -- Added reset_out port.
       else
                               reset_out <= '0';      -- Added reset_out port.
                       count <= new_count;
               end if;
       end if;
end process;
```

6.3.3. Adjustments made to the motor control

The implementation of the periodic reset of the servo motors is done in the adjusted motor controller. For this periodic reset to work, the motor control uses the reset_out signal from the time base as a condition to

go to the reset state (state0). So when reset_out signal is high, the PWM signal will go low. Hence the servo motors will be reset every 20 milliseconds. The code below shows how this is done in VHDL. Only the part that differs from the motor control in the line follower is shown. For the complete adjusted motor control, see figure H.2.

```
process (clk)
        begin
                if (rising_edge (clk)) then
                        if (reset = '1' OR reset_in = '1') then --added port reset_in
                                state <= state0;
                        else
                                state <= new_state;
                        end if;
                end if;
        end process;
```

6.4. Testing

Along with the extra components and newly implemented time base, motor control, and drastically changed controller, the robot has been tested. Since the controller is not yet elaborated upon, the testing of the robot will be explained at the end of Chapter 7. Understanding the test results requires an understanding of the biggest component first.

6.5. Conclusion

When expanding the line follower, new components were added and existing ones were adjusted. The mine sensor and UART component were added, to enable the robot to sense mines and communicate with the C program. However, the biggest and most important component that was adjusted, the controller, will be discussed next chapter. In the next chapter, the testing results of the robot will also be found.

7

The controller component of the robot

7.1. Introduction

For a clearer overview of the controller, a separate chapter was created. The corresponding code can be found in Appendix H.3.2. The controller is as seen in Chapter 5, the decision maker of the robot using the input signals. In Chapter 5, these input signals were only light-sensitive sensor signals. However, in the EPO2 robot, the additional input signals are the mine and communication signals. Furthermore, additional output signals are also added to communicate with the C program. This chapter will elaborate on how the controller deals with these extra signals and how it is designed. Last but not least, this chapter will cover the testing of the robot with its new and adjusted components.

The controller consists of four processes, which all have their own purpose. The processes are the clock process, the cross-counter process, the communication process and the motor control process. Each of the processes will be explained in the following subsections.

7.2. Clock process

The clock process ensures that all the events are synchronized with the rising edge of the clock. This means that all the FSMs will transition to the next state when the clock goes high. Furthermore, it arranges a synchronous reset which enforces every FSM to the reset state at the first rising clock edge after the reset signal was high.

7.3. Cross-counter process

As shown in Figure 1.1, the challenge grid does not simply consist of straight black lines and definite crossings. In the middlepoint of all the edges, a small black line can be observed that will be referred to as a patch. Since the robot needs to make decisions at an intersection, hence when the sensor_l_in is '0', sensor_m_in is '0' and sensor_r_in is '0', which means it detects black black black, a patch might falsely trigger a decision and thus make decisions at the wrong times. This means that, if the robot can't differentiate the patches from the crossings, it might e.g. turn on a patch, thus leading it to fail the challenge.
The cross-counter FSM is based on the position of the robot on the map and in which direction it's heading. There are four states: state_ptc (patch to crossing), state_crossing, state_ptc (patch to crossing), and state_patch.

The principle is that the robot cycles between these four states and that the state in which this FSM is, should represent where the robot is. When the robot finds itself above a crossing, a signal called 'crossing' is pulled high in order to let the communication FSM know that it can communicate to the C program, so the robot knows what to do at the right time. Below an explanation of the states can be found.
The states are listed below in order of occurrence.

- state_ptc: This state resembles the situation where the next time the robot detects black-black-black, which means that it is at a crossing. It is also the state that the robot returns to after a reset. At the start of a challenge, the robot starts at a station and therefore it first encounters a crossing.

- state_crossing: When the finite state machine is in this state, the robot is actively above a crossing and the crossing signal is set high.
- state_ctp: As soon as the robot continues its trajectory and leaves the crossing (being detected as white-black-white) the finite state machine transfers to this state and pulls the crossing signal low.
- state_patch: The next time black-black-black is detected, the robot is above a patch and therefore the crossing signal should stay low.

A more in-depth schematic of this FSM can be found in Appendix H.1. Additionally, the full cross-counter process can be found in lines 89-126 of Appendix H.3.2.

7.4. Communication process

The functionality of the communication process is to send and read the signals between VHDL and the C application via UART protocol. This process consists of four states:

- state_com_r: reset state of the communication process
- state_com_write: the state where data will be written by VHDL to the C application.
- state_com_wait: the state where the robot waits until the data is sent.
- state_com_read: the state where data will be read by the robot.

The robot needs to send data to C on three different occasions. When the robot detects the mine, when the robot is at a crossing and when the robot is at a station (dead end). State_com_r checks when those events occur using mine_s, crossing and sensors_l/m/r signals. At state_com_write the robot sends specific data based on those events. Table 7.1 provides a clear overview. Next, the FSM will reach the wait states. In these states, named state_com_wait_1 and state_com_wait_2, the write is pulled down immediately and waits for data coming from C. If data_ready is high then data is ready to be read from the Xbee. This is followed by the state_com_read state which reads the data_in vector and sets specific signals high based on the data from C. These signals are used to create a linkage between the communication process and the control motor process in order to let the robot react to the input data. Moreover will be explained in the next section. To ensure a well-functioning communication process, the state_com_read state also checks for the occasions when it doesn't need to write data before going to the state_com_r again. Afterwards, the process can repeat itself again.

Table 7.1: Data sent from the robot to the PC

Bit value	Description
00000010	The robot has encountered a mine.
00000011	The robot has encountered an intersection.
00000100	The robot has encountered a dead end.

7.5. Control motors process

The control motors process is to control the motors for example when the robot has received data from the PC. The whole control motors process can be divided into five main branches: forward, left, right, U-turn and stop branches. With these divisions of the branches, the robot can go towards any direction and stop when needed. The control motors process has been written in lines 243-481 in Appendix H.3.2. In this section, all the different branches of this process are explained.

7.5.1. Reset state

The reset state is the state where the robot starts and is purely utilized as a 'start state'. This means that the robot will not be able to go back to this state, except at the start. This state is trivial because the robot will not ride immediately when he is turned on and solely starts riding when he is on the line. If the robot is on the line, and thus the sensor_l , sensor_m and sensor_r are '101', white black white respectively, the robot will go to the next state the state_f which will be described in section 7.5.2.

7.5.2. Forward branch

The VHDL code for the forward branch consists only of state_f, but it can be considered as the core branch of all the branches and thus also the core for the VHDL code for the motor control itself. The reason for this consideration is that the forward branch is connected to all the branches in the motor control and eventually, every branch returns to the forward branch except the stop branch. The exclusion of the stop branch will be explained in section 7.5.6.

The connections of the states with state_f can be divided into 2 groups. The first group is the line follower, consisting of state_gl, state_sl, state_gr and state_sr. The operation of the line follower has been explained in section 5.3.

The second group is the extension of line follower in the form of branches where this FSM goes to when data signals from the communication process come in. In other words, the robot executes a behaviour base on signals that come from the communication process. This second group of the extension of the line follower consists of state_gl_d, state_sl_d_2, state_gr_d, state_sr_d_2, state_s, state_u_turn and state_u_turn_2. Each branch can be reached by the matching signal that is sent via the communication process described above, for instance when left_signal is '1', the left branch will be reached. Only the U-turn branch can be reached by two signals, U-turn_signal or mine_s, the latter being a signal from the mine sensors (section 2.4.1).

7.5.3. Left branch

The left branch is only reached when the incoming left_signal from the communication process is high and consists of two states, namely consists of state_gl_d and state_sl_d_2 and is implemented as follows. The robot starts at state_gl_d and turns gently left until the sensors detect white white white, meaning that the sensor_l and sensor_m and sensor_r are '111' respectively. The robot is now away from the line it was on and needs to keep turning until he sees the line on his left that's orthogonal to the original line. Thus after detecting '111' with the light sensors, the robot goes to the next state state_sl_d_2, where it turns sharp left till the sensors detect white black white, meaning that the sensor_l and sensor_m and sensor_r are '101' respectively. If the sensors detect '101' the robot goes to state_f and continues on with the maze. The VHDL code of the left branch can be found in the lines 393-416 of appendix H.3.2.

7.5.4. Right branch

The right branch is only reached when the incoming right_signal from the communication process is high and consists of two states, namely consists of state_gr_d and state_sr_d_2 and is implemented as follows. The robot starts at state_gr_d and turns gently right until the sensors detect white white white, meaning that the sensor_l and sensor_m and sensor_r are '111' respectively. The robot is now away from the line it was on and needs to keep turning until it sees the line on its right that is orthogonal to the original line. Thus after detecting '111' with the light sensors, the robot goes to the next state state_sr_d_2, where it turns sharp right till the sensors detect white black white, meaning that the sensor_l and sensor_m and sensor_r are '101' respectively. If the sensors detect '101' the robot goes to state_f and continues on with the maze. The VHDL code of the right branch can be found in the lines 420-447 of appendix H.3.2.

7.5.5. U-turn branch

The robot needs to make a turn when it is at a dead end or when it has detected a mine. The U-turn is made by turning the robot sharp right until it detects the line again. The U-turn consists of two states, the state_u_turn and state_u_turn_2, described in lines 452-477 in Appendix H.3.2.

When one of the two conditions to make a U-turn is met, u_turn_signal gets value '1' assigned and the robot goes from state_f to state_u_turn. In this state, the robot makes sharp right turns until the light-sensitive sensors detect '111' and as a result, the robot goes to state_u_turn_2. In this state, the robot keeps making a sharp right turn until the light-sensitive sensors detect '101', white black white. This is when the robot detects the line again and thus will go back to state_f. However, it also stops turning when the light-sensitive sensors detect '000' because when the robot senses black-black-black, it means it's either on a crossing or patch and thus should go forward and follow the line. The VHDL code for the U-turn can be found in lines 452-478 of Appendix H.3.2.

7.5.6. Stop branch

The robot will only stop and be in the stop branch when the stop_signal coming from the communication process is high. To stop the robot from moving the motor controls are constantly fed high reset signals. Thus the count_reset, motor_l_reset and the motor_r_reset have the output value '1' in this state. The value of the motor_l_direction and the motor_r_direction should not matter, because both motors were reset by the motor_l_reset and the motor_r_reset. The VHDL code for the stop branch can be found in lines 379-386 of Appendix H.3.2.

7.6. Testing

The controller has been tested on functionality. This is done by using LEDs on the FPGA and fixing (brute forcing) a specific opcode at the data_in input vector.

First of all, the behavior of the robot has been tested by fixing data_in to a specific opcode. For example, by fixing the data_in to the value '000000001' in the robot top entity. then the robot is expected to turn left at all times. This has been done with all the opcodes to check whether the robot will behave properly. Below can be found the manner in which data_in value was fixed.

```
LB2: controller port map(
...
-- changed code to test the behaviour of the robot
                                data_in =>      "000000001"

--              data_in         => data_uoci,
...
)
```

Furthermore, LEDs have been used to show how the state transitions proceeded within the controller. With this information, it could be observed whether the robot was going to the correct state or whether it was going to the wrong state. Aside from testing correct state assignments, signals were also checked with LEDs. To check the states with LEDs by assigning different bits of the data_out value to different LEDs on the FPGA. Therefore, the LEDs need to be added as input ports in the robot's top entity. Below can be found the manner in which the output of the top entity is connected to the LEDs on the FPGA.

```
entity robot is
        port (
 ...
            led_7: out std_logic;
            led_6: out std_logic;
            led_5: out std_logic;
            led_4: out std_logic;
            led_3: out std_logic;
            led_2: out std_logic;
            led_1: out std_logic;
            led_0: out std_logic
...
);
```

In the port map of the controller, the bytes of the data_out bit vector have been assigned to different LEDs. The following port map of the controller has been used in the robot top entity to test. The rest of the port map of the controller has stayed the same. These two changes in the top entity of the VHDL code will make the LEDs on the FPGA light up according to the data_out signal. This makes it clear whether the robot is in the expected state when forced into a specific data_in signal.

```
LB2: controller port map(
...
-- changed code to test with the LED
                                data_out(7)     =>      led_7,
                                data_out(6)     =>      led_6,
                                data_out(5)     =>      led_5,
                                data_out(4)     =>      led_4,
                                data_out(3)     =>      led_3,
                                data_out(2)     =>      led_2,
                                data_out(1)     =>      led_1,
                                data_out(0)     =>      led_0,

--              data_out        => data_uico,
...
)
```

When doing such actions on the robot, the robot has been observed to display the correct outputs given the inputs.

7.7. Conclusion

The above-explained implementation of the controller module was tested for correct operation firstly by putting the internal preexisting signals and signals dedicated to distinguishing the states of the FSMs on LEDs, and then by tests described in Chapter 8. Everything worked as intended.

This design focuses on simplicity and therefore is easy to modify and highly modular, however, it offloads the computational load to the C program, therefore, requiring higher complexity of it.

8

Test results of the robot's performance in the challenges

After constructing all the different components of the robot. It was time to test the robot's performance as a whole and whenever the robot is able to complete the 3 different challenges.

For challenge A, the robot has to visit a certain sequence of stations in a given order. To carry out the challenge, the C application sends orders to the robot via UART.

Unfortunately, communication with the robot did not function well. One of the major problems was that the C application, received too many data packets from the robot, instead of just one at a time. A 'workaround' was implemented for that, which was ignoring the excess packets. However, that gave rise to a new problem. Once the robot had made a U-turn, the C application lost communication and control over the robot.

As stated before in Chapter 4, the C code had been tested and it worked as intended, it was also tested by connecting two Xbees on a pc and pairing the Xbees together. The C program sent its output through one Xbee and the other Xbee would receive it. By reading the receiving Xbee using a serial monitor it was concluded the sent packets from the C program were as they were intended.
Also, the VHDL robot was tested using Xbee modules. Instead of having the C program send data packets to the robot. Robot instructions were manually sent via a the XCTU program to the robot. By having the instructions manually sent using the Xbees, the robot controller performed the instructions as intended. From this, it could be concluded the VHDL code of the controller works fine and that most likely something with the UART connection between the C Program and robot went wrong.
A lot of time went into debugging the problems of the robot, ultimately the U-turn problem after doing a right or left turn was not able to be resolved. This led to the realization that not only challenge A but also challenges B and C could not be executed by group A1_1. As an alternative, the group showed that the robot could at least follow the line and communicate with the C application if it did not have to make a U-turn.

9

Conclusion

The question of whether the project was successful depends truly on the perspective of the judgment. During the symposium, the robot did not complete any of the challenges, however, the project was an educational assignment and we learned a lot through it, therefore it can be called a success from this perspective.

We know what steps to take to debug and fix the issues that stopped us from succeeding, however, we ran out of time to proceed with it.

It might be doubted whether the time problem was actually a problem of the skill, however the huge delay in the troubleshooting and integrating process was caused by the VHDL group and they were willing to correct it with hard work and determination, by working during multiple weekends and afternoons. In the end, they delivered a mostly working code and they learned a lot in the process. The final issue lies in UART communication.

This experience though us how to mitigate such management issues, so they do not have to be corrected with hard work in the future. We can apply this knowledge in any future project.

The most crucial part is to keep a perfect information flow between subgroups. This not only prevents miscommunications regarding the design decisions, but also ensures that all the group members know the state of the progress of other people and help them / push them to work hard, or faster when necessary.

To summarize, this report describes how the robot was designed and thought out, and even though it was not an engineering success it was truly an educational success. We not only gained a lot of hard skills but also soft skills.

A

C code

A.1. Algorithm-related files

What follows is the contents of the file located at `control/algorithm/board.h`.

```
1  //
2  // Created by Thijs van Esch on 06/06/2023.
3  //
4
5  #ifndef EPO2_CONTROL_ALG_BOARD_H
6  #define EPO2_CONTROL_ALG_BOARD_H
7
8  #include "location.h"
9
10 typedef int** board;
11
12 //
13 // Returns a newly created board with all default values.
14 //
15 board new_board();
16
17 //
18 // Returns one of the four different directions the robot can move to from its
19 // current_loc location.
20 //
21 // Note that this function may return a location to which the robot can not
22 // move.
23 //
24 location get_direction(int index, location current);
25
26 //
27 // Returns the value of the given cell on the given board.
28 //
29 int read_cell(board b, location loc);
30
31 //
32 // Writes the given value to the given location at the given board.
33 //
34 // The given location is assumed to be valid.
35 //
36 // This function can be used to write the locations of mines on to the board.
37 // If this is done, the number -3 should be given for `value`.
38 //
39 void write_cell(board b, location loc, int value);
40
41 //
42 // Returns a boolean indicating whether the given location is valid.
43 //
44 int is_valid_loc(location loc);
45
```

```
46 //
47 // Calculates the shortest path between the start location and the destination.
48 // This path is written to the gl_board.
49 //
50 void calc_path(board b, location start, location end);
51
52 //
53 // Prints the given board to the console.
54 //
55 void print_board(board b);
56
57 #endif //EPO2_CONTROL_ALG_BOARD_H
```

What follows is the contents of the file located at `control/algorithm/board.c`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
3 //
4
5 #include "board.h"
6 #include "default.h"
7 #include "queue.h"
8
9 #include <stdio.h>
10 #include <stdlib.h>
11
12 //
13 // The status code with which the program should exit if an illicit value is
14 // encountered by the algorithm functions and the error is fatal.
15 //
16 #define ALGORITHM_ERROR 6
17
18 board new_board()
19 {
20   int **board = (int **) malloc(sizeof(int *) * 13);
21   for (int i = 0; i < 13; ++i)
22   {
23     board[i] = (int *) malloc(sizeof(int) * 13);
24     for (int j = 0; j < 13; ++j)
25     {
26       board[i][j] = DEFAULT_BOARD[i][j];
27     }
28   }
29   return board;
30 }
31
32 location get_direction(int index, location current)
33 {
34   const static int X_DIFF[] = {-1, 0, 1, 0};
35   const static int Y_DIFF[] = {0, 1, 0, -1};
36   location loc = {current.x + X_DIFF[index], current.y + Y_DIFF[index]};
37   return loc;
38 }
39
40 int is_valid_loc(location loc)
41 {
42   int valid_x = loc.x >= 0 && loc.x < 13;
43   int valid_y = loc.y >= 0 && loc.y < 13;
44   return valid_x && valid_y;
45 }
46
47 int read_cell(board b, location loc)
48 {
49   if (!is_valid_loc(loc))
50   {
51     printf(" ::\tattempted to read board value from illicit "
52            "location, causing a segfault\n");
53     exit(ALGORITHM_ERROR);
54   }
55   return b[loc.x][loc.y];
```

```
56 }
57
58 int should_move_to(board b, location loc)
59 {
60   if (!is_valid_loc(loc))
61   {
62     return 0;
63   }
64   int value = read_cell(b, loc);
65   return value == 0;
66 }
67
68 void write_cell(board b, location loc, int value)
69 {
70   if (!is_valid_loc(loc))
71   {
72     printf(" ::\tattempted to write board value to nonexistent "
73            "cell, causing a segfault\n");
74     exit(ALGORITHM_ERROR);
75   }
76   b[loc.x][loc.y] = value;
77 }
78
79 void reset_cell(board b, location loc)
80 {
81   int value = read_cell(b, loc);
82   if (value >= 0)
83   {
84     write_cell(b, loc, 0);
85   }
86 }
87
88 void reset_board(board b)
89 {
90   for (int x = 0; x < 13; ++x)
91   {
92     for (int y = 0; y < 13; ++y)
93     {
94       location loc = {x, y};
95       reset_cell(b, loc);
96     }
97   }
98 }
99
100 void calc_path(board b, location start, location end)
101 {
102   reset_board(b);
103   queue *q = create_queue();
104   append_queue(q, end, 1);
105   write_cell(b, end, 1);
106   int distance = -1;
107   while (queue_length(q))
108   {
109     location current_loc;
110     int current_dist;
111     from_queue(q, &current_loc, &current_dist);
112     if (equal_loc(start, current_loc) && distance == -1)
113     {
114       distance = current_dist;
115     }
116     for (int i = 0; i < 4; ++i)
117     {
118       location new_loc = get_direction(i, current_loc);
119       if (!should_move_to(b, new_loc))
120       {
121         continue;
122       }
123       write_cell(b, new_loc, current_dist + 1);
124       append_queue(q, new_loc, current_dist + 1);
125     }
126   }
```

```
127 }
128
129 void print_padded(int value, int print_zero)
130 {
131   if (value == 0 && !print_zero)
132   {
133     printf("   . ");
134     return;
135   } else if (value == -1 && !print_zero)
136   {
137     printf("     ");
138     return;
139   }
140   char s[6];
141   sprintf(s, "%d", value);
142   printf("%5s", s);
143 }
144
145 void print_header()
146 {
147   printf("\n row / col");
148   for (int i = 1; i < 13 + 1; ++i)
149   {
150     print_padded(i - 1, 13);
151   }
152   printf("\n");
153   for (int i = 0; i < 79; ++i)
154   {
155     printf("-");
156   }
157   printf("\n");
158 }
159
160 void print_board(board b)
161 {
162   print_header();
163   for (int i = 0; i < 13; ++i)
164   {
165     print_padded(i, 1);
166     printf("   |   ");
167     for (int j = 0; j < 13; ++j)
168     {
169       print_padded(b[i][j], 0);
170     }
171     printf("\n");
172   }
173 }
```

What follows is the contents of the file located at `control/algorithm/default.h`.

```
1  //
2  // Created by Thijs van Esch on 06/06/2023.
3  //
4
5  #ifndef EPO2_CONTROL_ALG_DEFAULT_H
6  #define EPO2_CONTROL_ALG_DEFAULT_H
7
8  //
9  // This 2D array holds the initial values for all cells:
10 //
11 const int DEFAULT_BOARD[13][13] = {
12   {-1, -1, -1, -1,  0, -1,  0, -1,  0, -1, -1, -1, -1},
13   {-1, -1, -1, -1,  0, -1,  0, -1,  0, -1, -1, -1, -1},
14   {-1, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1, -1},
15   {-1, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1, -1},
16   { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0},
17   {-1, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1, -1},
18   { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0},
19   {-1, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1, -1},
20   { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0},
```

```
21   {-1, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1, -1},
22   {-1, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1, -1},
23   {-1, -1, -1, -1,  0, -1,  0, -1,  0, -1, -1, -1, -1},
24   {-1, -1, -1, -1,  0, -1,  0, -1,  0, -1, -1, -1, -1}
25 };
26
27 #endif //EPO2_CONTROL_ALG_DEFAULT_H
```

What follows is the contents of the file located at `control/algorithm/location.h`.

```
1  //
2  // Created by Thijs van Esch on 06/06/2023.
3  //
4
5  #ifndef EPO2_CONTROL_ALG_LOCATION_H
6  #define EPO2_CONTROL_ALG_LOCATION_H
7
8  struct location
9  {
10   int x;
11   int y;
12 };
13
14 typedef struct location location;
15
16 //
17 // Returns the location of the given station.
18 //
19 // The given integer should be the identifier of the station. This value should
20 // be in the range [1, 12].
21 //
22 location get_station_loc(int station);
23
24 //
25 // Returns a boolean indicating whether the two given locations are equal.
26 //
27 int equal_loc(location loc0, location loc1);
28
29 #endif //EPO2_CONTROL_ALG_LOCATION_H
```

What follows is the contents of the file located at `control/algorithm/location.c`.

```
1  //
2  // Created by Thijs van Esch on 06/06/2023.
3  //
4
5  #include "location.h"
6
7  #include <stdio.h>
8
9  const int STATION_ROWS[] =
10   {12, 12, 12, 8, 6, 4, 0, 0, 0, 4, 6, 8};
11 const int STATION_COLS[] =
12   {4, 6, 8, 12, 12, 12, 8, 6, 4, 0, 0, 0};
13
14 location get_station_loc(int station)
15 {
16   location loc = {STATION_ROWS[station - 1], STATION_COLS[station - 1]};
17   return loc;
18 }
19
20 int equal_loc(location loc0, location loc1)
21 {
22   return loc0.x == loc1.x && loc0.y == loc1.y;
23 }
```

What follows is the contents of the file located at `control/algorithm/queue.h`.

```c
//
// Created by Thijs van Esch on 06/06/2023.
//

#ifndef EPO2_CONTROL_ALG_QUEUE_H
#define EPO2_CONTROL_ALG_QUEUE_H

#include "location.h"

struct queue_node;

typedef struct queue_node queue_node;

struct queue_node
{
  location loc;
  int dist;
  queue_node *next;
};

struct queue
{
  queue_node *first;
  queue_node *last;
};

typedef struct queue queue;

//
// Creates an empty queue and returns its address.
//
queue *create_queue();

//
// Returns the length of the given queue.
//
int queue_length(const queue *q);

//
// Appends the given data to the given queue.
//
void append_queue(queue *q, location loc, int dist);

//
// Removes the first node from the queue. Its data is written to the addresses
// given to the function.
//
void from_queue(queue *q, location *loc, int *dist);

//
// Prints each value in the given queue.
//
void print_queue(const queue *q);

#endif //EPO2_CONTROL_ALG_QUEUE_H
```

What follows is the contents of the file located at `control/algorithm/queue.c`.

```c
//
// Created by Thijs van Esch on 06/06/2023.
//

#include <stdio.h>
#include <stdlib.h>

#include "queue.h"

queue *create_queue()
{
  queue *q = (queue *) malloc(sizeof(queue));
```

```
13   q->first = NULL;
14   q->last = NULL;
15   return q;
16 }
17
18 int queue_length(const queue *q)
19 {
20   queue_node *cursor = q->first;
21   int length = 0;
22   while (cursor != NULL)
23   {
24     length += 1;
25     cursor = cursor->next;
26   }
27   return length;
28 }
29
30 void append_queue(queue *q, location loc, int dist)
31 {
32   queue_node *new_node = (queue_node *) malloc(sizeof(queue_node));
33   new_node->next = NULL;
34   new_node->loc = loc;
35   new_node->dist = dist;
36   int length = queue_length(q);
37   if (length == 0)
38   {
39     q->first = new_node;
40     q->last = new_node;
41   } else
42   {
43     q->last->next = new_node;
44     q->last = new_node;
45   }
46 }
47
48 void from_queue(queue *q, location *loc, int *dist)
49 {
50   if (!queue_length(q))
51   {
52     return;
53   }
54   queue_node *first = q->first;
55   q->first = q->first->next;
56   *loc = first->loc;
57   *dist = first->dist;
58   free(first);
59 }
60
61 void print_queue(const queue *q)
62 {
63   queue_node *cursor = q->first;
64   int index = 0;
65   while (cursor != NULL)
66   {
67     printf("%d\t%d\t%d\t%d\n",
68       index,
69       cursor->loc.x,
70       cursor->loc.y,
71       cursor->dist);
72     index += 1;
73     cursor = cursor->next;
74   }
75 }
```

A.2. Control-related files

What follows is the contents of the file located at `control/comm.h`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
```

```
3 //
4
5 #ifndef EPO2_CONTROL_COMM_H
6 #define EPO2_CONTROL_COMM_H
7
8 #include <stdint.h>
9
10 enum state_update
11 {
12   INITIALIZED = 'b',
13   AT_INTERSECTION = 'i',
14   AT_DEAD_END = 'd',
15   AT_MINE = 'm',
16   DONE_TURNING = 'u'
17 };
18
19 typedef enum state_update state_update;
20
21 //
22 // Returns a boolean indicating whether the given byte represents a possible
23 // state update or not.
24 //
25 int is_state_update(uint8_t incoming);
26
27 //
28 // Returns the state update corresponding to the given data.
29 //
30 state_update parse_incoming(uint8_t incoming);
31
32 #endif //EPO2_CONTROL_COMM_H
```

What follows is the contents of the file located at `control/comm.c`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
3 //
4
5 #include "comm.h"
6
7 int is_state_update(uint8_t incoming)
8 {
9   return incoming >= 0b001 && incoming <= 0b101;
10 }
11
12 state_update parse_incoming(uint8_t incoming)
13 {
14   switch (incoming)
15   {
16     case 0b001:
17       return INITIALIZED;
18     case 0b010:
19       return AT_INTERSECTION;
20     case 0b011:
21       return AT_DEAD_END;
22     case 0b100:
23       return AT_MINE;
24     default:
25       return DONE_TURNING;
26   }
27 }
```

What follows is the contents of the file located at `control/control.h`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
3 //
4
5 #ifndef EPO2_CONTROL_CONTROL_H
6 #define EPO2_CONTROL_CONTROL_H
```

```
7
8  #include <stdint.h>
9
10 //
11 // Initializes the robot control.
12 //
13 // The given integer should be the letter of the a. If an illicit
14 // character is given, an error will be printed to the console and the function
15 // will return a non-zero integer.
16 //
17 // All data that is required for the a will be written from standard
18 // input.
19 //
20 int init_control(char a, int start_station);
21
22 //
23 // Handles the given byte as one that was retrieved from the robot.
24 //
25 void handle_incoming(uint8_t data);
26
27 //
28 // Determines if anything should be sent to the robot.
29 //
30 // If data is to be sent to the robot, this data will be written to the given
31 // address and the function will return 1. If nothing is to be sent to the
32 // robot, nothing will be written to the given address and the function will
33 // return with 0.
34 //
35 int get_outgoing(uint8_t *data);
36
37 //
38 // Returns a boolean indicating whether the assignment has been executed.
39 //
40 int assignment_done();
41
42 #endif //EPO2_CONTROL_CONTROL_H
```

What follows is the contents of the file located at `control/control.c`.

```
1  //
2  // Created by Thijs van Esch on 06/06/2023.
3  //
4
5  #include <stdio.h>
6
7  #include "algorithm/board.h"
8  #include "comm.h"
9  #include "control.h"
10 #include "navigation.h"
11 #include "targets.h"
12
13 char assignment = '\0';
14
15 volatile board gl_board = NULL;
16 location current_loc = { -1, -1 };
17 orientation current_or = NORTH;
18
19 int init_control(char a, int start_s)
20 {
21   assignment = a;
22   gl_board = new_board();
23   print_board(gl_board);
24   current_loc = get_station_loc(start_s);
25   current_or = get_station_or(start_s);
26   calc_path(gl_board, current_loc, cur_target_loc());
27   printf("initialized robot control for assignment %c\n", assignment);
28   return 0;
29 }
30
31 int outgoing_available = 0;
```

```
32 uint8_t outgoing_data = 0;
33
34 void set_outgoing(nav_instruction instruction)
35 {
36   if (outgoing_available)
37   {
38     printf(" !!\twriting outgoing even though buffer not empty\n");
39   }
40   outgoing_available = 1;
41   outgoing_data = to_opcode(instruction);
42   printf("> sending op '%c' (0x%02x)\n", instruction, outgoing_data);
43 }
44
45 void flip_orientation()
46 {
47   orientation flipped = after_rot(after_rot(current_or, LEFT), LEFT);
48   current_or = flipped;
49 }
50
51 void log_robot_state(state_update robot_state)
52 {
53   switch (robot_state)
54   {
55     case INITIALIZED:
56       printf("> robot initialized\n");
57       break;
58     case AT_INTERSECTION:
59       printf("> at intersection\n");
60       break;
61     case AT_DEAD_END:
62       printf("> encountered dead end\n");
63       break;
64     case AT_MINE:
65       printf("> encountered mine\n");
66       break;
67     default: // case DONE_TURNING:
68       printf("> robot done turning\n");
69       break;
70   }
71 }
72
73 int assignment_done()
74 {
75   return targets_left() == 0;
76 }
77
78 //
79 // This function should be called every time an intersection is encountered.
80 //
81 // It updates the robots location and writes the next robot operation to the
82 // output buffer.
83 //
84 void update_location()
85 {
86   if (current_loc.x < 0 || current_loc.y < 0)
87   {
88     current_loc = get_station_loc(1);
89     current_or = NORTH;
90   } else
91   {
92     current_loc = inc_loc(current_loc, current_or, 2);
93   }
94   calc_path(gl_board, current_loc, cur_target_loc());
95   nav_instruction next;
96   next = get_instruction(gl_board, current_loc, current_or);
97   current_or = after_rot(current_or, next);
98   set_outgoing(next);
99 }
100
101 //
102 // This function should be called every time a mine is encountered.
```

```
103 //
104 // It writes the location of the mine to the board. It does not set any
105 // outgoing data.
106 //
107 void encountered_mine()
108 {
109   location mine = inc_loc(current_loc, current_or, 1);
110   write_cell(gl_board, mine, -3);
111 }
112
113 void handle_incoming(uint8_t data)
114 {
115   if (!is_state_update(data))
116   {
117     if (data)
118     {
119       printf("illicit data from robot: 0x%02x\n", data);
120     }
121     return;
122   }
123   state_update robot_state = parse_incoming(data);
124   log_robot_state(robot_state);
125   switch (robot_state)
126   {
127     case AT_INTERSECTION:
128       update_location();
129       break;
130     case AT_MINE:
131       encountered_mine();
132     case AT_DEAD_END:
133       set_outgoing(U_TURN);
134       inc_target();
135       break;
136     case DONE_TURNING:
137       flip_orientation();
138     default:
139       break;
140   }
141 }
142
143 int get_outgoing(uint8_t* data)
144 {
145   if (outgoing_available)
146   {
147     *data = outgoing_data;
148     outgoing_available = 0;
149     return 1;
150   } else
151   {
152     return 0;
153   }
154 }
```

What follows is the contents of the file located at `control/navigation.h`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
3 //
4
5 #ifndef EPO2_CONTROL_NAVIGATION_H
6 #define EPO2_CONTROL_NAVIGATION_H
7
8 #include <stdint.h>
9
10 #include "algorithm/board.h"
11
12 enum orientation
13 {
14   NORTH = 6,
15   EAST = 7,
```

```
16   SOUTH = 8,
17   WEST = 9
18 };
19
20 typedef enum orientation orientation;
21
22 //
23 // Returns the direction the robot will initially be facing if starting at
24 // the given station.
25 //
26 orientation get_station_or(int station);
27
28 //
29 // Increments a location.
30 //
31 // Returns the location that is COUNT cells removed from LOC in the direction
32 // of ORIENTATION_.
33 //
34 // TODO: provide example for function call
35 //
36 location inc_loc(location loc, orientation orientation_, int count);
37
38 enum nav_instruction
39 {
40   FORWARD = 'f',
41   LEFT = 'l',
42   RIGHT = 'r',
43   U_TURN = 'u',
44   STOP = 's'
45 };
46
47 typedef enum nav_instruction nav_instruction;
48
49 //
50 // Returns the direction the robot will be facing after executing the given
51 // instruction when currently facing the given direction.
52 //
53 orientation after_rot(orientation current, nav_instruction instruction);
54
55 //
56 //  Returns the robot opcode for the given navigation instruction.
57 //
58 uint8_t to_opcode(nav_instruction instruction);
59
60 //
61 // Determines the next navigation instruction for the robot, assuming that
62 // it is currently located at the given location and facing the given direction.
63 //
64 // The location is assumed to be one of an intersection.
65 //
66 nav_instruction get_instruction(board b, location loc, orientation o);
67
68 #endif //EPO2_CONTROL_NAVIGATION_H
```

What follows is the contents of the file located at `control/navigation.c`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
3 //
4
5 #include <stdio.h>
6
7 #include "navigation.h"
8
9 orientation get_station_or(int station)
10 {
11   station = (station - 1) / 3;
12   if (station == 0)
13   {
14     return NORTH;
```

```
15   } else if (station == 1)
16   {
17     return WEST;
18   } else if (station == 2)
19   {
20     return SOUTH;
21   } else
22   {
23     return EAST;
24   }
25 }
26
27 location inc_loc(location loc, orientation orientation_, int count)
28 {
29   int x = loc.x, y = loc.y;
30   switch (orientation_)
31   {
32     case NORTH:
33       x -= count;
34       break;
35     case EAST:
36       y += count;
37       break;
38     case SOUTH:
39       x += count;
40       break;
41     default: // case WEST:
42       y -= count;
43       break;
44   }
45   location incremented = { x, y };
46   return incremented;
47 }
48
49 orientation after_rot(orientation current, nav_instruction instruction)
50 {
51   int diff = 0;
52   if (instruction == LEFT)
53   {
54     diff = -1;
55   } else if (instruction == RIGHT)
56   {
57     diff = 1;
58   } else if (instruction == U_TURN)
59   {
60     diff = -2;
61   }
62   orientation new = current + diff;
63   return new < NORTH ? new + 4 : new;
64 }
65
66 uint8_t to_opcode(nav_instruction instruction)
67 {
68   switch (instruction)
69   {
70     case LEFT:
71       return 1;
72     case RIGHT:
73       return 2;
74     case FORWARD:
75       return 3;
76     case STOP:
77       return 4;
78     default: // U_TURN
79       return 5;
80   }
81 }
82
83 //
84 // Returns the cell where the robot should move to next.
85 //
```

```
86 // This cell always lies next to the cell where the robot is currently located.
87 //
88 location next_loc(board b, location current)
89 {
90   int smallest_value = read_cell(b, current);
91   int movement = -1;
92   for (int i = 0; i < 4; ++i)
93   {
94     location possibility = get_direction(i, current);
95     if (!is_valid_loc(possibility))
96     {
97       continue;
98     }
99     int value = read_cell(b, possibility);
100    if (value < smallest_value && value > 0)
101    {
102      movement = i;
103      smallest_value = value;
104    }
105  }
106  // TODO: maybe check if movement has actually been set?
107  return get_direction(movement, current);
108 }
109
110 orientation desired_orientation(location current, location next)
111 {
112   if (equal_loc(current, next))
113   {
114     printf("ERROR ...\n");
115     // TODO: some proper error handling here
116   }
117   if (next.x < current.x)
118   {
119     return NORTH;
120   } else if (next.y > current.x)
121   {
122     return EAST;
123   } else if (next.x > current.x)
124   {
125     return SOUTH;
126   } else // hopefully, next.y < current_loc.y
127   {
128     return WEST;
129   }
130 }
131
132 nav_instruction format_instruction(orientation current, orientation desired)
133 {
134   int diff = (int)current - (int)desired;
135   if (diff == 2 || diff == -2)
136   {
137     printf("ERROR 222\n"); // TODO: error handling
138     return FORWARD;
139   } else if (diff == 0)
140   {
141     return FORWARD;
142   } else if (diff == 1)
143   {
144     return LEFT;
145   } else // diff == -1
146   {
147     return RIGHT;
148   }
149 }
150
151 nav_instruction get_instruction(board b, location loc, orientation o)
152 {
153   location next = next_loc(b, loc);
154   orientation desired = desired_orientation(loc, next);
155   nav_instruction instruction = format_instruction(o, desired);
156   return instruction;
```

```
157 }
```

What follows is the contents of the file located at `control/targets.h`.

```
1  //
2  // Created by Thijs van Esch on 07/06/2023.
3  //
4
5  #ifndef EPO2_CONTROL_TARGETS_H
6  #define EPO2_CONTROL_TARGETS_H
7
8  #include "algorithm/location.h"
9
10 //
11 // Initializes the target tracker.
12 //
13 void init_t_tracker(int target_count, int* targets);
14
15 //
16 // Returns the current target location.
17 //
18 location cur_target_loc(void);
19
20 //
21 // Returns the current target station.
22 //
23 int cur_target_s(void);
24
25 //
26 // Sets the target to the target that is to be reached after the current target.
27 //
28 // Should be called when a target has been reached, so calls to `cur_target_*`
29 // actually return the new target.
30 //
31 void inc_target(void);
32
33 //
34 // Returns the amount of targets that the robot still has to reach.
35 //
36 int targets_left(void);
37
38 #endif //EPO2_CONTROL_TARGETS_H
```

What follows is the contents of the file located at `control/targets.c`.

```
1  //
2  // Created by Thijs van Esch on 07/06/2023.
3  //
4
5  #include "targets.h"
6
7  #include <stdio.h>
8
9  //
10 // The following is the array containing the station identifiers of the
11 // targets for the robot:
12 //
13 int g_t_count = -1;
14 int* g_targets = NULL;
15
16 int t_index = 0;
17
18 void init_t_tracker(int target_count, int* targets)
19 {
20   g_t_count = target_count;
21   g_targets = targets;
22   t_index = 0;
23 }
24
```

```
25 int cur_target_s(void)
26 {
27 #if DEBUG_BUILD == 1
28   if (g_targets == NULL)
29   {
30     printf("ERROR: attempt to use target tracker before it was "
31            "initialized\n");
32     return -1;
33   }
34 #endif
35   return g_targets[t_index];
36 }
37
38 location cur_target_loc(void)
39 {
40   int target_s = cur_target_s();
41   return get_station_loc(target_s);
42 }
43
44 void inc_target(void)
45 {
46   printf("> reached target\n");
47   t_index += 1;
48 }
49
50 int targets_left(void)
51 {
52   return g_t_count - t_index;
53 }
```

A.3. Serial communication

What follows is the contents of the file located at `serial/params.h`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
3 //
4
5 #ifndef EPO2_SERIAL_PARAMS_H
6 #define EPO2_SERIAL_PARAMS_H
7
8 #define BAUD B9600
9 #define BYTESIZE 8
10 #define PARITY 0
11 #define STOPBITS 0
12 #define PORT "/dev/ttyUSB0"
13
14 #endif //EPO2_SERIAL_PARAMS_H
```

What follows is the contents of the file located at `serial/serial.h`.

```
1  //
2  // Created by Thijs J.A. van Esch on 30-4-23.
3  //
4
5  #ifndef EPO2_SERIAL_H
6  #define EPO2_SERIAL_H
7
8  #include <stddef.h>
9  #include <stdint.h>
10
11 //
12 // Initializes the serial interface for serial communication with the robot.
13 //
14 int init_sio();
15
16 //
17 // Makes the serial interface connect to the port with the given name.
18 //
19 // This function will have no effect if `init_sio` has been called already.
20 //
21 void set_s_port(char *port);
22
23 //
24 // Reads a single byte from the serial interface.
25 //
26 // The integer that is returned is the amount of bytes that was actually
27 // read from the serial port.
28 //
29 int read_byte(uint8_t *byte);
30
31 //
32 // Writes a single byte over the serial interface.
33 //
34 // The integer that is returned is the amount of bytes that was actually
35 // written to the serial port.
36 //
37 int write_byte(const uint8_t *byte);
38
39 //
40 // Properly closes the connection to the robot.
41 //
42 int close_sio();
43
44 #endif //EPO2_SERIAL_H
```

What follows is the contents of the file located at `serial/serial.c`.

```c
//
// Created by Thijs van Esch on 06/06/2023.
//

#include "serial.h"
#include "params.h"

//
// Most of the code in this file is adapted from an answer to a question on
// stackoverflow, which can be found here:
//
//  https://stackoverflow.com/questions/6947413
//      /how-to-open-read-and-write-from-serial-port-in-c
//
//

#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>
#include <stdio.h>

//
// This is the handle for the serial communication.
//
// Its default value is INT32_MIN. The file descriptor will be set from within
//  the `init_sio()` function.
//
int fd = INT32_MIN;

void f_termios(struct termios *tty)
{
  tty->c_cflag = (tty->c_cflag & ~CSIZE) | CS8;
  tty->c_iflag &= ~IGNBRK;
  tty->c_lflag = 0;
  tty->c_oflag = 0;
  tty->c_cc[VMIN] = 0;
  tty->c_cc[VTIME] = 5;
  tty->c_iflag &= ~(IXON | IXOFF | IXANY);
  tty->c_cflag |= (CLOCAL | CREAD);
  tty->c_cflag &= ~(PARENB | PARODD);
  tty->c_cflag |= PARITY;
  tty->c_cflag &= ~CSTOPB;
  tty->c_cflag &= ~CRTSCTS;
}

int set_attrs()
{
  struct termios tty;
  if (tcgetattr(fd, &tty) != 0)
  {
    printf("error %d from tcgetattr", errno);
    return -1;
  }
  cfsetospeed(&tty, BAUD);
  cfsetispeed(&tty, BAUD);
  f_termios(&tty);
  if (tcsetattr(fd, TCSANOW, &tty) != 0)
  {
    printf("error %d from tcsetattr", errno);
    return -1;
  }
  return 0;
}

void set_blocking()
{
  struct termios tty;
```

```
70    memset(&tty, 0, sizeof tty);
71    if (tcgetattr(fd, &tty) != 0)
72    {
73      printf("error %d from tggetattr", errno);
74      return;
75    }
76    tty.c_cc[VMIN] = 0;
77    tty.c_cc[VTIME] = 5;
78    if (tcsetattr(fd, TCSANOW, &tty) != 0)
79    {
80      printf("error %d setting term attributes", errno);
81    }
82  }
83
84  //
85  // This value may be set from the `set_s_port` function. If no value is set, the
86  //  default port should be used.
87  //
88  char *port_name = NULL;
89
90  void set_s_port(char *port)
91  {
92    port_name = port;
93  }
94
95  //
96  // Returns the port that was given to `set_s_port`, or the default port.
97  //
98  char *get_port()
99  {
100   char *port = port_name;
101   if (port == NULL)
102   {
103     port = PORT;
104   }
105   return port;
106 }
107
108 int init_sio()
109 {
110   if (fd != INT32_MIN)
111   {
112     return 0;  // interface has already been initialized
113   }
114   char *port = get_port();
115   fd = open(port, O_RDWR | O_NOCTTY);
116   if (fd < 0)
117   {
118     printf("error %d opening %s: %s\n", errno, port,
119       strerror(errno));
120     return 1;
121   }
122   set_attrs();
123   set_blocking();
124   printf("serial connection established at port %s\n", port);
125   return 0;
126 }
127
128 int read_byte(uint8_t *byte)
129 {
130   if (fd == INT32_MIN)
131   {
132     // attempting to read before interface initialization
133     //   return error code -11
134     return -11;
135   }
136   return read(fd, byte, 1) > 0;
137 }
138
139 int write_byte(const uint8_t *byte)
140 {
```

```
141   if (fd == INT32_MIN)
142   {
143     // attempting to write before interface initialization
144     //   return error code -13
145     return -13;
146   }
147   return write(fd, byte, 1) > 0;
148 }
149
150 int close_sio()
151 {
152   if (fd == INT32_MIN)
153   {
154     // attempting to close connection before initialization:
155     //   nothing to do; simply return
156     return 0;
157   }
158   printf("closed serial connection\n");
159   int error = close(fd);
160   fd = INT32_MIN;
161   return error;
162 }
```

A.4. Utility files

What follows is the contents of the file located at `util/args.h`.

```
1  //
2  // Created by Thijs van Esch on 07/06/2023.
3  //
4
5  #ifndef EPO2_ARGS_H
6  #define EPO2_ARGS_H
7
8  //
9  // Sets the given values as globals so other functions relating to the parsing
10 // of command line arguments can use them without them needing to be given
11 // each time.
12 //
13 void set_args(int argc, char *argv[]);
14
15 //
16 // Returns a boolean that is true if the program should only show a help
17 // message describing its arguments and then exit.
18 //
19 int should_help();
20
21 //
22 // Prints a description of the application and each of its arguments.
23 //
24 void show_help(void);
25
26 //
27 // Returns a boolean that is true if the application should skip the prompt
28 // telling the user to press enter to start the robot control.
29 //
30 int start_quick();
31
32 //
33 // Returns the character of the assignment.
34 //
35 // Possible values are '0', 'A' and 'B'. This is a required argument. The
36 // program will exit if an illicit value is given for the 'assignment'
37 // argument.
38 //
39 char get_assignment();
40
41 //
42 // Returns the time in millisecond the program should wait for in between
43 // receiving a byte from the robot and writing one.
```

```
44 //
45 // If none is provided, the function will return 40.
46 //
47 int get_rw_delay();
48
49 //
50 // Returns a string describing the serial port to connect to.
51 //
52 // This is a required argument.
53 //
54 char *get_s_port();
55
56 //
57 // Returns the start station.
58 //
59 int get_start_s();
60
61 //
62 // Returns the amount of target stations that were given.
63 //
64 int target_amount();
65
66 //
67 // Returns an array of target stations.
68 //
69 int *get_targets();
70
71 #endif //EPO2_ARGS_H
```

What follows is the contents of the file located at `util/args.c`.

```
1 //
2 // Created by Thijs van Esch on 07/06/2023.
3 //
4
5 #include "args.h"
6
7 #include <errno.h>
8 #include <stdio.h>
9 #include <string.h>
10 #include <stdlib.h>
11
12 //
13 // The error code for encountering a missing argument or encountering an
14 // illicit value for an argument.
15 //
16 #define MISSING_ARG_ERROR 4
17
18 //
19 // The error code for encountering a missing argument or encountering an
20 // illicit value for an argument.
21 //
22 #define ARG_VALUE_ERROR 5
23
24 struct arg_list_node;
25
26 typedef struct arg_list_node arg_list_node;
27
28 struct arg_list_node
29 {
30   char* name;
31   char* value;
32   arg_list_node* next;
33 };
34
35 struct arg_list
36 {
37   arg_list_node* first;
38   arg_list_node* last;
39 };
```

```
40
41 typedef struct arg_list arg_list;
42
43 arg_list args = { NULL, NULL };
44
45 int arg_amount()
46 {
47   arg_list_node* cursor = args.first;
48   int count = 0;
49   while (cursor != NULL)
50   {
51     count += 1;
52     cursor = cursor->next;
53   }
54   return count;
55 }
56
57 void append_arg(char* name, char* value)
58 {
59   arg_list_node* new = (arg_list_node*)malloc(sizeof(arg_list_node));
60   new->next = NULL;
61   new->name = name;
62   new->value = value;
63   int count = arg_amount();
64   if (count == 0)
65   {
66     args.first = new;
67     args.last = new;
68   } else
69   {
70     args.last->next = new;
71     args.last = new;
72   }
73 }
74
75 //
76 // Returns a true value if the given argument is not properly formatted.
77 //
78 int parse_arg(char* arg, char** arg_name, char** arg_value)
79 {
80   int t_count = 0;
81   char* token = strtok(arg, "=");
82   *arg_value = NULL;
83   while (token != NULL)
84   {
85     if (!t_count)
86     {
87       *arg_name = token;
88     } else if (t_count == 1)
89     {
90       *arg_value = token;
91     } else
92     {
93       printf("invalid argument '%s'\n", arg);
94       return 1;
95     }
96     t_count += 1;
97     token = strtok(NULL, " ");
98   }
99   return 0;
100 }
101
102 void set_args(int argc, char* argv[])
103 {
104   for (int i = 1; i < argc; ++i)
105   {
106     char* arg_name, * arg_value;
107     parse_arg(argv[i], &arg_name, &arg_value);
108     append_arg(arg_name, arg_value);
109   }
110 }
```

```
111
112 void print_args_amount()
113 {
114   int arg_count = arg_amount();
115   if (arg_count)
116   {
117     printf("a total of %d arguments were given:\n", arg_count);
118   } else
119   {
120     printf("no arguments were given\n");
121     return;
122   }
123 }
124
125 //
126 // Returns a boolean indicating whether any value has been given for the
127 // argument with the given name.
128 //
129 int arg_is_set(char* arg_name)
130 {
131   arg_list_node* cursor = args.first;
132   while (cursor != NULL)
133   {
134     if (strcmp(cursor->name, arg_name) == 0)
135     {
136       return 1;
137     }
138     cursor = cursor->next;
139   }
140   return 0;
141 }
142
143 //
144 // Returns NULL if no value has been given for the argument with the given name.
145 //
146 char* get_arg_value(char* arg_name)
147 {
148   arg_list_node* cursor = args.first;
149   while (cursor != NULL)
150   {
151     if (strcmp(cursor->name, arg_name) == 0)
152     {
153       return cursor->value;
154     }
155     cursor = cursor->next;
156   }
157   return NULL;
158 }
159
160 int should_help()
161 {
162   return arg_is_set("print_help");
163 }
164
165 // TODO: write help message to print
166 void show_help(void)
167 {
168   printf("help\n");
169 }
170
171 int start_quick()
172 {
173   return arg_is_set("start_quick");
174 }
175
176 char get_assignment()
177 {
178   if (!arg_is_set("assignment"))
179   {
180     return '\0';
181   }
```

```
182   char assignment = *get_arg_value("assignment");
183   if ('A' <= assignment && assignment <= 'Z')
184   {
185     assignment += 32;
186   }
187   return assignment;
188 }
189
190 int get_rw_delay()
191 {
192   static char* key = "rw_delay";
193   if (!arg_is_set(key))
194   {
195     return 40;
196   }
197   char* value = get_arg_value(key);
198   int rw_delay = (int)strtol(value, NULL, 10);
199   if (rw_delay == 0)
200   {
201     /* here we'll assume that a user may give an illicit number,
202        but not a number that would not fit in a 32-bit integer */
203     if (errno == EINVAL)
204     {
205       printf("illicit value for %s: '%s'\n", key, value);
206     }
207   }
208   return rw_delay;
209 }
210
211 char* get_s_port()
212 {
213   static char* key = "serial_port";
214   if (!arg_is_set(key))
215   {
216     printf(" :: no value given for '%s'\n", key);
217     exit(MISSING_ARG_ERROR);
218   }
219   return get_arg_value(key);
220 }
221
222 int get_start_s()
223 {
224   static char* key = "start_station";
225   if (!arg_is_set(key))
226   {
227     printf(" :: no value given for '%s'\n", key);
228     exit(MISSING_ARG_ERROR);
229   }
230   char* val = get_arg_value("start_station");
231   int start_station = (int)strtol(val, NULL, 10);
232   if (start_station < 1 || start_station > 12)
233   {
234     printf(" :: illicit value given for '%s': '%s'\n", key, val);
235     exit(ARG_VALUE_ERROR);
236   }
237   return start_station;
238 }
239
240 int target_amount()
241 {
242   static char* key = "target_stations";
243   if (!arg_is_set(key))
244   {
245     printf(" :: no value given for '%s'\n", key);
246     exit(MISSING_ARG_ERROR);
247   }
248   char* targets = get_arg_value(key);
249   int count = 1;
250   while (*targets)
251   {
252     if (*targets == ',')
```

```
253     {
254       count += 1;
255     } else if (*targets < '0' || *targets > '9')
256     {
257       printf(" :: illicit value given for '%s': '%s'\n",
258         key, targets);
259       exit(ARG_VALUE_ERROR);
260     }
261     targets++;
262   }
263   return count;
264 }
265
266 int* get_targets()
267 {
268   static char* key = "target_stations";
269   int t_count = target_amount();
270   if (t_count == -1)
271   {
272     printf(" :: no value given for '%s'\n", key);
273     exit(MISSING_ARG_ERROR);
274   }
275   char* given = get_arg_value(key);
276   /* t_count should be positive or equal -2, but just to make sure we
277      check t_count <= 0 */
278   if (t_count <= 0)
279   {
280     printf(" :: illicit value given for '%s': '%s'\n", key, given);
281     exit(ARG_VALUE_ERROR);
282   }
283   char copy[100];
284   strcpy(copy, given);
285   int* targets = (int*)malloc(sizeof(int) * t_count);
286   char* token = strtok(copy, ",");
287   int t_index = 0;
288   while (token != NULL)
289   {
290     int converted = (int)strtol(token, NULL, 10);
291     if (converted < 1 || converted > 12)
292     {
293       printf(" :: illicit value given for '%s': '%s'\n",
294         key, given);
295       exit(ARG_VALUE_ERROR);
296     }
297     targets[t_index] = converted;
298     t_index += 1;
299     token = strtok(NULL, ",");
300   }
301   return targets;
302 }
```

A.5. Main function

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5
6  #include "control/control.h"
7  #include "control/targets.h"
8  #include "serial/serial.h"
9  #include "util/args.h"
10
11 void wait_for_start()
12 {
13   if (start_quick())
14   {
15     return;
16   }
```

```
17    printf("to start, press ENTER");
18    char _;
19    scanf("%c", &_);
20  }
21
22  void line_follower()
23  {
24    char* s_port = get_s_port();
25    set_s_port(s_port);
26    init_sio();
27    int second_byte = 0;
28    int actual_data = 0;
29    uint8_t last_op = 0;
30    while (1)
31    {
32      if (actual_data >= 200)
33      {
34        break;
35      }
36      uint8_t incoming;
37      int bytes_read = read_byte(&incoming);
38      if (bytes_read && incoming)
39      {
40        if (second_byte)
41        {
42          uint8_t outgoing = last_op;
43          printf(" >> writing last op\n");
44          write_byte(&outgoing);
45          usleep(400 * 1000);
46          second_byte = 0;
47        } else
48        {
49          printf(" :: received 0x%02x\n", incoming);
50          uint8_t outgoing = 3;
51          if (actual_data == 5)
52          {
53            outgoing = 5;
54          }
55          printf(" >> writing 0x%02x\n", outgoing);
56          write_byte(&outgoing);
57          usleep(400 * 1000);
58          last_op = outgoing;
59          second_byte = 1;
60          actual_data += 1;
61        }
62      } else if (bytes_read)
63      {
64        printf(" .. writing 0\n");
65        uint8_t outgoing = 0;
66        write_byte(&outgoing);
67        usleep(400 * 1000);
68      }
69    }
70  }
71
72  void assignment_a()
73  {
74    char* s_port = get_s_port();
75    set_s_port(s_port);
76    init_sio();
77    int rw_delay = get_rw_delay();
78    printf("rw_delay: %d\n", rw_delay);
79    int* targets = get_targets();
80    init_t_tracker(target_amount(), targets);
81    init_control('a', get_start_s());
82    wait_for_start();
83    while (!assignment_done())
84    {
85      uint8_t incoming = 0;
86      int bytes_read = read_byte(&incoming);
87      if (incoming == 0)
```

```
88      {
89        printf("received NULL from robot\n");
90        continue;
91      }
92      if (bytes_read == 1 && incoming)
93      {
94        if (!incoming)
95        {
96          printf("received zero\n");
97          continue;
98        }
99        handle_incoming(incoming);
100       uint8_t outgoing;
101       if (get_outgoing(&outgoing))
102       {
103         write_byte(&outgoing);
104         usleep(rw_delay * 1000);// rw_delay is in ms
105       }
106     }
107   }
108   free(targets);
109 }
110
111 int main(int argc, char* argv[])
112 {
113   set_args(argc, argv);
114   if (should_help())
115   {
116     show_help();
117     return 0;
118   }
119   char assignment = get_assignment();
120   if (assignment == '0')
121   {
122     line_follower();
123   } else if (assignment == 'a')
124   {
125     assignment_a();
126   } else
127   {
128     printf("unknown or not implemented assignment: '%c'\n",
129            assignment);
130   }
131   return 0;
132 }
```

B

Application testing

B.1. Testing interface

What follows is the contents of the file located at `tests/testing.h`.

```
1  //
2  // Created by Thijs van Esch on 08/06/2023.
3  //
4
5  #ifndef EPO2_TESTING_H
6  #define EPO2_TESTING_H
7
8  //
9  // Exits with an error code if ACTUAL does not equal EXPECTED. An error will be
10 // printed stating that VARNAME has a wrong value.
11 //
12 void assert_equal(int actual, int expected, const char *varname);
13
14 #endif //EPO2_TESTING_H
```

What follows is the contents of the file located at `tests/testing.c`.

```
1  //
2  // Created by Thijs van Esch on 08/06/2023.
3  //
4
5  #include <stdlib.h>
6  #include <stdio.h>
7
8  #include "testing.h"
9
10 //
11 // The status code with which the program should exit if a test failed.
12 //
13 #define TEST_ERROR 7
14
15 void assert_equal(int actual, int expected, const char *varname)
16 {
17   if (actual != expected)
18   {
19     printf("wrong value encountered for '%s'\n", varname);
20     printf(" :: expected %d, encountered %d\n", expected, actual);
21     exit(TEST_ERROR);
22   }
23 }
```

B.2. Serial communication test

```c
//
// Created by Thijs van Esch on 07/06/2023.
//

#include <stdio.h>
#include <unistd.h>

#include "../serial/serial.h"

char *SERIAL_PORT = "/dev/cu.usbserial-AE01CSS5";

void write_something()
{
  uint8_t c = 'a';
  while (c != 'z' + 1)
  {
    int bytes_written = write_byte(&c);
    if (!bytes_written)
    {
      printf("no bytes written\n");
    }
    usleep(200 * 1000);
    c += 1;
  }
}

void read_something()
{
  int it_c = 0;
  while (it_c < 999777)
  {
    uint8_t incoming = 0;
    int bytes_read = read_byte(&incoming);
    if (bytes_read && incoming)
    {
      printf("received data: %c (0x%02x)\n",
             incoming, incoming);
    } else if (bytes_read)
    {
      printf("received NULL\n");
    }
    it_c += 1;
  }
}

int main()
{
  set_s_port(SERIAL_PORT);
  init_sio();
  write_something();
  read_something();
  return 0;
}
```

B.3. Application tests

What follows is the contents of the file located at `tests/queue.c`.

```c
//
// Created by Thijs van Esch on 06/06/2023.
//

#include <stdio.h>

#include "../control/algorithm/queue.h"

#include "testing.h"

//
// The name of this test.
```

```
13 //
14 static const char *TEST_NAME = "queue";
15
16 //
17 // Tests if items can be properly added to the queue.
18 //
19 queue *append_test()
20 {
21   queue* q = create_queue();
22   assert_equal(queue_length(q), 0, "queue_length");
23   location loc = {0, 0};
24   append_queue(q, loc, 1);
25   assert_equal(queue_length(q), 1, "queue_length");
26   append_queue(q, loc, 2);
27   append_queue(q, loc, 3);
28   assert_equal(queue_length(q), 3, "queue_length");
29   return q;
30 }
31
32 //
33 // Tests if items can be properly removed from the queue.
34 //
35 void remove_test(queue *q)
36 {
37   int dist;
38   location loc;
39   from_queue(q, &loc, &dist);
40   assert_equal(dist, 1, "distance (0)");
41   from_queue(q, &loc, &dist);
42   assert_equal(dist, 2, "distance (1)");
43   assert_equal(queue_length(q), 1, "queue_length (3)");
44   from_queue(q, &loc, &dist);
45   assert_equal(dist, 3, "distance (2)");
46   assert_equal(queue_length(q), 0, "queue_length (4)");
47 }
48
49 int main()
50 {
51   printf(" -- %s test\n", TEST_NAME);
52   queue *q = append_test();
53   remove_test(q);
54   printf("%s test success\n", TEST_NAME);
55   return 0;
56 }
```

What follows is the contents of the file located at `tests/algorithm.c`.

```
1 //
2 // Created by Thijs van Esch on 06/06/2023.
3 //
4
5 #include <stdio.h>
6
7 #include "../control/algorithm/board.h"
8
9 #include "testing.h"
10
11 //
12 // The name of this test.
13 //
14 static const char* TEST_NAME = "algorithm";
15
16 void check_board_values(board actual, int expected[13][13], const char *type)
17 {
18   for (int x = 0; x < 13; ++x)
19   {
20     for (int y = 0; y < 13; ++y)
21     {
22       if (!expected[x][y])
23       {
```

```
24          continue;
25        }
26        char varname[100];
27        sprintf(varname, "%s board_value (%d/%d)", type, x, y);
28        assert_equal(actual[x][y], expected[x][y], varname);
29      }
30    }
31  }
32
33  //
34  // The expected value of a board after applying the Lee algorithm to calculate
35  // the shortest path from station 1 to station 12.
36  //
37  // A value of zero means that any value as obtained from the algorithm for that
38  // cell will be considered valid.
39  //
40  static int NO_MINES_EXPECTED[13][13] =
41    {
42      { 0, 0, 0, 0,  13, 0,  15, 0,  17, 0,  0,  0,  0 },
43      { 0, 0, 0, 0,  12, 0,  14, 0,  16, 0,  0,  0,  0 },
44      { 0, 0, 9, 10, 11, 12, 13, 14, 15, 16, 17, 0,  0 },
45      { 0, 0, 8, 0,  10, 0,  12, 0,  14, 0,  16, 0,  0 },
46      { 9, 8, 7, 8,  9,  10, 11, 12, 13, 14, 15, 16, 17 },
47      { 0, 0, 6, 0,  8,  0,  10, 0,  12, 0,  14, 0,  0 },
48      { 7, 6, 5, 6,  7,  8,  9,  10, 11, 12, 13, 14, 15 },
49      { 0, 0, 4, 0,  6,  0,  8,  0,  10, 0,  12, 0,  0 },
50      { 1, 2, 3, 4,  5,  6,  7,  8,  9,  10, 11, 12, 13 },
51      { 0, 0, 4, 0,  6,  0,  8,  0,  10, 0,  12, 0,  0 },
52      { 0, 0, 5, 6,  7,  8,  9,  10, 11, 12, 13, 0,  0 },
53      { 0, 0, 0, 0,  8,  0,  10, 0,  12, 0,  0,  0,  0 },
54      { 0, 0, 0, 0,  9,  0,  11, 0,  13, 0,  0,  0,  0 }
55    };
56
57  //
58  // Tests if the algorithm can determine the most optimal path for a board that
59  // contains no mines.
60  //
61  void no_mines_test()
62  {
63    board b = new_board();
64    calc_path(b, get_station_loc(1), get_station_loc(12));
65    check_board_values(b, NO_MINES_EXPECTED, "no mines");
66    printf("%s (no mines) test success\n", TEST_NAME);
67  }
68
69  //
70  // The expected value of a board after applying the Lee algorithm to calculate
71  // the shortest path from station 1 to station 12.
72  //
73  // A value of zero means that any value as obtained from the algorithm for that
74  // cell will be considered valid.
75  //
76  static int WITH_MINES_EXPECTED[13][13] =
77    {
78      {0, 0, 0,  0,  13, 0,  15, 0,  17,  0,  0,  0,  0},
79      {0, 0, 0,  0,  12, 0,  14, 0,  16,  0,  0,  0,  0},
80      {0, 0, 9, 10,  11, 12, 13, 14, 15,  16, 17, 0,  0},
81      {0, 0, 8,  0,  10, 0,  12, 0,  14,  0,  18, 0,  0},
82      {9, 8, 7,  8,  9,  10, 11, 12, 13, -3,  19, 20, 21},
83      {0, 0, 6,  0,  8,  0,  10, 0, -3,   0,  20, 0,  0},
84      {7, 6, 5,  6,  7,  8,  9, -3,  23,  22, 21, 22, 23},
85      {0, 0, 4,  0,  6,  0, -3,  0,  24,  0,  22, 0,  0},
86      {1, 2, 3,  4,  5, -3,  27, 26, 25,  24, 23, 24, 25},
87      {0, 0, 4,  0, -3,  0,  28, 0,  26,  0,  24, 0,  0},
88      {0, 0, 5, -3,  31, 30, 29, 28, 27,  26, 25, 0,  0},
89      {0, 0, 0,  0,  32, 0,  30, 0,  28,  0,  0,  0,  0},
90      {0, 0, 0,  0,  33, 0,  31, 0,  29,  0,  0,  0,  0}
91    };
92
93  //
94  // Writes the location of some mines to the board.
```

```
95 //
96 void set_mines(board b)
97 {
98   int x = 4, y = 9;
99   for (int i = 0; i < 7; ++i)
100  {
101    location loc = { x + i, y - i };
102    write_cell(b, loc, -3);
103  }
104 }
105
106 //
107 // Runs the test for the algorithm.
108 //
109 void test_algorithm()
110 {
111   board b = new_board();
112   set_mines(b);
113   calc_path(b, get_station_loc(1), get_station_loc(12));
114   check_board_values(b, WITH_MINES_EXPECTED, "with mines");
115   printf("%s (with mines) test success\n", TEST_NAME);
116 }
117
118 int main()
119 {
120   printf(" -- %s test\n", TEST_NAME);
121   no_mines_test();
122   test_algorithm();
123   return 0;
124 }
```

What follows is the contents of the file located at `tests/navigation.c`.

```
1  //
2  // Created by Thijs van Esch on 12/06/2023.
3  //
4
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  #include "../control/navigation.h"
9
10 #include "testing.h"
11
12 //
13 // The name of this test.
14 //
15 static const char* TEST_NAME = "navigation";
16
17 //
18 // The board that will be used for determining navigational instructions.
19 //
20 static int BOARD[13][13] =
21   {
22     { -1, -1, -1, -1,  13, -1,  15, -1,  17, -1,  -1,  -1,  -1 },
23     { -1, -1, -1, -1,  12, -1,  14, -1,  16, -1,  -1,  -1,  -1 },
24     { -1, -1,  9,  10, 11,  12, 13,  14, 15,  16,  17, -1,  -1 },
25     { -1, -1,  8, -1,  10, -1,  12, -1,  14, -1,  16, -1,  -1 },
26     {  9,  8,  7,  8,  9,   10, 11,  12, 13,  14,  15,  16,  17 },
27     { -1, -1,  6, -1,  8,  -1,  10, -1,  12, -1,  14, -1,  -1 },
28     {  7,  6,  5,  6,  7,   8,  9,   10, 11,  12,  13,  14,  15 },
29     { -1, -1,  4, -1,  6,  -1,  8,  -1,  10, -1,  12, -1,  -1 },
30     {  1,  2,  3,  4,  5,   6,  7,   8,  9,   10,  11,  12,  13 },
31     { -1, -1,  4, -1,  6,  -1,  8,  -1,  10, -1,  12, -1,  -1 },
32     { -1, -1,  5, -1,  7,   8,  9,   10, 11,  12,  13, -1,  -1 },
33     { -1, -1, -1, -1,  8,  -1,  10, -1,  12, -1,  -1,  -1,  -1 },
34     { -1, -1, -1, -1,  9,  -1,  11, -1,  13, -1,  -1,  -1,  -1 }
35   };
36
37 board get_board()
```

```
38 {
39   board b = (int **) malloc(sizeof (int *) * 13);
40   for (int x = 0; x < 13; ++x)
41   {
42     b[x] = (int *) malloc(sizeof (int) * 13);
43     for (int y = 0; y < 13; ++y)
44     {
45       b[x][y] = BOARD[x][y];
46     }
47   }
48   return b;
49 }
50
51 void nav_test()
52 {
53   board b = get_board();
54   location loc;
55   nav_instruction instruction;
56   loc.x = 10;
57   loc.y = 4;
58   instruction = get_instruction(b, loc, NORTH);
59   assert_equal((int)instruction, (int)'f', "instruction (0)");
60   loc.y = 6;
61   instruction = get_instruction(b, loc, EAST);
62   assert_equal((int)instruction, (int)'l', "instruction (0)");
63 }
64
65 int main()
66 {
67   printf(" -- %s test\n", TEST_NAME);
68   nav_test();
69   printf("%s test success\n", TEST_NAME);
70   return 0;
71 }
```

B.4. Test script

```
#!/usr/bin/env bash

TEST_NAMES=("queue" "algorithm" "navigation")

for T_NAME in "${TEST_NAMES[@]}"
do
        EXECUTABLE="cmake-build-debug/test_$T_NAME"
        if [ ! -x "$EXECUTABLE" ]
        then
                echo " !! no executable seems to exist for test '$T_NAME'"
                echo "    skipping ..."
                continue
        fi
        echo "running test '$T_NAME' ..."
        if ! $EXECUTABLE
        then
                echo "test '$T_NAME' failed"
                exit 1
        fi
done

echo " "
echo "TEST SUCCESS"
```

C

Application build file

```
cmake_minimum_required(VERSION 3.22)
project(epo_redone C)

set(CMAKE_C_STANDARD 11)

#
# To disable much debug logging, set to 0 here.
#
add_compile_definitions(DEBUG_BUILD=1)

#
# Turning on all warnings, and warnings become errors
#
add_compile_options(-Wall -Wextra -Wpedantic)

#
# The following are the lists containing the source files:
#

set(
        SERIAL_SRC
        serial/serial.c
        serial/serial.h
        serial/params.h)

set(
        ALG_SRC
        control/targets.c
        control/targets.h
        control/algorithm/board.c
        control/algorithm/board.h
        control/algorithm/default.h
        control/algorithm/location.c
        control/algorithm/location.h
        control/algorithm/queue.c
        control/algorithm/queue.h)

set(
        CONTROL_SRC
```

```
        ${ALG_SRC}
        control/comm.c
        control/comm.h
        control/navigation.c
        control/navigation.h
        control/control.c
        control/control.h)

set(
        UTIL_SRC
        util/args.c
        util/args.h)

#
# The main executable
#
add_executable(epo_redone ${SERIAL_SRC} ${CONTROL_SRC} ${UTIL_SRC} main.c)

#
# Tests:
#
set(
        TEST_SRC
        tests/testing.c
        tests/testing.h)

add_executable(test_serial ${TEST_SRC} ${SERIAL_SRC} tests/serial.c)

add_executable(test_queue ${TEST_SRC} ${ALG_SRC} tests/queue.c)
add_executable(test_algorithm ${TEST_SRC} ${CONTROL_SRC} tests/algorithm.c)
add_executable(test_navigation ${TEST_SRC} ${CONTROL_SRC} tests/navigation.c)
```

D

Code formatting

What follows is the `.clang-format` file used for the C-code written for this project. This file contains rules that the formatter can use to automatically format all code.

```
# Generated from CLion C/C++ Code Style settings
BasedOnStyle: LLVM
AccessModifierOffset: 0
AlignAfterOpenBracket: Align
AlignConsecutiveAssignments: None
AlignOperands: DontAlign
AllowAllArgumentsOnNextLine: false
AllowAllConstructorInitializersOnNextLine: false
AllowAllParametersOfDeclarationOnNextLine: false
AllowShortBlocksOnASingleLine: Always
AllowShortCaseLabelsOnASingleLine: false
AllowShortFunctionsOnASingleLine: None
AllowShortIfStatementsOnASingleLine: Always
AllowShortLambdasOnASingleLine: All
AllowShortLoopsOnASingleLine: true
AlwaysBreakAfterReturnType: None
AlwaysBreakTemplateDeclarations: MultiLine
BreakBeforeBraces: Custom
BraceWrapping:
  AfterCaseLabel: false
  AfterClass: true
  AfterControlStatement: Always
  AfterEnum: true
  AfterFunction: true
  AfterNamespace: true
  AfterUnion: true
  BeforeCatch: true
  BeforeElse: false
  IndentBraces: false
  SplitEmptyFunction: true
  SplitEmptyRecord: true
BreakBeforeBinaryOperators: NonAssignment
BreakBeforeTernaryOperators: false
BreakConstructorInitializers: BeforeColon
BreakInheritanceList: BeforeColon
ColumnLimit: 0
```

```
CompactNamespaces: false
ContinuationIndentWidth: 8
IndentCaseLabels: true
IndentPPDirectives: None
IndentWidth: 8
KeepEmptyLinesAtTheStartOfBlocks: true
MaxEmptyLinesToKeep: 1
NamespaceIndentation: All
ObjCSpaceAfterProperty: false
ObjCSpaceBeforeProtocolList: true
PointerAlignment: Left
ReflowComments: false
SpaceAfterCStyleCast: false
SpaceAfterLogicalNot: false
SpaceAfterTemplateKeyword: false
SpaceBeforeAssignmentOperators: true
SpaceBeforeCpp11BracedList: false
SpaceBeforeCtorInitializerColon: true
SpaceBeforeInheritanceColon: true
SpaceBeforeParens: ControlStatements
SpaceBeforeRangeBasedForLoopColon: true
SpaceInEmptyParentheses: false
SpacesBeforeTrailingComments: 0
SpacesInAngles: false
SpacesInCStyleCastParentheses: false
SpacesInContainerLiterals: false
SpacesInParentheses: false
SpacesInSquareBrackets: false
TabWidth: 8
UseTab: ForContinuationAndIndentation
```

E

Mine sensor

E.1. Mine sensor top entity

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity mine_sensor_top is
6          port (  clk             : in std_logic;
7                  mine_sensor_in  : in std_logic;
8                  mine_out        : out std_logic
9          );
10 end entity mine_sensor_top;
11
12 architecture structural of mine_sensor_top is
13         component mine_sensor is
14                 port (  clk     : in std_logic;
15                         reset   : in std_logic;
16                         sensor  : in std_logic;
17                         mine_out: out std_logic
18                 );
19         end component mine_sensor;
20
21         component mine_counter is
22                 port (  clk             : in std_logic;
23                         mine_out        : in std_logic;
24                         reset           : out std_logic
25                 ) ;
26         end component mine_counter;
27
28 signal mine_reset_signal, mine_out_signal: std_logic;
29
30
31 begin
32         mine_detector   : mine_sensor port map (        clk             =>      clk,
33                                                         reset           =>      mine_reset_signal,
34                                                         sensor          =>      mine_sensor_in,
35                                                         mine_out        =>      mine_out_signal
36                                                 );
37
38         count           : mine_counter port map (       clk             =>      clk,
39                                                         mine_out        =>      mine_out_signal,
40                                                         reset           =>      mine_reset_signal
41                                                 );
42         mine_out <= mine_out_signal;
43 end architecture structural;
```

E.2. Mine sensor FSM

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity mine_sensor is
6          port (  clk             : in  std_logic;
7                  reset           : in  std_logic;
8                  sensor          : in  std_logic;                mine_out        : out std_logic );
9  end entity mine_sensor;
10
11 architecture behavioural of mine_sensor is
12
13 type sensor_states is (reset_state, count_state, compare_state, detected_state);
14 signal count, new_count: unsigned (11 downto 0) := to_unsigned(0,12);
15 signal state, next_state : sensor_states;
16 begin
17         process(clk)
18                 begin
19                         if (rising_edge(clk)) then
20                                 if (reset = '1') then
21                                         state <= reset_state;
22                                         count <= (others => '0');
23                                 else
24                                         state <= next_state;
25                                         count <= new_count;
26                                 end if;
27                         end if;
28                 end process;
29
30         process(state, count, sensor)
31                 begin
32                 case state is
33                         when reset_state =>
34                                 mine_out <= '0';
35                                 new_count <= (others => '0');
36                                 if (sensor = '1') then
37                                         next_state <= reset_state;
38                                 else
39                                         next_state <= count_state;
40                                 end if;
41
42                         when count_state =>
43                                 mine_out <= '0';
44                                 if (sensor = '0') then
45                                         new_count <= count + 1;
46                                         next_state <= count_state;
47                                 else
48                                         new_count <= count;
49                                         next_state <= compare_state;
50                                 end if;
51
52                         when compare_state =>
53                                 mine_out <= '0';
54                                 new_count <= count;
55                                 if (count > 2250) then
56                                         next_state <= detected_state;
57                                 else
58                                         next_state <= reset_state;
59                                 end if;
60
61                         when detected_state =>
62                                 mine_out <= '1';
63                                 new_count <= count;
64                                 next_state <= detected_state;
65
66                         when others =>
67                         end case;
68                 end process;
69 end architecture behavioural;
```

E.3. Mine sensor counter reset

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity mine_counter is
6          port (  clk             : in std_logic;
7                  mine_out        : in std_logic;
8                  reset           : out std_logic
9          ) ;
10  end entity mine_counter ;
11
12  architecture behavioural of mine_counter is
13
14         signal count, new_count : unsigned (19 downto 0);
15
16  begin
17         process ( clk )
18         begin
19                 if ( clk'event and clk = '1' ) then
20                         if ( mine_out = '0' ) then
21
22                                 count <= ( others => '0');
23                                 reset <= '0';
24
25                         elsif ( count = 4000 ) then
26
27                                 count <= count;
28                                 reset <= '1';
29
30                         else
31                                 count <= new_count;
32
33                         end if;
34                 end if;
35         end process;
36
37
38         process ( count )
39         begin
40
41                 new_count <= count + 1;
42
43         end process;
44  end architecture behavioural;
```

F

Appendix for C

$$
\begin{bmatrix}
0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
1,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,1,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1, \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,
\end{bmatrix}
$$

Table F.1: Schematic overview of the adjacent matrix

G

VHDL codes and their corresponding testbenches for the line follower

G.1. Time base

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity timebase is
6          port (  clk             : in std_logic;
7                  reset           : in std_logic;
8                  count_out       : out std_logic_vector (19 downto 0)
9          ) ;
10  end entity timebase ;
11
12  architecture behavioural of timebase is
13
14         signal count, new_count : unsigned (19 downto 0);
15
16  begin
17         process ( clk )
18         begin
19                 if ( clk'event and clk ='1' ) then
20                         if ( reset = '1' ) then
21                                 count <= ( others => '0');
22                         else
23                                 count <= new_count;
24                         end if;
25                 end if;
26         end process;
27
28
29         process ( count )
30         begin
31
32                 new_count <= count + 1;
33
34         end process;
35
36         count_out <= std_logic_vector ( count );
37  end architecture behavioural;
```

G.2. Time base testbench

To check whether the timebase works as intended, a testbench was made. The simulation of this testbench can be observed in Figure G.1.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity testbench is
5  end entity testbench;
6
7  architecture struct of testbench is
8
9          component counter is
10                 port (  clk      : in std_logic;
11                         reset    : in std_logic;
12                         count_out : out std_logic_vector(19 downto 0)
13                 ) ;
14
15         end component counter;
16
17
18          signal clk      : std_logic;
19          signal reset    : std_logic;
20          signal count_out : std_logic_vector (19 downto 0);
21
22 begin
23
24          clk   <=       '1' after 0 ns,
25                         '0' after 10 ns when clk /= '0' else '1' after 10 ns;
26
27          reset <=       '1' after 0 ns,
28                         '0' after 11 ns,
29                         '1' after 20000020 ns;
30
31
32 T1: counter port map ( clk      => clk,
33                                  reset    => reset,
34                                  count_out => count_out);
35
36 end architecture struct;
```

Figure G.1: Testbench of the timebase

G.3. Motor control

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  -- Please add necessary libraries:
6
7  entity motorcontrol is
8          port (  clk             : in    std_logic;
9                  reset           : in    std_logic;
10                 direction       : in    std_logic;
11                 count_in        : in    std_logic_vector (19 downto 0);  -- Please enter upper bound
12
13                 pwm             : out   std_logic
14         );
15 end entity motorcontrol;
16
17 architecture behavioural of motorcontrol is
18
19         type motorcontrol_state is (state0, state1, state2);
20
21         signal state, new_state: motorcontrol_state;
22
23 begin
24
25         process (clk)
26         begin
27                 if (rising_edge (clk)) then
28                         if (reset = '1') then
29                                 state <= state0;
30                         else
31                                 state <= new_state;
32                         end if;
33                 end if;
34         end process;
35
36
37         process (direction, count_in, state)
38         begin
39                 case state is
40
41                         when state0 => pwm <= '0';
42                                 new_state<=state1;
43
44
45                         when state1 => pwm <= '1';
46                                 if (direction ='0' and unsigned(count_in) >= to_unsigned(50000, 20)) then
47                                         new_state <= state2;
48
49                                 elsif (direction ='1' and unsigned(count_in) >= to_unsigned(100000, 20))
50                                     then new_state <= state2;
51
52                                 else
53                                         new_state <= state1;
54                                 end if;
55
56
57                         when state2 => pwm <='0';
58                                 new_state <= state2;
59
60                 end case;
61         end process;
62 end architecture behavioural;
```

G.4. Motor control testbench

To check whether the motor control works as intended, a testbench was made. The simulation of this testbench can be observed in Figure G.2.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity motor_tb is
5  end entity motor_tb;
6
7  architecture structural of motor_tb is
8
9          component timebase is
10                 port (  clk                     : in    std_logic;
11                         reset                   : in    std_logic;
12
13                         count_out               : out   std_logic_vector (19 downto 0)
14                 );
15         end component timebase;
16
17         component motorcontrol is
18                 port (  clk                     : in    std_logic;
19                         reset                   : in    std_logic;
20                         direction               : in    std_logic;
21                         count_in                : in    std_logic_vector (19 downto 0);
22
23                         pwm                     : out   std_logic
24                 );
25         end component motorcontrol;
26
27         signal  clk, reset, direction          : std_logic;
28         signal  count                          : std_logic_vector (19 downto 0);
29         signal  pwm                            : std_logic;
30
31 begin
32
33         lbl0: timebase port map (       clk                     => clk,
34                                         reset                   => reset,
35                                         count_out               => count
36                                 );
37
38         lbl1: motorcontrol port map (   clk                     => clk,
39                                         reset                   => reset,
40                                         direction               => direction,
41                                         count_in                => count,
42                                         pwm                     => pwm
43                                 );
44
45         -- 20 ns = 50 MHz
46         clk                     <=      '0' after 0 ns,
47                                         '1' after 10 ns when clk /= '1' else '0' after 10 ns;
48
49         reset                   <=      '1' after 0 ns,
50                                         '0' after 20 ns,
51                                         '1' after 20000000 ns,
52                                         '0' after 20000020 ns,
53                                         '1' after 40000000 ns,
54                                         '0' after 40000020 ns;
55
56         direction               <=      '0' after 0 ns,
57                                         '1' after 20000000 ns;
58
59 end architecture structural;
```

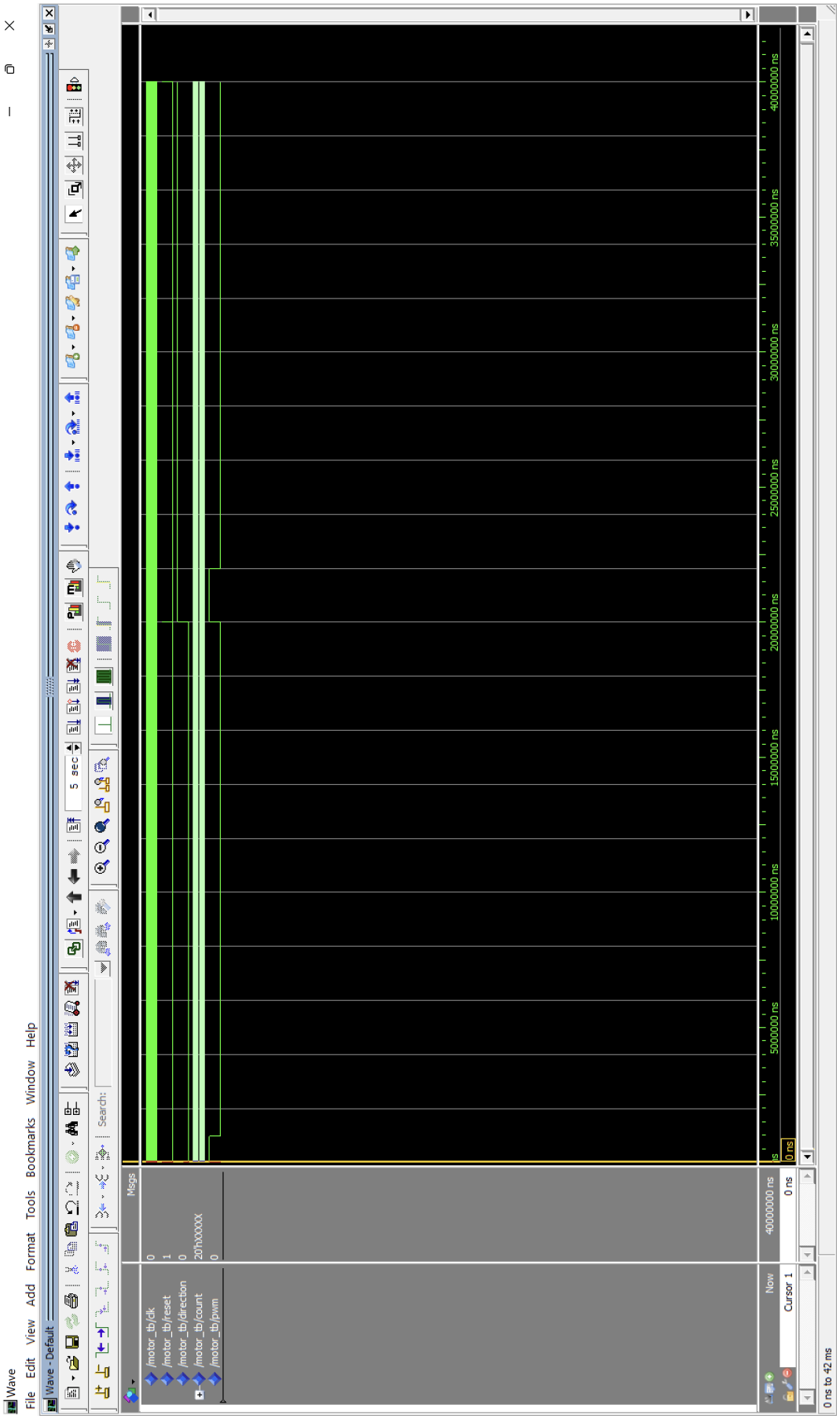

Figure G.2: Testbench of the motor control

G.5. Input buffer

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  -- Please add necessary libraries:
5
6
7  entity inputbuffer is
8          port (  clk             : in    std_logic;
9
10                 sensor_l_in     : in    std_logic;
11                 sensor_m_in     : in    std_logic;
12                 sensor_r_in     : in    std_logic;
13
14                 sensor_l_out    : out   std_logic;
15                 sensor_m_out    : out   std_logic;
16                 sensor_r_out    : out   std_logic
17         );
18 end entity inputbuffer;
19
20
21
22
23 architecture structural of inputbuffer is
24
25 component flipflop is
26
27         port(clk, sensor_l_in, sensor_m_in, sensor_r_in: in std_logic;
28             sensor_l_out, sensor_m_out, sensor_r_out: out std_logic);
29 end component;
30
31 signal s1, s2, s3: std_logic;
32
33 begin
34
35 FF1: flipflop port map( clk => clk,
36
37                         sensor_l_in => sensor_l_in,
38                         sensor_m_in => sensor_m_in,
39                         sensor_r_in => sensor_r_in,
40
41                         sensor_l_out => s1,
42                         sensor_m_out => s2,
43                         sensor_r_out => s3);
44
45 FF2: flipflop port map( clk => clk,
46
47                         sensor_l_in => s1,
48                         sensor_m_in => s2,
49                         sensor_r_in => s3,
50
51                         sensor_l_out => sensor_l_out,
52                         sensor_m_out => sensor_m_out,
53                         sensor_r_out => sensor_r_out);
54
55 end structural;
```

G.6. Input buffer testbench

To check whether the input buffer works as intended, a testbench was made. The simulation of this testbench can be observed in Figure G.3.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity inputbuffer_tb is
6  end entity inputbuffer_tb;
7
8  architecture structural of inputbuffer_tb is
9
10         component inputbuffer is
11                 port (  clk             : in    std_logic;
12
13                         sensor_l_in     : in    std_logic;
14                         sensor_m_in     : in    std_logic;
15                         sensor_r_in     : in    std_logic;
16
17                         sensor_l_out    : out   std_logic;
18                         sensor_m_out    : out   std_logic;
19                         sensor_r_out    : out   std_logic
20                 );
21         end component inputbuffer;
22
23
24         signal clk, sensor_l_in, sensor_m_in, sensor_r_in, sensor_l_out,
25                 sensor_m_out, sensor_r_out: std_logic;
26
27
28 begin
29         lb10: inputbuffer port map(     clk                     => clk,
30                                         sensor_l_in             => sensor_l_in,
31                                         sensor_m_in             => sensor_m_in,
32                                         sensor_r_in             => sensor_r_in,
33                                         sensor_l_out            => sensor_l_out,
34                                         sensor_m_out            => sensor_m_out,
35                                         sensor_r_out            => sensor_r_out
36                                 );
37
38         clk                     <=      '0' after 0 ns,
39                                         '1' after 10 ns when clk /= '1' else '0' after 10 ns;
40
41
42
43         sensor_l_in             <=      '0' after 0 ns,
44                                         '1' after 5 ns,
45                                         '0' after 35 ns;
46
47         sensor_m_in             <=      '0' after 0 ns,
48                                         '1' after 15 ns;
49
50         sensor_r_in             <=      '0' after 0 ns,
51                                         '1' after 20 ns;
52
53 end architecture structural;
```

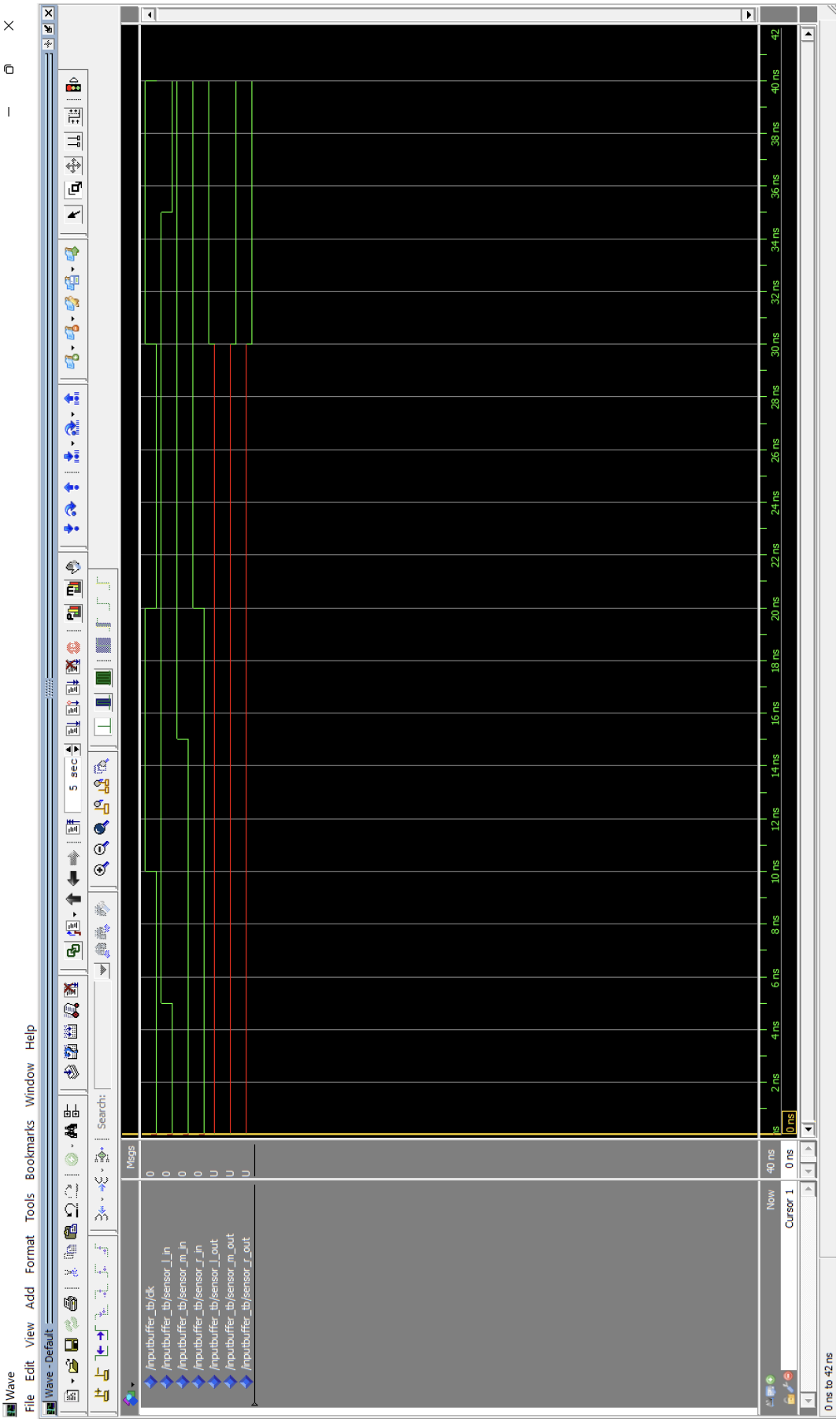

Figure G.3: Testbench of the inputbuffer

G.7. 3bit_flipflop

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  -- Please add necessary libraries:
5
6
7  entity flipflop is
8          port (  clk             : in    std_logic;
9
10                 sensor_l_in     : in    std_logic;
11                 sensor_m_in     : in    std_logic;
12                 sensor_r_in     : in    std_logic;
13
14                 sensor_l_out    : out   std_logic;
15                 sensor_m_out    : out   std_logic;
16                 sensor_r_out    : out   std_logic
17         );
18 end entity flipflop;
19
20
21 architecture behavioral of flipflop is
22
23 begin
24         process (clk)
25         begin
26                 if rising_edge(clk) then
27                         sensor_l_out <= sensor_l_in;
28                         sensor_m_out <= sensor_m_in;
29                         sensor_r_out <= sensor_r_in;
30
31                 end if;
32         end process;
33
34 end behavioral;
```

G.8. Controller

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5
6  entity controller is
7          port (  clk                     : in    std_logic;
8                  reset                   : in    std_logic;
9
10                 sensor_l                : in    std_logic;
11                 sensor_m                : in    std_logic;
12                 sensor_r                : in    std_logic;
13
14                 count_in                : in    std_logic_vector (19 downto 0);
15                 count_reset             : out   std_logic;
16
17                 motor_l_reset           : out   std_logic;
18                 motor_l_direction       : out   std_logic;
19
20                 motor_r_reset           : out   std_logic;
21                 motor_r_direction       : out   std_logic
22         );
23
24 end entity controller;
25
26 architecture behavioural of controller is
27
28         type controller_state is (state_r, state_f, state_gl, state_sl, state_gr, state_sr);
29
30         signal state, new_state: controller_state;
31
32
33 begin
34         process (clk)
35         begin
36                 if (rising_edge (clk)) then
37                         if (reset = '1') then
38                                 state <= state_r;
39                         else
40                                 state <= new_state;
41                         end if;
42                 end if;
43         end process;
44
45
46
47         process (sensor_l, sensor_m, sensor_r, count_in, state)
48         begin
49                 case state is
50
51                 when state_r =>         count_reset <= '1';
52                                         motor_l_reset <= '1';
53                                         motor_r_reset <= '1';
54
55                                         motor_l_direction <= '0';
56                                         motor_r_direction <= '0';
57
58                                 if ((sensor_l = '0') and (sensor_m = '0') and (sensor_r = '0')) then
59                                         new_state <= state_f;
60
61
62                                 elsif (sensor_l = '0' and sensor_m = '0' and sensor_r = '1') then
63                                         new_state <= state_gl;
64
65
66                                 elsif (sensor_l = '0' and sensor_m = '1' and sensor_r = '0') then
67                                         new_state <= state_f;
68
69
```

```
70                              elsif (sensor_l = '0' and sensor_m = '1' and sensor_r = '1') then
71                                      new_state <= state_sl;
72
73
74
75                              elsif (sensor_l = '1' and sensor_m = '0' and sensor_r = '0') then
76                                      new_state <= state_gr;
77
78
79
80                              elsif (sensor_l = '1' and sensor_m = '0' and sensor_r = '1') then
81                                      new_state <= state_f;
82
83
84
85                              elsif (sensor_l = '1' and sensor_m = '1' and sensor_r = '0') then
86                                      new_state <= state_sr;
87
88
89
90                              elsif (sensor_l = '1' and sensor_m = '1' and sensor_r = '1') then
91                                      new_state <= state_f;
92
93
94                              else
95                                      new_state <= state_r;
96
97                              end if;
98
99
100
101                     when state_f =>         count_reset <= '0';
102                                             motor_l_reset <= '0';
103                                             motor_r_reset <= '0';
104
105                                             motor_l_direction <= '1';
106                                             motor_r_direction <= '0';
107
108
109                             if (unsigned(count_in) < to_unsigned(1000000, 20)) then
110                                     new_state <= state_f;
111
112                             elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
113                                     new_state <= state_r;
114
115                             end if;
116
117
118                     when state_gl =>        count_reset <= '0';
119                                             motor_l_reset <= '1';
120                                             motor_r_reset <= '0';
121
122                                             motor_l_direction <= '0';
123                                             motor_r_direction <= '0';
124
125                             if (unsigned(count_in) < to_unsigned(1000000, 20)) then
126                                     new_state <= state_gl;
127
128                             elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
129                                     new_state <= state_r;
130
131                             end if;
132
133
134
135                     when state_sl =>        count_reset <= '0';
136                                             motor_l_reset <= '0';
137                                             motor_r_reset <= '0';
138
139                                             motor_l_direction <= '0';
140                                             motor_r_direction <= '0';
```

```
141
142                                 if (unsigned(count_in) < to_unsigned(1000000, 20)) then
143                                         new_state <= state_sl;
144
145                                 elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
146                                         new_state <= state_r;
147
148                                 end if;
149
150
151
152                 when state_gr =>        count_reset <= '0';
153                                         motor_l_reset <= '0';
154                                         motor_r_reset <= '1';
155
156                                         motor_l_direction <= '1';
157                                         motor_r_direction <= '0';
158
159                                 if (unsigned(count_in) < to_unsigned(1000000, 20)) then
160                                         new_state <= state_gr;
161
162                                 elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
163                                         new_state <= state_r;
164
165                                 end if;
166
167
168                 when state_sr =>        count_reset <= '0';
169                                         motor_l_reset <= '0';
170                                         motor_r_reset <= '0';
171
172                                         motor_l_direction <= '1';
173                                         motor_r_direction <= '1';
174
175                                 if (unsigned(count_in) < to_unsigned(1000000, 20)) then
176                                         new_state <= state_sr;
177
178                                 elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
179                                         new_state <= state_r;
180
181                                 end if;
182
183                 end case;
184         end process;
185 end architecture behavioural;
```

G.9. Controller testbench

To check whether the controller works as intended, a testbench was made. The simulation of this testbench can be observed in Figure G.4.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5
6  entity controller_tb is
7  end entity controller_tb;
8
9
10 architecture structural of controller_tb is
11
12         component timebase is
13                 port (  clk                     : in    std_logic;
14                         reset                   : in    std_logic;
15
16                         count_out               : out   std_logic_vector (19 downto 0)
17                 );
18         end component timebase;
19
20
21
22         component motorcontrol is
23                 port (  clk                     : in    std_logic;
24                         reset                   : in    std_logic;
25                         direction               : in    std_logic;
26                         count_in                : in    std_logic_vector (19 downto 0);
27
28                         pwm                     : out   std_logic
29                 );
30         end component motorcontrol;
31
32
33
34         component controller is
35                 port (  clk                     : in    std_logic;
36                         reset                   : in    std_logic;
37
38                         sensor_l                : in    std_logic;
39                         sensor_m                : in    std_logic;
40                         sensor_r                : in    std_logic;
41
42                         count_in                : in    std_logic_vector (19 downto 0);
43                         count_reset             : out   std_logic;
44
45                         motor_l_reset           : out   std_logic;
46                         motor_l_direction       : out   std_logic;
47
48                         motor_r_reset           : out   std_logic;
49                         motor_r_direction       : out   std_logic
50                 );
51         end component controller;
52
53
54 signal clk, reset, sensor_l_in, sensor_m_in, sensor_r_in: std_logic;
55 signal count : std_logic_vector(19 downto 0);
56 signal sensors : std_logic_vector(2 downto 0);
57 signal motor_RD, motor_LD, motor_RR, motor_LR, count_RS : std_logic;
58
59
60
61 begin
62 LB1: controller port map(       clk                     => clk,
63                                 reset                   => reset,
64                                 sensor_l                => sensor_l_in,
65                                 sensor_m                => sensor_m_in,
66                                 sensor_r                => sensor_r_in,
```

```
67                                  count_in                => count,
68                                  count_reset             => count_RS,
69                                  motor_l_direction       => motor_LD,
70                                  motor_r_direction       => motor_RD,
71                                  motor_l_reset           => motor_LR,
72                                  motor_r_reset           => motor_RR);
73
74
75  LB2: timebase port map ( clk => clk , reset => count_RS , count_out => count);
76
77
78
79          clk                     <=      '0' after 0 ns,
80                                          '1' after 10 ns when clk /= '1' else '0' after 10 ns;
81
82          reset                   <=      '1' after 0 ns,
83                                          '0' after 20 ns,
84                                          '1' after 20000000 ns,
85                                          '0' after 20000020 ns,
86                                          '1' after 40000000 ns,
87                                          '0' after 40000020 ns;
88
89          sensors                 <=      "000" after 0 ms,
90                                          "001" after 70 ms,
91                                          "010" after 110 ms,
92                                          "011" after 150 ms,
93                                          "100" after 190 ms,
94                                          "101" after 230 ms,
95                                          "110" after 280 ms,
96                                          "111" after 330 ms;
97
98          sensor_l_in             <=      sensors(2);
99          sensor_m_in             <=      sensors(1);
100         sensor_r_in             <=      sensors(0);
101
102
103 end architecture structural;
```

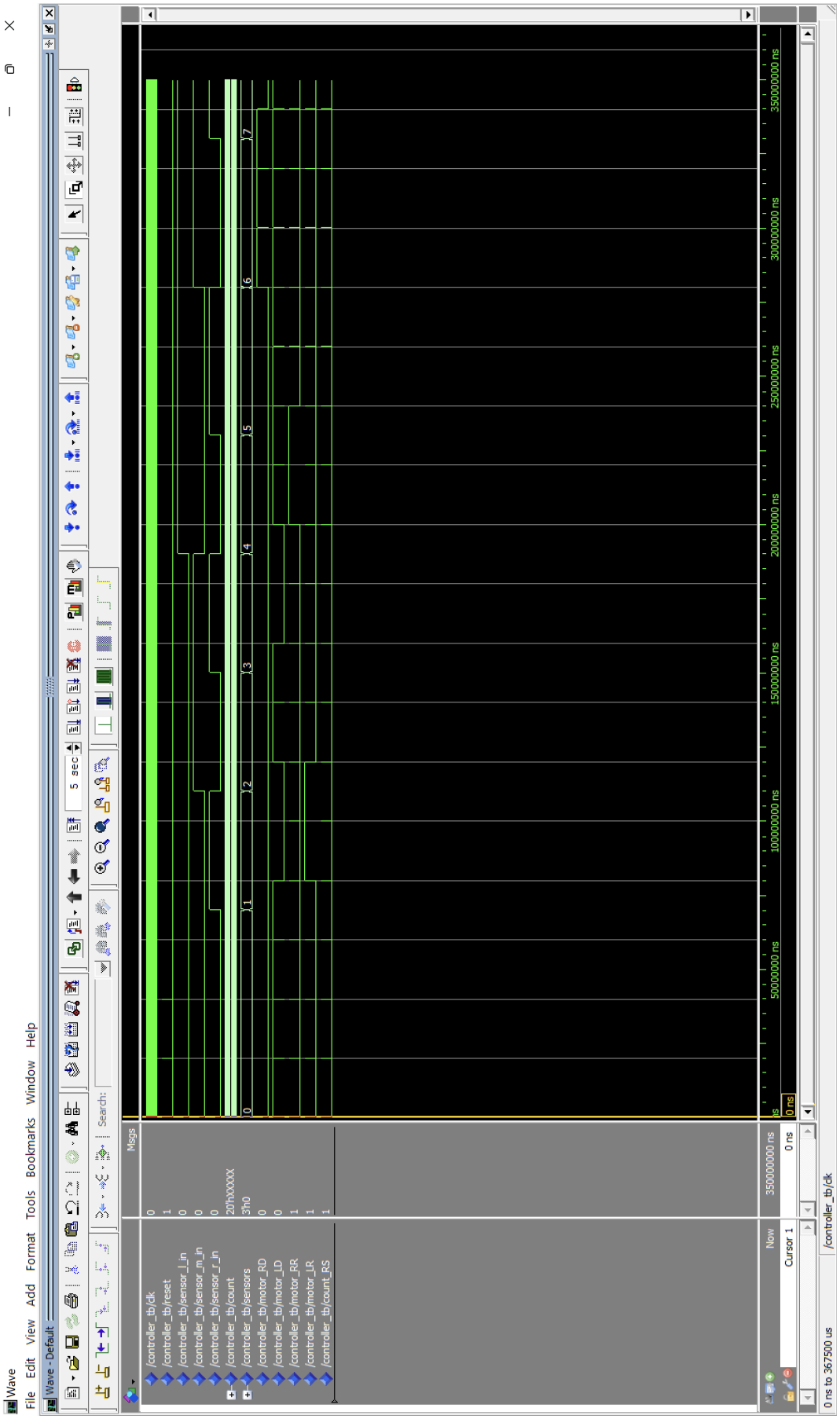

Figure G.4: Testbench of the controller

G.10. Robot

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity robot is
5          port (  clk             : in    std_logic;
6                  reset           : in    std_logic;
7
8                  sensor_l_in     : in    std_logic;
9                  sensor_m_in     : in    std_logic;
10                 sensor_r_in     : in    std_logic;
11
12                 motor_l_pwm     : out   std_logic;
13                 motor_r_pwm     : out   std_logic
14         );
15 end entity robot;
16
17 architecture structural of robot is
18
19 component inputbuffer is
20         port (  clk             : in    std_logic;
21
22                 sensor_l_in     : in    std_logic;
23                 sensor_m_in     : in    std_logic;
24                 sensor_r_in     : in    std_logic;
25
26                 sensor_l_out    : out   std_logic;
27                 sensor_m_out    : out   std_logic;
28                 sensor_r_out    : out   std_logic
29         );
30 end component inputbuffer;
31
32 component controller is
33         port (  clk                     : in    std_logic;
34                 reset                   : in    std_logic;
35
36                 sensor_l                : in    std_logic;
37                 sensor_m                : in    std_logic;
38                 sensor_r                : in    std_logic;
39
40                 count_in                : in    std_logic_vector (19 downto 0);
41                 count_reset             : out   std_logic;
42
43                 motor_l_reset           : out   std_logic;
44                 motor_l_direction       : out   std_logic;
45
46                 motor_r_reset           : out   std_logic;
47                 motor_r_direction       : out   std_logic
48         );
49
50 end component controller;
51
52
53 component timebase is
54         port (  clk             : in std_logic;
55                 reset           : in std_logic;
56                 count_out       : out std_logic_vector (19 downto 0)
57         ) ;
58  end component timebase ;
59
60
61 component motorcontrol is
62         port (  clk             : in    std_logic;
63                 reset           : in    std_logic;
64                 direction       : in    std_logic;
65                 count_in        : in    std_logic_vector (19 downto 0);
66
67                 pwm             : out   std_logic
68         );
69 end component motorcontrol;
```

```
70
71
72
73  signal sensor_l, sensor_m, sensor_r, count_rs, motor_l_rs, motor_l_d, motor_r_rs, motor_r_d: std_logic;
74  signal count: std_logic_vector(19 downto 0);
75
76  begin
77  LB1: inputbuffer port map(      clk             =>      clk,
78
79                                  sensor_l_in     =>      sensor_l_in,
80                                  sensor_m_in     =>      sensor_m_in,
81                                  sensor_r_in     =>      sensor_r_in,
82
83                                  sensor_l_out    =>      sensor_l,
84                                  sensor_m_out    =>      sensor_m,
85                                  sensor_r_out    =>      sensor_r);
86
87
88
89  LB2: controller port map(       clk                     =>      clk,
90                                  reset                   =>      reset,
91
92                                  sensor_l                =>      sensor_l,
93                                  sensor_m                =>      sensor_m,
94                                  sensor_r                =>      sensor_r,
95
96                                  count_in                =>      count,
97
98                                  count_reset             =>      count_rs,
99
100                                 motor_l_reset           =>      motor_l_rs,
101                                 motor_l_direction       =>      motor_l_d,
102
103                                 motor_r_reset           =>      motor_r_rs,
104                                 motor_r_direction       =>      motor_r_d);
105
106
107 LB3: timebase port map(         clk             =>      clk,
108                                 reset           =>      count_rs,
109                                 count_out       =>      count);
110
111
112
113
114 MCL: motorcontrol port map(     clk             =>      clk,
115                                 reset           =>      motor_l_rs,
116                                 direction       =>      motor_l_d,
117                                 count_in        =>      count,
118
119                                 pwm             =>      motor_l_pwm);
120
121
122
123 MCR: motorcontrol port map(     clk             =>      clk,
124                                 reset           =>      motor_r_rs,
125                                 direction       =>      motor_r_d,
126                                 count_in        =>      count,
127
128                                 pwm             =>      motor_r_pwm);
129
130
131 end architecture structural;
```

G.11. Robot testbench

To check whether the robot top entity works as intended, a testbench was made. The simulation of this testbench can be observed in Figure G.5.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity robot_tb is
5  end entity robot_tb;
6
7  architecture structural of robot_tb is
8
9          component robot is
10                 port (  clk             : in    std_logic;
11                         reset           : in    std_logic;
12
13                         sensor_l_in     : in    std_logic;
14                         sensor_m_in     : in    std_logic;
15                         sensor_r_in     : in    std_logic;
16
17                         motor_l_pwm     : out   std_logic;
18                         motor_r_pwm     : out   std_logic
19                 );
20         end component robot;
21
22         signal  clk, reset                      : std_logic;
23         signal  sensor_l, sensor_m, sensor_r    : std_logic;
24         signal  sensors                         : std_logic_vector(2 downto 0);
25         signal  motor_l_pwm, motor_r_pwm        : std_logic;
26
27 begin
28
29         lbl0: robot port map    (       clk                     => clk,
30                                         reset                   => reset,
31                                         sensor_l_in             => sensor_l,
32                                         sensor_m_in             => sensor_m,
33                                         sensor_r_in             => sensor_r,
34                                         motor_l_pwm             => motor_l_pwm,
35                                         motor_r_pwm             => motor_r_pwm
36                                 );
37
38         -- 20 ns = 50 MHz
39         clk                     <=      '0' after 0 ns,
40                                         '1' after 10 ns when clk /= '1' else '0' after 10 ns;
41
42         reset                   <=      '1' after 0 ns,
43                                         '0' after 40 ms;
44
45         sensors                 <=      "000" after 0 ns,   -- bbb
46                                         "001" after 70 ms,  -- bbw
47                                         "010" after 110 ms, -- bwb
48                                         "011" after 150 ms, -- bww
49                                         "100" after 190 ms, -- wbb
50                                         "101" after 230 ms, -- wbw
51                                         "110" after 270 ms, -- wwb
52                                         "111" after 310 ms; -- www
53
54         sensor_l                <=      sensors(2);
55         sensor_m                <=      sensors(1);
56         sensor_r                <=      sensors(0);
57
58 end architecture structural;
```

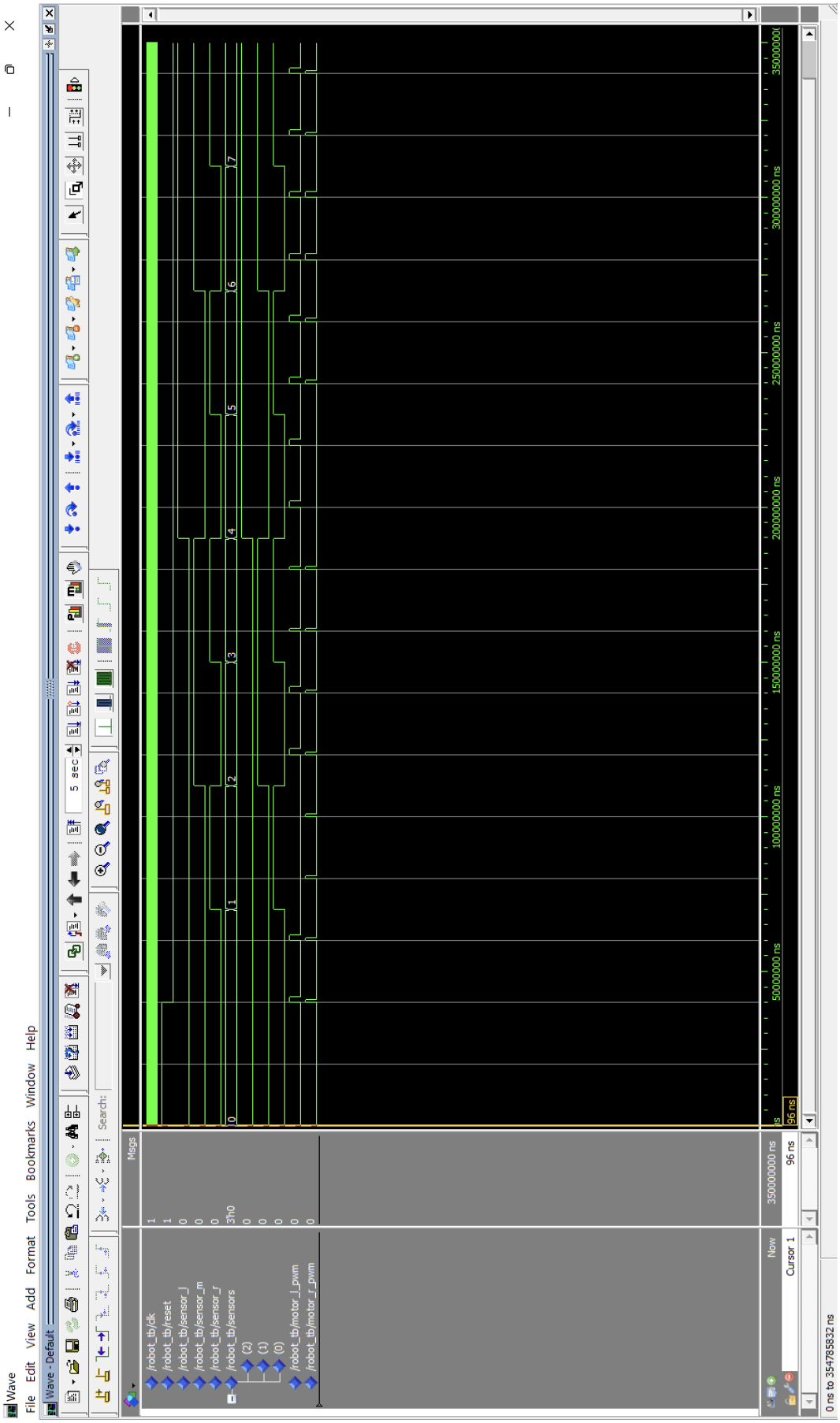

Figure G.5: Testbench of the robot

H

VHDL codes for the EPO2 Robot

H.1. Time base of the robot

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity timebase is
6          port (  clk             : in std_logic;
7                  reset           : in std_logic;
8                  count_out       : out std_logic_vector (19 downto 0);
9
10                 --New periodic reset
11                 reset_out       : out std_logic
12         ) ;
13  end entity timebase ;
14
15  architecture behavioural of timebase is
16
17         signal count, new_count : unsigned (19 downto 0);
18
19  begin
20         process ( clk )
21         begin
22                 if ( clk'event and clk ='1' ) then
23                         if ( (reset = '1') or (unsigned(count) >= to_unsigned(1000000, 20))) then
   -- or (unsigned(count) >= to_unsigned(1000000, 20) added
24                                 count <= ( others => '0');
25                                 reset_out <= '1';       -- Added reset_out signal.
26                         else
27                                 count <= new_count;
28                                 reset_out <= '0';
29                         end if;
30                 end if;
31         end process;
32
33
34         process ( count )
35         begin
36
37                 new_count <= count + 1;
38
39         end process;
40
41         count_out <= std_logic_vector ( count );
42  end architecture behavioural;
```

H.2. Motor controller of the robot

```
1
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4  use IEEE.numeric_std.all;
5
6  -- Please add necessary libraries:
7
8  entity motorcontrol is
9          port (  clk             : in    std_logic;
10                 reset           : in    std_logic;
11                 direction       : in    std_logic;
12                 count_in        : in    std_logic_vector (19 downto 0);
13
14                 reset_in        : in    std_logic;
15                 -- input signal that comes from the timebase reset_out output signal.
16                 pwm             : out   std_logic
17         );
18 end entity motorcontrol;
19
20 architecture behavioural of motorcontrol is
21
22         type motorcontrol_state is (state0, state1, state2);
23
24         signal state, new_state: motorcontrol_state;
25
26 begin
27
28         process (clk)
29         begin
30                 if (rising_edge (clk)) then
31                         if (reset_in = '1') then        -- added (reset_in = '1')
32                                 state <= state0;
33                         else
34                                 state <= new_state;
35                         end if;
36                 end if;
37         end process;
38
39
40         process (direction, count_in, state, reset)
41         begin
42                 case state is
43
44                         when state0 => pwm <= '0';
45                                 if (reset = '0') then
46                                         new_state <= state1;
47                                 else
48                                         new_state <= state0;
49                                 end if;
50
51
52
53                         when state1 => pwm <= '1';
54                                 if (direction ='0' and unsigned(count_in) >= to_unsigned(50000, 20)) then
55                                         new_state <= state2;
56
57                                 elsif (direction ='1' and unsigned(count_in) >= to_unsigned(100000, 20))
58                                         then new_state <= state2;
59
60                                 else
61                                         new_state <= state1;
62                                 end if;
63
64
65
66                         when state2 => pwm <='0';
67                                 new_state <= state2;
68
69
```

```
70                  end case;
71          end process;
72  end architecture behavioural;
```

H.3. Controller of the robot

H.3.1. Final state machines of the controller

In figure H.1, H.2 and H.3, the final state machine of the cross counter process, communication process and the control motor process is provided respectively.

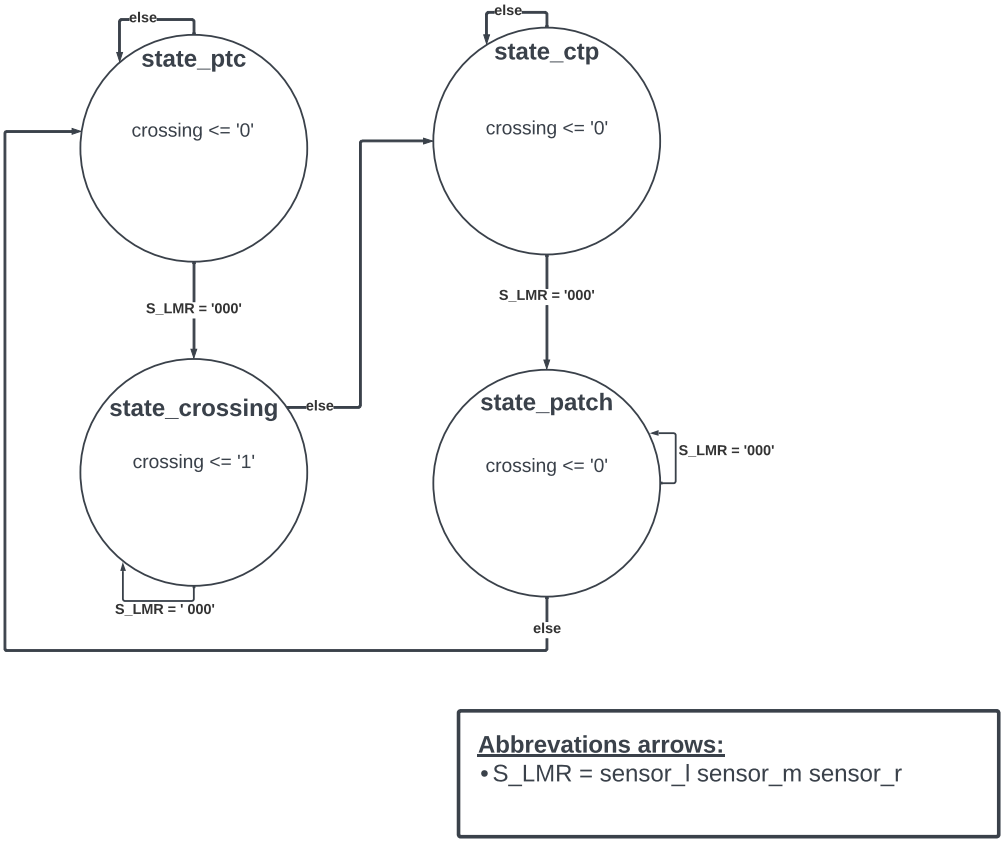

Figure H.1: final state machine crosscounter process

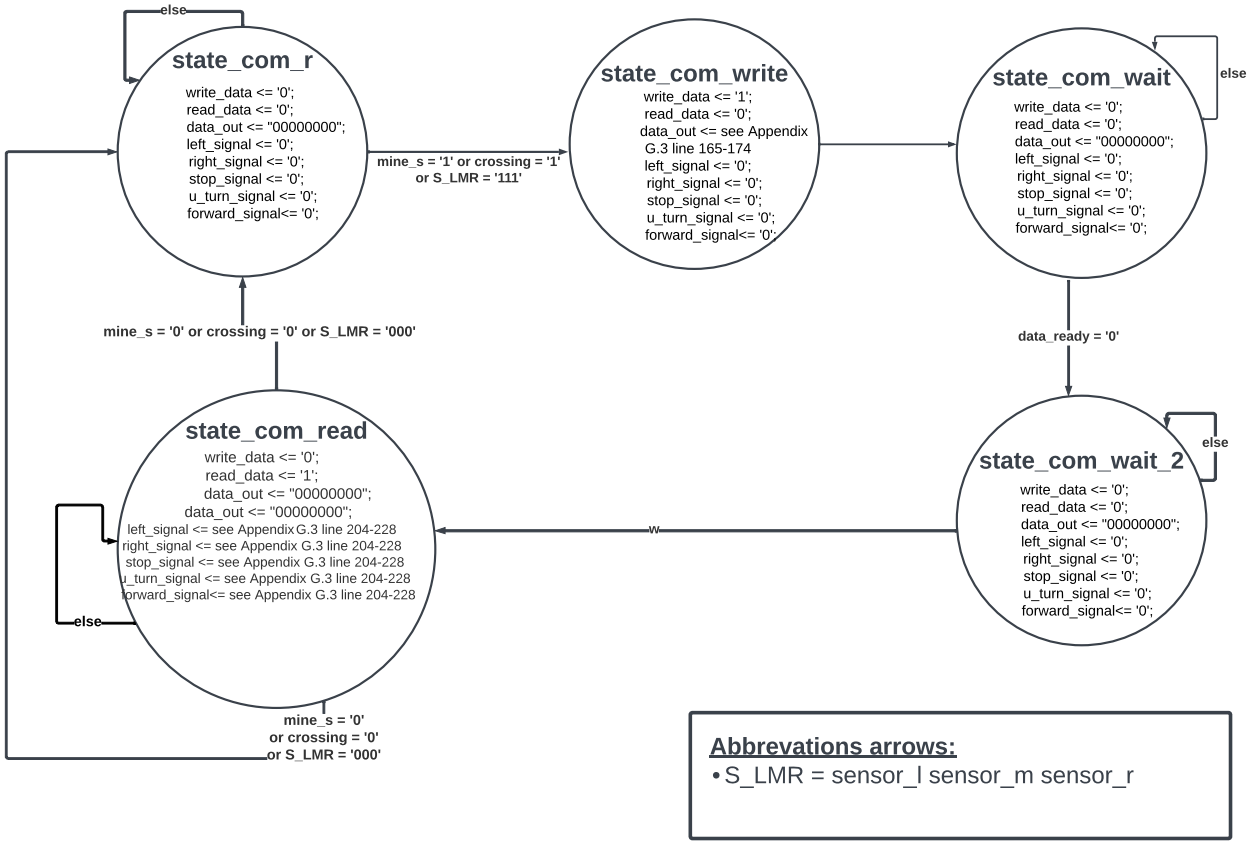

Figure H.2: final state machine communication process

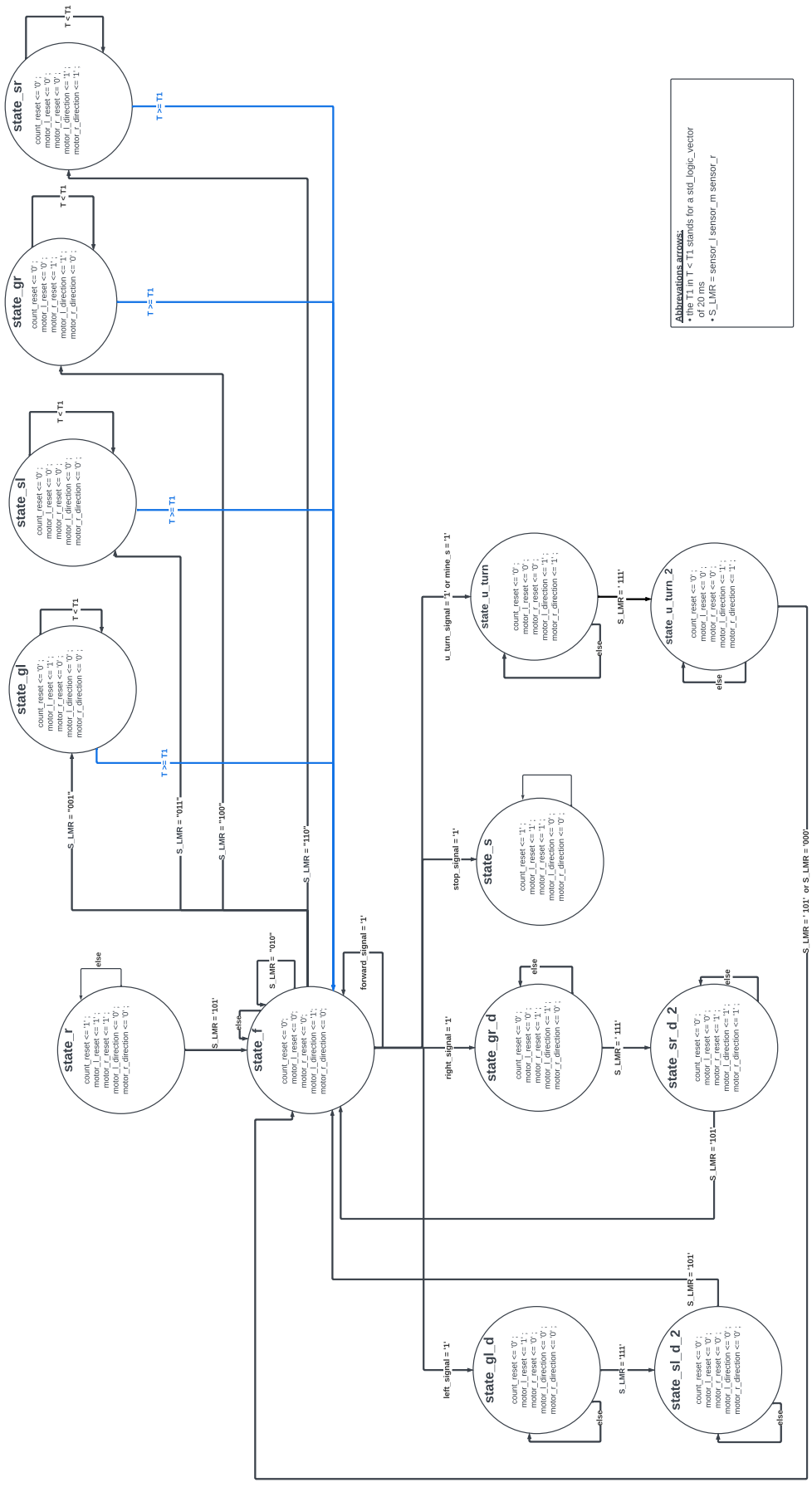

Figure H.3: final state machine control motor process

H.3.2. VHDL code of the controller

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  -- Please add necessary libraries:
5
6
7  entity controller is
8          port (  clk                     : in    std_logic;
9                  reset                   : in    std_logic;
10
11                 sensor_l                : in    std_logic;
12                 sensor_m                : in    std_logic;
13                 sensor_r                : in    std_logic;
14
15                 count_in                : in    std_logic_vector (19 downto 0);
16
17 --new inputs
18                 data_in                 : in    std_logic_vector (7 downto 0);
19                 data_ready              : in    std_logic;
20                 mine_s                  : in    std_logic;
21
22 --new outputs
23                 data_out                : out   std_logic_vector (7 downto 0);
24                 write_data              : out   std_logic;
25                 read_data               : out   std_logic;
26
27                 count_reset             : out   std_logic;
28
29                 motor_l_reset           : out   std_logic;
30                 motor_l_direction       : out   std_logic;
31
32                 motor_r_reset           : out   std_logic;
33                 motor_r_direction       : out   std_logic
34
35
36         );
37
38 end entity controller;
39
40 architecture behavioural of controller is
41
42         type controller_state is        (state_r,
43                                          state_s,
44                                          state_gl_d, state_sl_d_2,
45                                          state_gr_d, state_sr_d_2,
46                                          state_f, state_gl, state_sl, state_gr, state_sr,
47                                          state_u_turn, state_u_turn_2
48 );
49
50         type crossing_state is          (state_ptc, --state patch to crossing
51                                          state_crossing, --state when on crossing
52                                          state_ctp,  --state crossing to patch
53                                          state_patch    --state when on patch
54
55 );
56
57         type communication_state is     (state_com_r,
58                                          state_com_write,
59                                          state_com_wait, state_com_wait_2,
60                                          state_com_read
61 );
62
63
64         signal state, new_state: controller_state;
65         signal state_p, new_state_p: crossing_state;
66         signal state_com, new_state_com: communication_state;
67         signal crossing: std_logic;
68         signal left_signal, right_signal, stop_signal, forward_signal, u_turn_signal: std_logic;
69         signal count_turn, new_count_turn : unsigned (27 downto 0);
70
```

```
71  begin
72          process (clk)
73          begin
74                  if (rising_edge (clk)) then
75                          if (reset = '1') then
76                                  state   <= state_r;
77                                  state_p <= state_ptc;
78                                  state_com <= state_com_r;
79
80                          else
81                                  state   <= new_state;
82                                  state_p <= new_state_p;
83                                  state_com <= new_state_com;
84                          end if;
85                  end if;
86          end process;
87
88  --FSM voor cross counting
89          process (sensor_l, sensor_m, sensor_r, state_p)
90          begin
91                  case state_p is
92
93                          when state_ptc =>       crossing <= '0';
94
95                                          if (sensor_l = '0' and sensor_m = '0' and sensor_r = '0') then
96                                                  new_state_p     <=      state_crossing;
97                                          else
98                                                  new_state_p     <=      state_ptc;
99                                          end if;
100
101                         when state_crossing =>  crossing <= '1';
102
103                                         if (sensor_l = '0' and sensor_m = '0' and sensor_r = '0') then
104                                                 new_state_p     <=      state_crossing;
105                                         else
106                                                 new_state_p     <=      state_ctp;
107                                         end if;
108
109                         when state_ctp =>       crossing <= '0';
110
111                                         if (sensor_l = '0' and sensor_m = '0' and sensor_r = '0') then
112                                                 new_state_p     <=      state_patch;
113                                         else
114
115                                                 new_state_p     <=      state_ctp;
116                                         end if;
117
118                         when state_patch =>     crossing <= '0';
119
120                                         if (sensor_l = '0' and sensor_m = '0' and sensor_r = '0') then
121                                                 new_state_p     <=      state_patch;
122                                         else
123                                                 new_state_p     <=      state_ptc;
124                                         end if;
125                 end case;
126         end process;
127 --Eind FSM crosscounting
128
129
130
131 --Process for communication
132         process (state_com, mine_s, crossing, sensor_l, sensor_m, sensor_r, data_ready, data_in, count_turn)
133         begin
134                 left_signal <= '0';
135                 right_signal <= '0';
136                 stop_signal <= '0';
137                 u_turn_signal <= '0';
138                 forward_signal<= '0';
139
140                 case state_com is
141
```

```
142                         when state_com_r =>
143                                         read_data <= '0';
144                                         data_out <= "00000000";
145                                         write_data <= '0';
146
147                                 if (mine_s = '1') then
148                                         new_state_com <= state_com_write;
149
150                                 elsif (crossing = '1') then
151                                         new_state_com <= state_com_write;
152
153                                 elsif ((sensor_l = '1' and sensor_m = '1' and sensor_r = '1')) then
154                                         new_state_com <= state_com_write;
155
156                                 else
157                                         new_state_com <= state_com_r;
158                                 end if;
159
160
161                         when state_com_write =>
162                                                 read_data <= '0';
163                                                 write_data <= '1';
164                                                 new_state_com <= state_com_wait;
165
166                                                 if (mine_s = '1') then
167                                                         data_out <= "00000100";  --Mine detected
168                                                 elsif (crossing = '1') then
169                                                         data_out <= "00000010";  -- at crossing
170                                                 elsif (sensor_l = '0' and sensor_m = '0' and sensor_r = '0')
171                                                         then data_out <= "00000011"; -- at dead end
172                                                 else
173                                                         data_out <= "00000000";
174                                                 end if;
175
176                         when state_com_wait     =>
177                                                         write_data <= '0';
178                                                         read_data <= '0';
179
180                                                         data_out <= "00000000";
181
182                                                 if (data_ready = '0') then
183                                                         new_state_com <= state_com_wait_2;
184                                                 else
185                                                         new_state_com <= state_com_wait;
186                                                 end if;
187
188                         when state_com_wait_2   =>
189                                                         write_data <= '0';
190                                                         read_data <= '0';
191
192                                                         data_out <= "00000000";
193
194                                                 if ((data_ready = '1')) then
195                                                         new_state_com <= state_com_read;
196                                                 else
197                                                         new_state_com <= state_com_wait_2;
198                                                 end if;
199
200                         when state_com_read     =>
201                                                         write_data <= '0';
202                                                         data_out <= "00000000";
203                                                         read_data<= '1';
204
205                                                 if (data_in = "00000001") then
206                                                         left_signal <= '1';
207
208                                                 elsif (data_in = "00000010") then
209                                                         right_signal <= '1';
210
211                                                 elsif (data_in = "00000011") then
212                                                         forward_signal<= '1';
```

```
213
214
215                                             elsif (data_in = "00000100") then
216                                                     stop_signal <= '1';
217
218
219                                             elsif (data_in = "00000101") then
220                                                     u_turn_signal <= '1';
221
222                                             else
223                                                     left_signal <= '0';
224                                                     right_signal <= '0';
225                                                     stop_signal <= '0';
226                                                     u_turn_signal <= '0';
227                                                     forward_signal<= '0';
228                                             end if;
229
230                                             if (mine_s = '0' or crossing = '0' or (sensor_l = '1' and
231                                     sensor_m = '0' and sensor_r = '1')) then
232                                                     new_state_com <= state_com_r;
233                                             else
234                                                     new_state_com <= state_com_read;
235                                             end if;
236                     end case;
237             end process;
238
239     --End process for communication
240
241
242     --Process for motors
243             process (sensor_l, sensor_m, sensor_r, count_in, state, mine_s,
244                     left_signal, right_signal, stop_signal, forward_signal, u_turn_signal, count_turn)
245             begin
246                     case state is
247
248                     when state_r =>         count_reset <= '1';
249                                             motor_l_reset <= '1';
250                                             motor_r_reset <= '1';
251
252                                             motor_l_direction <= '0';
253                                             motor_r_direction <= '0';
254
255     -- this part makes the robot start working without initializing signal coming in from uart.
256                                     if (sensor_l = '1' and sensor_m = '0' and sensor_r = '1') then
257                                             new_state <= state_f;
258
259                                     else
260                                             new_state <= state_r;
261                                     end if;
262
263                     when state_f =>
264                                             count_reset <= '0';
265                                             motor_l_reset <= '0';
266                                             motor_r_reset <= '0';
267
268                                             motor_l_direction <= '1';
269                                             motor_r_direction <= '0';
270
271                                     if (left_signal = '1') then
272                                             new_state <= state_gl_d;
273
274                                     elsif (right_signal = '1') then
275                                             new_state <= state_gr_d;
276
277                                     elsif (stop_signal = '1') then
278                                             new_state <= state_s;
279
280                                     elsif (forward_signal = '1') then
281                                             new_state <= state_f;
282
283                                     elsif (u_turn_signal = '1' or mine_s = '1') then
```

```
284                                     new_state <= state_u_turn;
285
286                             elsif (sensor_l = '0' and sensor_m = '0' and sensor_r = '1') then
287                                     new_state <= state_gl;
288
289
290                             elsif (sensor_l = '0' and sensor_m = '1' and sensor_r = '0') then
291                                     new_state <= state_f;
292
293
294                             elsif (sensor_l = '0' and sensor_m = '1' and sensor_r = '1') then
295                                     new_state <= state_sl;
296
297                             elsif (sensor_l = '1' and sensor_m = '0' and sensor_r = '0') then
298                                     new_state <= state_gr;
299
300
301                             elsif (sensor_l = '1' and sensor_m = '0' and sensor_r = '1') then
302                                     new_state <= state_f;
303
304
305                             elsif (sensor_l = '1' and sensor_m = '1' and sensor_r = '0') then
306                                     new_state <= state_sr;
307
308                             else
309                                     new_state <= state_f;
310
311                             end if;
312
313
314                     when state_gl =>        count_reset <= '0';
315                                             motor_l_reset <= '1';
316                                             motor_r_reset <= '0';
317
318                                             motor_l_direction <= '0';
319                                             motor_r_direction <= '0';
320
321                             if (unsigned(count_in) < to_unsigned(1000000, 20)) then
322                                     new_state <= state_gl;
323
324                             elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
325                                     new_state <= state_f;
326                             end if;
327
328
329                     when state_sl =>        count_reset <= '0';
330                                             motor_l_reset <= '0';
331                                             motor_r_reset <= '0';
332
333                                             motor_l_direction <= '0';
334                                             motor_r_direction <= '0';
335
336                             if (unsigned(count_in) < to_unsigned(1000000, 20)) then
337                                     new_state <= state_sl;
338
339                             elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
340                                     new_state <= state_f;
341
342                             end if;
343
344
345                     when state_gr =>        count_reset <= '0';
346                                             motor_l_reset <= '0';
347                                             motor_r_reset <= '1';
348
349                                             motor_l_direction <= '1';
350                                             motor_r_direction <= '0';
351
352                             if (unsigned(count_in) < to_unsigned(1000000, 20)) then
353                                     new_state <= state_gr;
354
```

```
355                                 elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
356                                         new_state <= state_f;
357
358                                 end if;
359
360
361                 when state_sr =>        count_reset <= '0';
362                                         motor_l_reset <= '0';
363                                         motor_r_reset <= '0';
364
365                                         motor_l_direction <= '1';
366                                         motor_r_direction <= '1';
367
368                                 if (unsigned(count_in) < to_unsigned(1000000, 20)) then
369                                         new_state <= state_sr;
370
371                                 elsif (unsigned(count_in) >= to_unsigned(1000000, 20)) then
372                                         new_state <= state_f;
373
374                                 end if;
375
376
377 -- stop branch, bestaande uit state_s_write en state_s_read.
378
379                 when state_s     =>             count_reset             <= '1';
380                                                motor_l_reset           <= '1';
381                                                motor_r_reset           <= '1';
382
383                                                motor_l_direction       <= '0';
384                                                motor_r_direction       <= '0';
385
386                       new_state <= state_s;
387                 -- end state, you never leave this, because if you come here you are finished with challenge.
388
389
390 -- left branch door data van thijs, bestaande uit state_gl_d, state_gl_d_2
391 --We need to make sure the robot turns early enough so lmr = 110 when we encounter the line again.
392
393                 when state_gl_d =>      count_reset             <= '0';
394                                         motor_l_reset           <= '1';
395                                         motor_r_reset           <= '0';
396
397                                         motor_l_direction       <= '0';
398                                         motor_r_direction       <= '0';
399
400                                 if ((sensor_l = '1' and sensor_m = '1' and sensor_r = '1')) then
401                                         new_state <= state_sl_d_2;
402                                 else
403                                         new_state <= state_gl_d;
404                                 end if;
405
406                 when state_sl_d_2 =>    count_reset             <= '0';
407                                         motor_l_reset           <= '0';
408                                         motor_r_reset           <= '0';
409                                         motor_l_direction       <= '0';
410                                         motor_r_direction       <= '0';
411
412                                 if (sensor_l = '1' and sensor_m = '0' and sensor_r = '1') then
    -- go back when we see the line again
413                                         new_state <= state_f;
414                                 else
415                                         new_state <= state_sl_d_2;
416                                 end if;
417
418
419 -- right branch door data van thijs, bestaande uit state_gr_d, state_gr_d_2
420                 when state_gr_d  =>
421                                         count_reset <= '0';
422                                         motor_l_reset <= '0';
423                                         motor_r_reset <= '1';
424
```

```
425                                         motor_l_direction <= '1';
426                                         motor_r_direction <= '0';
427
428                                 if ((sensor_l = '1' and sensor_m = '1' and sensor_r = '1')) then
429                                         new_state <= state_sr_d_2;
430                                 else
431                                         new_state <= state_gr_d;
432                                 end if;
433
434
435                 when state_sr_d_2  =>
436                                         count_reset <= '0';
437                                         motor_l_reset <= '0';
438                                         motor_r_reset <= '0';
439
440                                         motor_l_direction <= '1';
441                                         motor_r_direction <= '1';
442
443                                 if (sensor_l = '1' and sensor_m = '0' and sensor_r = '1') then
-- same comment as state_gl_d_2.
444                                         new_state <= state_f;
445                                 else
446                                         new_state <= state_sr_d_2;
447                                 end if;
448
449
450 --u-turn branch implented as two sharp right states, we turn sharp right until we see lmr = 101 again.
451
452                 when state_u_turn =>    count_reset <= '0';
453                                         motor_l_reset <= '0';
454                                         motor_r_reset <= '0';
455
456                                         motor_l_direction <= '1';
457                                         motor_r_direction <= '1';
458
459                                 if (sensor_l = '1' and sensor_m = '1' and sensor_r = '1') then
460                                         new_state <= state_u_turn_2;
461                                 else
462                                         new_state <= state_u_turn;
463                                 end if;
464
465
466                 when state_u_turn_2 =>  count_reset <= '0';
467                                         motor_l_reset <= '0';
468                                         motor_r_reset <= '0';
469
470                                         motor_l_direction <= '1';
471                                         motor_r_direction <= '1';
472
473                                 if ((sensor_l = '1' and sensor_m = '0' and sensor_r = '1') or
474     (sensor_l = '0' and sensor_m = '0' and sensor_r = '0') ) then
475                                         new_state <= state_f;
476                                 else
477                                         new_state <= state_u_turn_2;
478                                 end if;
479
480                 end case;
481         end process;
482 end architecture behavioural;
```

H.3.3. EPO2 Robot top entity

```
1
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity robot is
6          port (
7
8                  clk             : in    std_logic;
9                  reset           : in    std_logic;
10
11                 sensor_l_in     : in    std_logic;
12                 sensor_m_in     : in    std_logic;
13                 sensor_r_in     : in    std_logic;
14
15                 sensor_mine_in  : in    std_logic;
16
17                 rx              : in    std_logic;
18                 tx              : out   std_logic;
19                 buffer_empty    : out   std_logic;
20
21                 motor_l_pwm     : out   std_logic;
22                 motor_r_pwm     : out   std_logic;
23
24                 --led_7 : out std_logic;
25                 --led_6 : out std_logic;
26                 --led_5 : out std_logic;
27                 --led_4 : out std_logic;
28                 --led_3 : out std_logic;
29                 --led_2 : out std_logic;
30                 --led_1 : out std_logic;
31                 led_0 : out std_logic
32
33         );
34 end entity robot;
35
36 architecture structural of robot is
37
38 component inputbuffer is
39         port (  clk             : in    std_logic;
40
41                 sensor_l_in     : in    std_logic;
42                 sensor_m_in     : in    std_logic;
43                 sensor_r_in     : in    std_logic;
44
45                 sensor_l_out    : out   std_logic;
46                 sensor_m_out    : out   std_logic;
47                 sensor_r_out    : out   std_logic
48         );
49 end component inputbuffer;
50
51 component controller is
52         port (  clk                     : in    std_logic;
53                 reset                   : in    std_logic;
54
55                 sensor_l                : in    std_logic;
56                 sensor_m                : in    std_logic;
57                 sensor_r                : in    std_logic;
58
59                 count_in                : in    std_logic_vector (19 downto 0);
60
61 --new inputs
62                 data_in                 : in    std_logic_vector (7 downto 0);
63                 data_ready              : in    std_logic;
64                 mine_s                  : in    std_logic;
65
66
67 --new outputs
68                 data_out                : out   std_logic_vector (7 downto 0);
69                 write_data              : out   std_logic;
70                 read_data               : out   std_logic;
```

```
71  --
72                  count_reset             : out   std_logic;
73
74                  motor_l_reset           : out   std_logic;
75                  motor_l_direction       : out   std_logic;
76
77                  motor_r_reset           : out   std_logic;
78                  motor_r_direction       : out   std_logic
79
80          );
81
82  end component controller;
83
84
85  component timebase is
86          port (  clk             : in std_logic;
87                  reset           : in std_logic;
88
89                  reset_out       : out std_logic;
90
91                  count_out       : out std_logic_vector (19 downto 0)
92          ) ;
93   end component timebase ;
94
95
96  component motorcontrol is
97          port (  clk             : in    std_logic;
98                  reset           : in    std_logic;
99                  direction       : in    std_logic;
100                 count_in        : in    std_logic_vector (19 downto 0);
101
102                 reset_in        : in std_logic;
103
104                 pwm             : out   std_logic
105         );
106 end component motorcontrol;
107
108 component uart is
109         port (
110         clk             : in  std_logic;
111         reset           : in  std_logic;
112
113         rx              : in  std_logic;
114         tx              : out std_logic;
115
116         data_in         : in  std_logic_vector (7 downto 0);
117         buffer_empty    : out std_logic;
118         write           : in  std_logic;
119
120         data_out        : out std_logic_vector (7 downto 0);
121         data_ready      : out std_logic;
122         read            : in  std_logic
123     );
124 end component uart;
125
126 component mine_sensor_top is
127         port (  clk             : in std_logic;
128                 mine_sensor_in  : in std_logic;
129                 mine_out        : out std_logic;
130                 led_0           : out std_logic
131         );
132 end component mine_sensor_top;
133
134 signal sensor_l, sensor_m, sensor_r, count_rs, reset_periodical, motor_l_rs, motor_l_d, motor_r_rs,
135         motor_r_d: std_logic;
136 signal count: std_logic_vector(19 downto 0);
137
138
139 -- new signals
140 signal data_uico: std_logic_vector (7 downto 0);  --uart in controller out
141 signal data_uoci: std_logic_vector (7 downto 0); -- uart out controller in
```

```
142 signal write_sig, read_sig, data_ready_sig: std_logic;
143 signal mine_signal: std_logic;
144 signal mine_out_signal: std_logic;
145
146
147 begin
148 LB1: inputbuffer port map(      clk             =>      clk,
149
150                                 sensor_l_in     =>      sensor_l_in,
151                                 sensor_m_in     =>      sensor_m_in,
152                                 sensor_r_in     =>      sensor_r_in,
153
154                                 sensor_l_out    =>      sensor_l,
155                                 sensor_m_out    =>      sensor_m,
156                                 sensor_r_out    =>      sensor_r);
157
158
159 M1: mine_sensor_top port map (  clk             =>      clk,
160                                 mine_sensor_in  =>      sensor_mine_in,
161                                 mine_out        =>      mine_out_signal,
162                                 led_0                           => led_0);
163
164
165 U1:     uart port map(
166                                         clk  =>                 clk,
167                                         reset =>                reset,
168
169                                         rx =>                   rx,
170                                         tx =>                   tx,
171
172                                         data_in =>              data_uico,
173                                         buffer_empty =>         buffer_empty,
174                                         write =>                write_sig,
175
176                                         data_out =>             data_uoci,
177                                         data_ready =>           data_ready_sig,
178                                         read =>                 read_sig
179
180 );
181
182 LB2: controller port map(
183                                 clk                     =>      clk,
184                                 reset                   =>      reset,
185
186                                 sensor_l                =>      sensor_l,
187                                 sensor_m                =>      sensor_m,
188                                 sensor_r                =>      sensor_r,
189
190                                 count_in                =>      count,
191
192 --new inputs
193
194                                 --data_in(7)    =>      led_7,
195                                 --data_in(6)    =>      led_6,
196                                 --data_in(5)    =>      led_5,
197                                 --data_in(4)    =>      led_4,
198                                 --data_in(3)    =>      led_3,
199                                 --data_in(2)    =>      led_2,
200                                 --data_in(1)    =>      led_1,
201                                 --data_in(0)    =>      led_0,
202
203                                 data_in         => data_uoci,
204
205                                 data_ready              =>      data_ready_sig,
206
207                                 mine_s                  =>      mine_out_signal,
208
209
210 --new outputs
211
212                                 --data_out(7)   =>      led_7,
```

```
213                                 --data_out(6)    =>      led_6,
214                                 --data_out(5)    =>      led_5,
215                                 --data_out(4)    =>      led_4,
216                                 --data_out(3)    =>      led_3,
217                                 --data_out(2)    =>      led_2,
218                                 --data_out(1)    =>      led_1,
219                                 --data_out(0)    =>      led_0,
220
221                                 data_out                 =>      data_uico,
222                                 write_data               =>      write_sig,
223                                 read_data                =>      read_sig,
224 --
225
226                                 count_reset              =>      count_rs,
227
228                                 motor_l_reset            =>      motor_l_rs,
229                                 motor_l_direction        =>      motor_l_d,
230
231                                 motor_r_reset            =>      motor_r_rs,
232                                 motor_r_direction        =>      motor_r_d
233                                 );
234
235
236 T1: timebase port map(          clk              =>      clk,
237                                 reset            =>      count_rs,
238                                 count_out        =>      count,
239                                 reset_out        =>      reset_periodical
240 );
241
242 MCL: motorcontrol port map(     clk              =>      clk,
243                                 reset            =>      motor_l_rs,
244                                 direction        =>      motor_l_d,
245                                 count_in         =>      count,
246
247                                 reset_in         =>      reset_periodical,
248
249                                 pwm              =>      motor_l_pwm
250                         );
251
252
253
254 MCR: motorcontrol port map(     clk              =>      clk,
255                                 reset            =>      motor_r_rs,
256                                 direction        =>      motor_r_d,
257                                 count_in         =>      count,
258
259                                 reset_in         =>      reset_periodical,
260
261                                 pwm              =>      motor_r_pwm
262                 );
263
264
265 end architecture structural;
```

Project plan

I.1. Introduction & background

The purpose of the project is to develop hardware and software modules to implement a self-operating robot. For this project, we are divided into groups of nine students. Each of the groups is split up into several groups with various tasks.
The main objective of the robot is to enable it to drive, in the least amount of time possible, through a given route and avoid mines placed along the route. An overview of the parts will be shown in the next chapter.
We have 3 challenges A, B, and C that need to be completed, these challenges are described in chapter two of this project plan.

In order to make the robot the group is divided into subgroups with the following tasks below.

- Mine Detector: two students will work on building a mine-detecting sensor, that will be able to detect mines on a distance.
- C-Code: two students will work on creating a program with an interface and an algorithm for computing the route.
- Serial Communication: two students will work on creating an interface that can guide the robot via the PC.
- VHDL: three students will work on the FSM to perform the line-following and detecting the mines.

The precise tasks and objectives of the individual subgroups will also be shown more in-depth in chapter two. It is common knowledge that every project has its limits. Whether they come from within the group or from the outside, it is always good practice to mention and thus be aware of them. Chapter three will cover some of these limits as well as the task distribution of the group. Furthermore, chapter four will describe the tasks of each subgroup each week. This schedule will be used as precisely as possible.

Lastly, chapter five will cover some risks that might be troublesome for the project. The difference between a limit, as mentioned in chapter three, and a risk is its manageability. Risks can be mitigated while limits are given. Thus chapter five will also propose solutions/precautions to the mentioned risks.

The project will last eight weeks and in the ninth week, there will be oral exams. We have been taught background information in the third semester of the study which we should apply in this project, however, we will learn more about the obstacles, lessons, and details involved with creating the hardware and software modules to implement a self-operating robot.

In the end, we will perform the three challenges and compete against other teams. If all our sub-systems will work correctly we should successfully complete these challenges and perhaps become #1.

I.2. Project Results

I.2.1. Physical objective

The objective of this project is to construct an autonomous robot based on the line follower project from the labs of the Digital Systems B course. The robot has to not only utilize IR sensors to follow the line but also be able to pass the challenges outlined below while navigating through the map (sometimes referred to as "the maze"), as described in Chapter 1 of the project manual.
The routing algorithm is executed on the computer that communicates with the robot through UART and XBee wireless modules, which completes the entire autonomous system.

- In the first challenge the robot has to visit a series of stations in the given order.
- The second challenge consists of the first one and the addition of "mines" which are metal disks that have to be avoided by the robot.
- During the last challenge more mines will be placed in the maze and the robot has to find them all. In stage 2 a new mine will be added and the robot has to find it also.

The structure of the robot is shown in Figure 1.2 of the project manual and summarized here in the figure below.

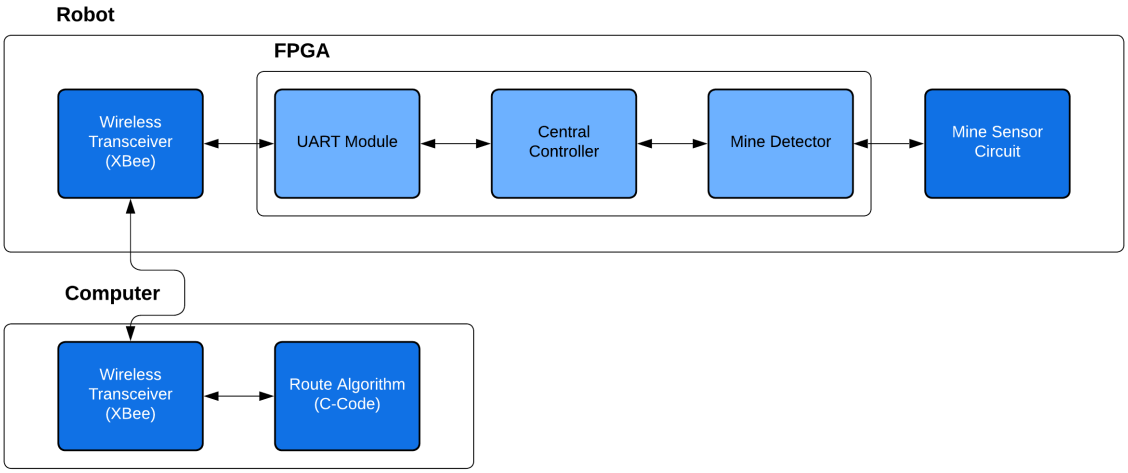

Figure I.1: Block diagram of the entire system.

The workload was split into the following parts that correspond to every subgroup in the project:

- VHDL: The line follower's FSM from the lab sessions of Digital Systems B needs to be extended in order to complete the functionality of the robot as it needs to detect a mine for example. After the FSM has been designed it needs to be written in VHDL.
- Serial Communication: The main objective of this section is to create an interface that can guide the robot via the PC. This will be done using the XBee modules that communicate with each other with the ZigBee protocol and with the external hardware via a UART protocol. To achieve this we will write a small application in C (programming language) for the laptop and a corresponding VHDL program for the robot.
- C-code: The objective for the smart robot is to create a separate program with a simple user interface. This program will be implemented into a larger program that will control the robot. We will describe an algorithm that can be used to compute the route through the maze and also a data structure to implement this algorithm.
- Mine detector: One of the important aspects of the smart robot is the ability to detect mines in their path when going from position A to position B. The robot must detect the mines and avoid them and thus redetermine a new path. In order to detect them a mine detector sensor must be built. This sensor must also be able to roughly determine the mine distance to the robot. It will be implemented using an LC oscillator.

I.2.2. Learning objectives

Besides creating a working autonomous robot system, the goal is to master certain learning objectives of the EPO project.
After completion of the project, we will be able to apply the theoretical concepts of the first year in practice. Another important aspect is that we will be able to methodologically design an electronic system utilizing a project-based team approach. At last, we will learn and practice critical thinking and the ability to find information in relevant sources such as datasheets and manuals, as well as critically assess and utilize those pieces of information.

I.3. Project activities

In order to achieve the project's results, the group must divide the tasks evenly and communicate with each other clearly. A detailed description of the workload distribution with a timeline can be found in the next chapter.

Regarding communication and file management, the group will use a shared Google Drive, WhatsApp, and GIT. In this chapter, we will briefly talk about what each group member will be responsible for and the roles of the meetings of the team. Furthermore, we will go over some of our constraints in the second part of this chapter.

I.3.1. Task distribution

As mentioned, efficient task distribution is key in order for the project to end with success. Below in table I.1 the subgroups of our project can be found.

Sensors	Darren van de Pol, Wiktor Tomanek
VHDL	Chantal Chen, Eojin Lim, Wilson Rong
C code and Serial Communication	Thijs van Esch, Maximiliaan Sobry
Serial Communication	Kevin Pang, Pieter Olyslaegers

Table I.1: Subgroups

I.3.2. Project limits

Every project has its limits, and this one is no exception. Some of these limits are the physical quality of the robot, time limit, our ability to work together, our limited knowledge about robots, no beforehand advanced experience with robots, circuitry, etc. Below we will elaborate on some of the major limits of our project.

Physical quality of the robot
This one is quite easy to explain. We have a limited amount of money so the components (capacitors, inductors, resistors, batteries, PCBs, etc.) will not be of any accuracy, which means the physical quality of the robot we are building will have some limitations. Nevertheless, we can maximize the quality of our robot system by implementing our knowledge of electrical engineering as best as possible.

Time limit
Time is also a factor when we are talking about the limits of our system. The more time we have the better we can adjust, troubleshoot and solve problems. This means that we have to manage our time very efficiently in order to maximize the possible outcome.

Limited knowledge
The knowledge of the group is also limited. There is still a lot of it to acquire for the group members and this project is one way to do that. Not every member of the group may keep up with the peace required, therefore, resulting in a delay.

Surely, there are more project limits, but these are the most important. It is therefore crucial to minimize the negative effects of these limits on our project whenever we can.

I.4. Organization & Schedule

For a group to work as efficiently as possible, there has to be a concrete plan to work toward the end goal. For this project, we have to work in four subgroups with each its own tasks. In this chapter, we will discuss what and when each group has to complete in order to fulfill the requirements for this project.

I.4.1. Plan

In the table Schedule are all of the tasks described to whom or what subgroup has to do within a certain amount of time, which can deviate, from the corresponding deadline.

I.4.2. Schedule

In the table, the letters E, S, C, SC, and V stand for Everyone, Sensors, C-code, Serial communication, and VHDL respectively.

Week	Subgroup	Task
1	E	Task distribution, reading the manual and starting the project plan.
1	S	Sketching a general design of the mine sensor sub-circuit.
1	V	Study manual and determine tasks that need to be done for the VHDL (FSM) part of the project.
1	SC	Make tutorials in the manual and learn how to work with ZigBee.
1	C	Refresh C-coding and read through the manual.
2	S	Finishing the design, simulating and assembling the analog board.
2	C	Sketch a general design of the route planner and start coding it.
2	SC	Finish the tutorials and start the C code of the serial communication.
2	V	Start drawing FSD and writing the VHDL code and communicating with sensor and UART subgroup regarding their signals of the FSM.
3	S	Writing the VHDL code for the frequency counter and distance approximator.
3	C	Finalize the route planner in C.
3	SC	Finalize the serial communication and eliminate possible errors.
3	V	Continue writing the VHDL code for the FSM.
4	C	Improve the C-code and remove any errors.
4	V	Start finishing writing the VHDL code for the FSM.
5	C	Test the improved C-code.
6	V	Debugging the VHDL code if necessary and finishing the VHDL code.
7	V	Debugging and finishing the VHDL code if necessary.
7	E	Test with Roversim if possible.
8	E	Testing and final symposium.
8	E	Completing the final report.
9	E	Preparing the oral presentation.

Table I.2: Schedule

The group will try to follow this schedule as strictly as possible. However, in case of being behind schedule, one should make efforts to get back on track. On the other hand, if one is ahead of the schedule, one can always finish his/her part and perhaps assist the other subgroups.

I.5. Risks Analysis

In this chapter, we will review several risks with a high likelihood of occurring and how to counter or resolve the possible issues that might ensue.

The risks are divided into three categories based on their main cause.

I.5.1. Causes

Time related risks
Time-related risks involve the possibility of a team not fulfilling their work in time, which results in missing important deadlines. This can be caused by: miscommunication among team members, procrastination, misunderstanding of the assignment, and lack of motivation.

Member related risks
These risks are caused by a team member not fulfilling their workload or contributing to the project, which can be caused by a lack of time, motivation, or laziness. Unexpected personal issues are part of member-related risks. Even small occurrences such as missing meetings and not attending lab sessions can fall under member-related risks.

Communication related risks
These risks are caused by bad communication. For example a miscommunication of who will finish the report or working on a problem that is already finished or unnecessary. The unclear communication between members can lead to small disputes. In the worst case, those might escalate into quarrels that leave long-lasting grudges between members.

I.5.2. Avoiding and solving risks and issues

Time related risks
An easy way to avoid time-related risks is to keep track of the progress within the group and make good planning at the start of the project.

In this planning, we clearly state the expectations and deadlines so everyone can clearly see what has to be done and when. This way, we can avoid anyone misreading or misunderstanding an assignment. Expectations and deadlines will be made clear and there will be consequences, which are elaborated in the member-related risks subsection.

Member related risks
These issues can be countered by following up on the progress of each individual and assessing their input. If it is noticed that a member doesn't take his responsibilities seriously, actions will be taken.

Members will be held accountable if they are frequently late or miss meetings. Attendance will be noted for important meetings, as well as being late without a good reason. The attendance will be checked by the chair. He will relay this information to the secretary who will note this down in a public list.

A member could also compromise the group by delivering work that is unfinished or sub-par. Therefore it is important that everyone's work gets checked by the rest before submission. Before handing anything in, everyone has to go over everyone's work. Nothing will be submitted without unanimous approval.

Each subgroup will choose its own specific way of penalizing members. Penalties will be handed out on a case-by-case basis. This is important as many circumstantial factors might be at play (someone is late because of the NS, or traffic). It is important to remember that sub-member work is not necessarily the result of ill intent. Possible penalties might include: having them finish the work on the weekend, assigning them a lower grade at the end, and expulsion from the project [Worst case].
In the case that a member would drop out, the workload will have to be redivided among the remaining members to the best of our ability.

Communication related risks
These risks can be prevented by making a good plan and writing it down. Everyone should check if they clearly understand their tasks and duties, before finalizing any plan

Communication will mainly happen with each other on WhatsApp and during lab sessions on Monday and Thursday.

Risks due to human error
To prevent situations where a human error or problem with a programming environment causes a loss of code or its corruption, the GitHub repository will be used and accessed by all members through the usage of GIT. "The stupid content tracker" program as its manual describes it. This will track all the progress made to the code and ensure that when the problem occurs it can be simply reverted. GIT will also be very useful for revision tracking thus increasing the efficiency of workflow.

I.5.3. Conclusion of risks analysis
Most, if not all of the expected and probable risks are easy to avoid with clear planning and a clear task allocation that everyone follows. It is important to have a clear power structure that keeps the individual subgroups accountable according to that planning and ensures the quality of work.

I.6. Conclusion

In short, the goal of this project is to construct an autonomous robot that is able to pass a set of challenges. It should be able to find a solution to the maze, using an onboard infrared sensor and the instructions given to it by an algorithm running on a computer. These instructions will be passed to the robot via wireless communication.

The workload has been split over four subgroups. The first of them will write the VHDL code for a finite-state machine that will allow the robot to detect obstacles in the path. Another group is responsible for the serial communication between the computer and the robot. They will write a special subprogram for the computer and the corresponding VHDL code for the robot. The third group will write the software that will be solving the maze and sending corresponding instructions to the robot. The last group is responsible for making a sensor that can detect mines in the maze.

There are also risks and limitations that pose a challenge to this project. For example, the time constrain, or risks related to miscommunication between group members, as outlined in Chapters 3 and 5. Our goal is to carefully analyze them and utilize corresponding solutions and tools that will help us to minimize the possibility of failure during this project. Examples of those tools are even workload division, Gantt Chart, and GIT.

In the end, careful execution of the content of this project plan will ensure this project's success.

Bibliography

[1] Ing. B. Jacobs et al. *Project Smart Robot Challenge*. Delft University of Technology. 2023.

[2] Prateek Narang. *Find shortest path*. https://www.scaler.com/topics/hamiltonian-cycle/.

[3] Ing. X. van Rijnsoever et al. *Manual Course Lab Digital Systems B Line-Follower*. Delft University of Technology. 2022.