BUL LAB - 4

Celestine Machuca (2570138)
Soodeh Mousaviasl (2571713)

Instructor: Prof.Dr.Rasmus Rettig

Date: 26.04.2023

Task 1 :CAN-Interface Setup und Test

- Install the USB CAN interface according to the instructions above
- Connect the two CAN interfaces (CANH-CANH and CANL-CANL) and activate the terminating resistors by wire bridges between R+ and R-for both interfaces
- Load the "CAN_test.py" program and analyse the function. Modify the program if required. Briefly describe the function. Draw a diagram of the data flow.

The program provided makes uses of canalyst for the usb control of the can transreceiver.

1. The program first creates a canalyst device object with a bitrate of 500000.

```
dev = canalystii.CanalystDevice(bitrate=500000)
```

2. Then it creates a message object with the following parameters

```
new_message = canalystii.Message(can_id=0x300,
                                  remote=True,
                                  extended=False,
                                  data_len=8,

data=(0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08))
```

3. Then it sends the message to channel 1 of the device.

```
dev.send(1, new_message)
```

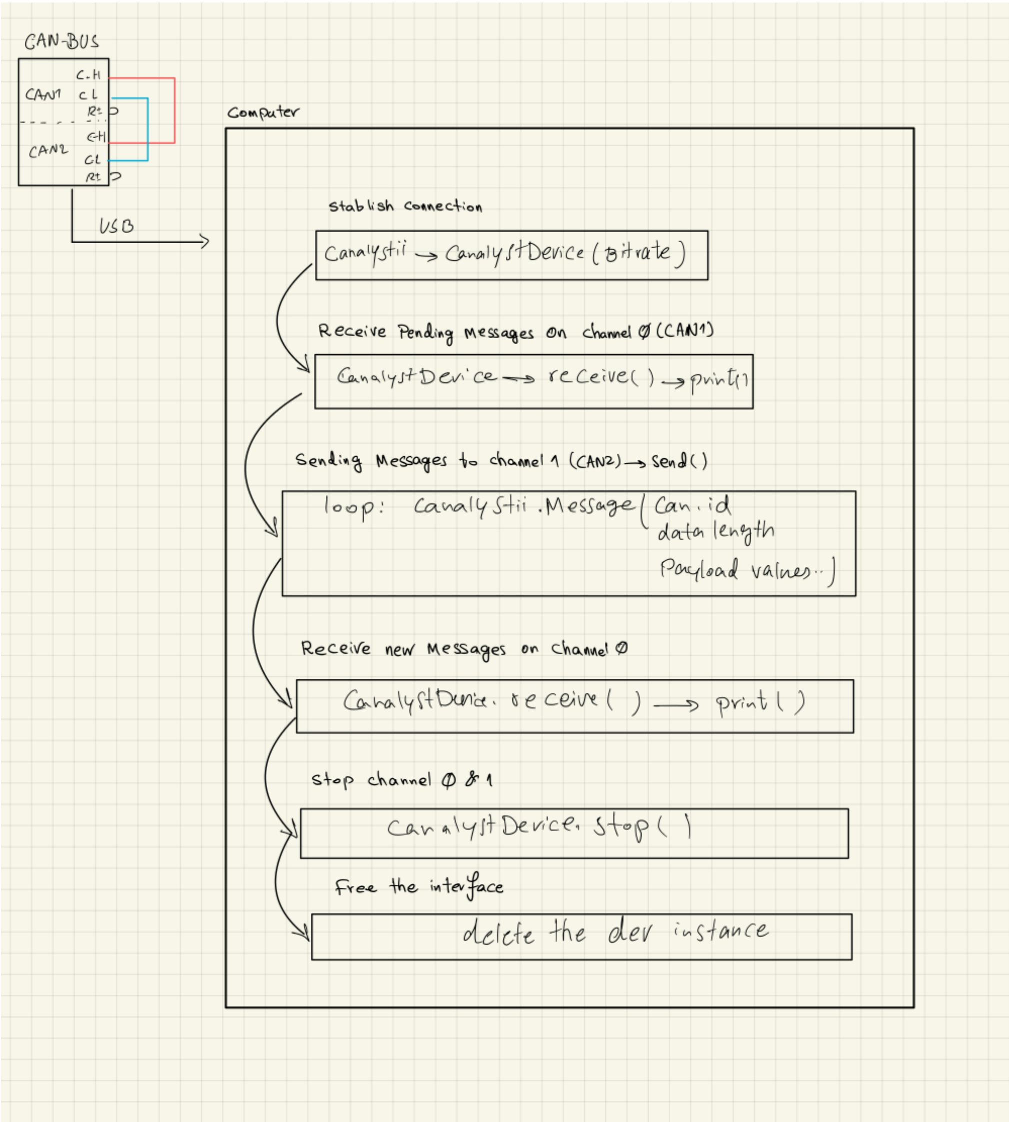
4. Then it receives the message from channel 0 of the device.

```
for msg in dev.receive(0):
    print(msg)
```

The message output is as follows

```
[CanalystMessage ID=0x300 TS=0x9f5227 Data=0000000000000000]
[CanalystMessage ID=0x300 TS=0x9f522b Data=0000000000000000, CanalystMessage
ID=0x300 TS=0x9f522c Data=0000000000000000, CanalystMessage ID=0x300
TS=0x9f522d Data=0000000000000000]
```

Data Flow Diagram



- Change the parameters and payload in the transmission (bitrate; can_id, remote, extended, data_len, data); document your investigations systematically and completely

We tested the message function by varying the different parameters and checking the output on code.

if `remote` is true the data field of the message will be ignored and the message will be sent as a remote frame.

with remote = True

```
[CanalystMessage ID=0x300 TS=0x9f5227 Data=0000000000000000]
```

```
[CanalystMessage ID=0x300 TS=0x9f522b Data=0000000000000000, CanalystMessage
ID=0x300 TS=0x9f522c Data=0000000000000000, CanalystMessage ID=0x300
TS=0x9f522d Data=0000000000000000]
```

with remote = False

```
[CanalystMessage ID=0x300 TS=0xa7d61a Data=0102030405060708]
[CanalystMessage ID=0x300 TS=0xa7d621 Data=0102030405060708, CanalystMessage
ID=0x300 TS=0xa7d623 Data=0102030405060708, CanalystMessage ID=0x300
TS=0xa7d626 Data=0102030405060708]
```

if `extended` is true then CAN ID is an extended address according to the documentation of canalystii.

extended = True

```
[CanalystMessage ID=0x300 TS=0xb872be Data=0102030405060708]
[CanalystMessage ID=0x300 TS=0xb872c2 Data=0102030405060708, CanalystMessage
ID=0x300 TS=0xb872c5 Data=0102030405060708, CanalystMessage ID=0x300
TS=0xb872c7 Data=0102030405060708]
```

extended = False

```
[CanalystMessage ID=0x300 TS=0xbd411d Data=0102030405060708]
[CanalystMessage ID=0x300 TS=0xbd4122 Data=0102030405060708, CanalystMessage
ID=0x300 TS=0xbd4125 Data=0102030405060708, CanalystMessage ID=0x300
TS=0xbd4127 Data=0102030405060708]
```

`data_len` is the length of the data field in bytes.

`data` is a tuple of bytes to be sent in the data field of the message.

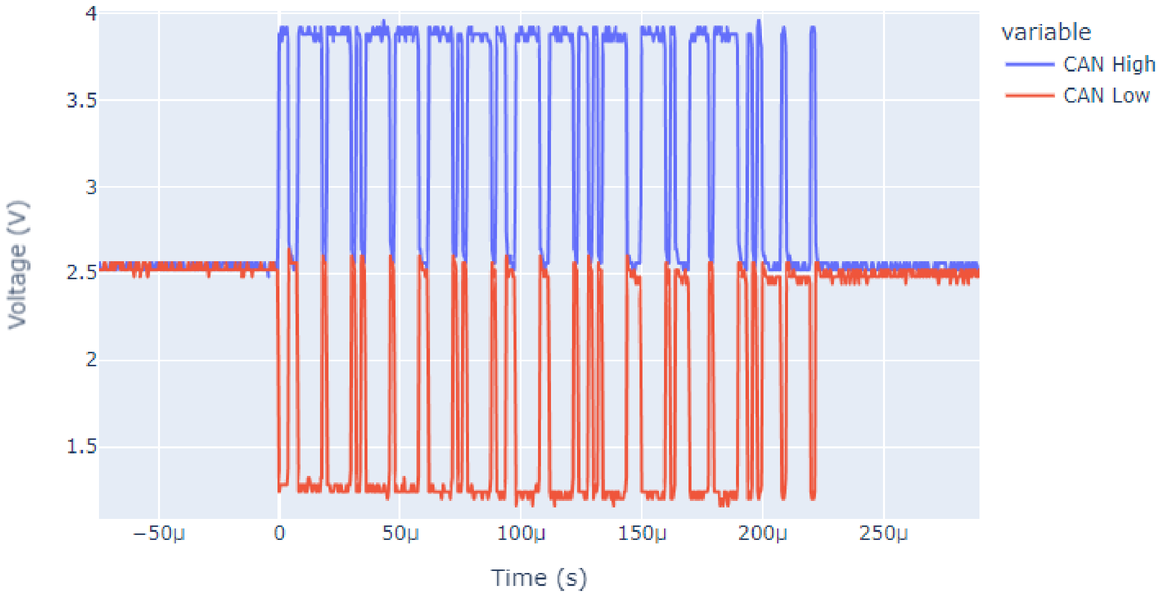
Exercise 2 – CAN Physical Layer (Loopback CAN1-CAN2)

- Use two channels of the oscilloscope to carry out a differential measurement of the voltage between CANH and CANL.
- Run the "CAN_test.py" program and follow the process on the oscilloscope.
- Attach a screen dump of a complete CAN telegram to your log and label the individual segments of the CAN telegram.
- Change the "bitrate" parameter from 500000 to 250000 and make a comparison measurement with the oscilloscope. Document your results with screen dumps and describe the differences.

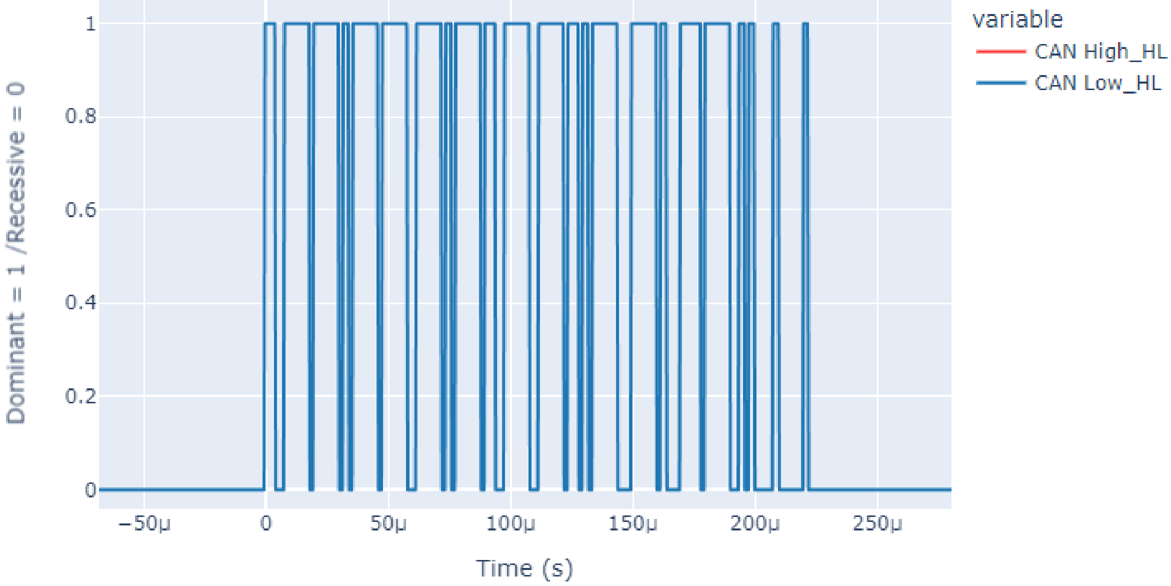
500000 baud rate

CH1 - CANH

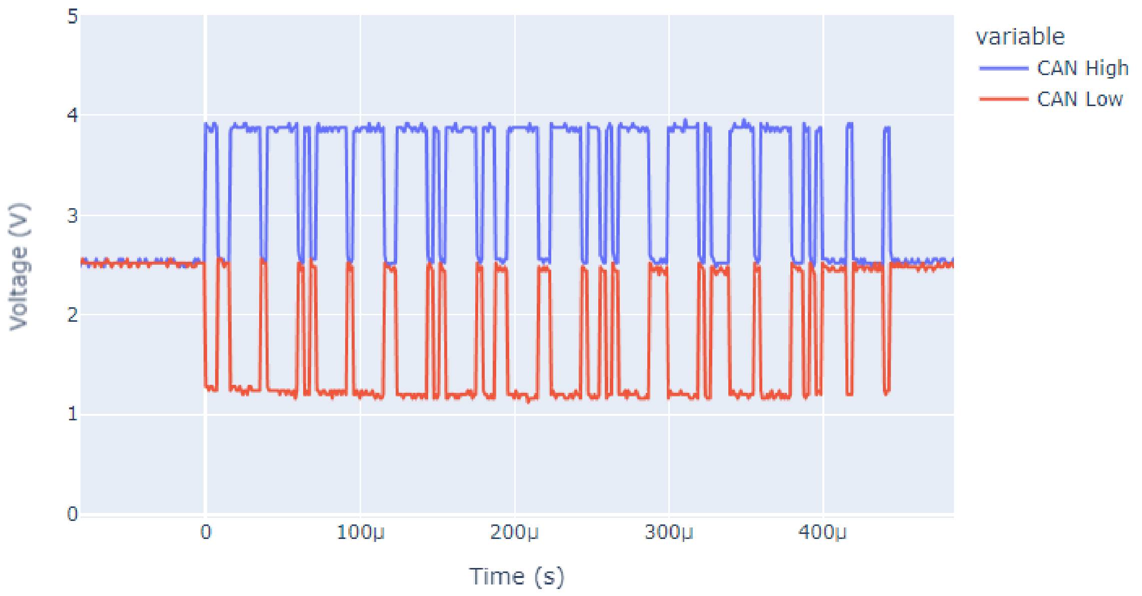
CH2 - CANL



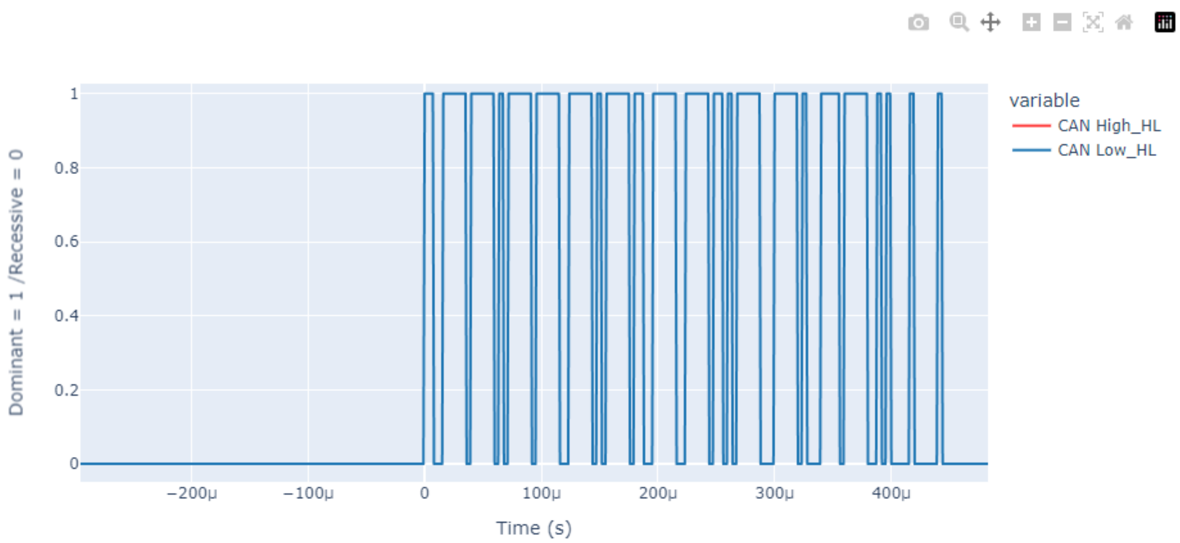
Post processing for dominant and recessive bits



250000 baud rate



Post processing for dominant and recessive bits



The time that it took to send one of the messages was 450 microseconds for 250000 baud rate and 225 microseconds for 500000 baud rate is inversely proportional to the baud rate.

Exercise 3 – Loopback CAN1-CAN2

- Setup DHT11 and CCS811 according to lab exercise ID S3.

- Modify the python program so that all 5 sensor values are first recorded and then transferred to the CAN bus (CAN1). To do this, first define your CAN telegram(s) with the assignment of the bits in the payload. You may have to distribute your payload across several telegrams

In order to record the data correctly out of the arduino we made use of serial.ReadLine() method to read the data from the serial port and then split the data using the comma as the delimiter.

```
df = pd.DataFrame(columns=["dht_temp", "dht_hum", "ccs811_co2",
"ccs811_tvoc", "ccs811_temp"])
linesToRead = 10
for i in range(0, linesToRead):
    try:
        line = ser.readline().decode('utf-8')
        parts = line.split(',')
        if len(parts) == 5:
            df = df.append({'dht_temp': float(parts[0]), 'dht_hum':
float(parts[1]), 'ccs811_co2': float(parts[2]), 'ccs811_tvoc':
float(parts[3]), 'ccs811_temp': float(parts[4])}, ignore_index=True)
        print(line)
    except Exception as e:
        print(e)
        continue
display(df)
```

We save the data as a pandas dataframe.

```
display(df)
```

✓ 0.0s

	dht_temp	dht_hum	ccs811_co2	ccs811_tvoc	ccs811_temp
0	49.0	29.0	0.0	0.0	25.00
1	49.0	29.0	0.0	0.0	21.33
2	49.0	29.0	0.0	0.0	32.62
3	48.0	28.7	400.0	0.0	37.68
4	48.0	28.7	400.0	0.0	43.55
5	48.0	28.7	406.0	0.0	30.17
6	48.0	28.8	400.0	0.0	25.56
7	48.0	28.8	400.0	0.0	27.86

The data is sent per row as a json object to the can bus.

```
for index, row in df.iterrows(): #iterare per row
```

```
    json_data = row.to_json()
    test(json_data)
```

The test function is as follows

```
def test(message : str):
    d = None
    mc = canpack.MessageConstructor()
    m_array = mc.create_message(message)
    reconstructed_message = ''
    for m in m_array:
        letters_received = canpack.Message.fromBytesToMessage(m)
        reconstructed_message += letters_received.chuck.decode('ascii')
        print('Sent: ', letters_received)
        # print(hex(int.from_bytes(m)))
        send_message(m)
    for i in range(0, len(m_array*2)):
        print('i: ', i)
        try:
            received_message = dataInputBuffer(0)
            str = received_message.__str__()
            matches = re.findall(r'Data=([a-fA-F0-9]+)', str, re.IGNORECASE)

            for match in matches:
                print('match: ', match)
                byte_array = bytes.fromhex(match)
                #convert back to messgae
                d = canpack.Message.fromBytesToMessage(byte_array)
                print('Received: ', d)
                #check if checksum is correct
                if d.checksum == mc.checksum(d.chuck.decode('ascii')):
                    print('Checksum is correct')
                else:
                    exit('Checksum is incorrect')
                    #this will kill everything


        except IndexError:
            print('No data received')


        print(received_message)
    print('Original message: ', message)
    print('Reconstructed message: ', reconstructed_message)
    return d
```

- Check whether the sensor data corresponds to the data read back on the CAN bus (CAN2).
- Document your tests as examples.

the output of one test can be seen below

```
Original message:
{"dht_temp":49.0,"dht_hum":29.0,"ccs811_co2":0.0,"ccs811_tvoc":0.0,"ccs811_te
mp":25.0}
{"dht_temp":49.0,"dht_hum":29.0,"ccs811_co2":0.0,"ccs811_tvoc":0.0,"ccs811_te
mp":21.33}
Sent:  Chuck: b'{"dh', Checksum: 361, Counter: 1
Sent:  Chuck: b't_te', Checksum: 428, Counter: 2
Sent:  Chuck: b'mp":', Checksum: 313, Counter: 3
Sent:  Chuck: b'49.0', Checksum: 203, Counter: 4
Sent:  Chuck: b',"dh', Checksum: 282, Counter: 5
Sent:  Chuck: b't_hu', Checksum: 432, Counter: 6
Sent:  Chuck: b'm":2', Checksum: 251, Counter: 7
Sent:  Chuck: b'9.0,', Checksum: 195, Counter: 8
Sent:  Chuck: b'"ccs', Checksum: 347, Counter: 9
Sent:  Chuck: b'811_', Checksum: 249, Counter: 10
Sent:  Chuck: b'co2"', Checksum: 294, Counter: 11
Sent:  Chuck: b':0.0', Checksum: 200, Counter: 12
Sent:  Chuck: b',"cc', Checksum: 276, Counter: 13
Sent:  Chuck: b's811', Checksum: 269, Counter: 14
Sent:  Chuck: b'_tvo', Checksum: 440, Counter: 15
Sent:  Chuck: b'c":0', Checksum: 239, Counter: 16
Sent:  Chuck: b'.0,"', Checksum: 172, Counter: 17
Sent:  Chuck: b'ccs8', Checksum: 369, Counter: 18
Sent:  Chuck: b'11_t', Checksum: 309, Counter: 19
Sent:  Chuck: b'emp"', Checksum: 356, Counter: 20
Sent:  Chuck: b':21.', Checksum: 203, Counter: 21
Sent:  Chuck: b'33}\x00', Checksum: 227, Counter: 22
```

```
7b22646869010100
745f7465ac010200
6d70223a39010300
34392e30cb000400
2c2264681a010500
745f6875b0010600
6d223a32fb000700
392e302cc3000800
226363735b010900
3831315ff9000a00
636f322226010b00
3a302e30c8000c00
2c22636314010d00
733831310d010e00
5f74766fb8010f00
63223a30ef001000
2e302c22ac001100
6363733871011200
31315f7435011300
656d702264011400
3a32312ecb001500
33337d00e3001600
22
Chuck: b'{"dh', Checksum: 361, Counter: 1
Checksum is correct
Chuck: b't_te', Checksum: 428, Counter: 2
```

```
Checksum is correct
Chuck: b'mp":', Checksum: 313, Counter: 3
Checksum is correct
Chuck: b'49.0', Checksum: 203, Counter: 4
Checksum is correct
Chuck: b',"dh', Checksum: 282, Counter: 5
Checksum is correct
Chuck: b't_hu', Checksum: 432, Counter: 6
Checksum is correct
Chuck: b'm":2', Checksum: 251, Counter: 7
Checksum is correct
Chuck: b'9.0,', Checksum: 195, Counter: 8
Checksum is correct
Chuck: b'"ccs', Checksum: 347, Counter: 9
Checksum is correct
Chuck: b'811_', Checksum: 249, Counter: 10
Checksum is correct
Chuck: b'co2"', Checksum: 294, Counter: 11
Checksum is correct
Chuck: b':0.0', Checksum: 200, Counter: 12
Checksum is correct
Chuck: b',"cc', Checksum: 276, Counter: 13
Checksum is correct
Chuck: b's811', Checksum: 269, Counter: 14
Checksum is correct
Chuck: b'_tvo', Checksum: 440, Counter: 15
Checksum is correct
Chuck: b'c":0', Checksum: 239, Counter: 16
Checksum is correct
Chuck: b'.0,"', Checksum: 172, Counter: 17
Checksum is correct
Chuck: b'ccs8', Checksum: 369, Counter: 18
Checksum is correct
Chuck: b'11_t', Checksum: 309, Counter: 19
Checksum is correct
Chuck: b'emp"', Checksum: 356, Counter: 20
Checksum is correct
Chuck: b':21.', Checksum: 203, Counter: 21
Checksum is correct
Chuck: b'33}\x00', Checksum: 227, Counter: 22
Checksum is correct

reconstructed message:
{"dht_temp":49.0,"dht_hum":29.0,"ccs811_co2":0.0,"ccs811_tvoc":0.0,"ccs811_te
mp":21.33}
```

- Draw the data flow diagram. How do you get the data onto the CAN bus, how do you read it back?

Exercise 4 – Automated error detection

- Automate the process of exercise 3 so that deviations are automatically detected and reported (displayed on the screen).

```
def checksum(self, message):
    return sum(ord(c) for c in message) % 2**16 # 16 bit checksum
```

For automatic error detection we made use of the checksum of the message. The checksum is calculated by summing up the ascii values of the characters in the message and then taking the modulo 2^16 of the sum. The checksum is then compared to the checksum of the message. If the checksums are not equal, then the program will exit.

The code for the automatic error detection is as follows:

```
#convert back to messgae
d = canpack.Message.fromBytesToMessage(byte_array)
print('Received: ', d)
#check if checksum is correct
if d.checksum == mc.checksum(d.chuck.decode('ascii')):
    print('Checksum is correct')
else:
    exit('Checksum is incorrect')
    #this will kill everything
```

the message is converted back to a message object and then the checksum is calculated and compared to the checksum of the message. If the checksums are not equal, then the program will exit.

- Run the test with at least 100 messages. Are messages lost? Do you observe transmission errors?
- Include the documented source code with your lab report.

We ran the program for 100 messages and no messages were lost. We did not observe any transmission errors. The source code is included in the appendix.

Exercise 5 – Large CAN-Network

- Coordinate your activities with the other teams in the lab: Which bit rate do you want to use? Who uses which identifiers?

We used a bit rate of 500 kbit/s. We used the following identifier 0x300

- Document the result in a table.

- Describe the encoding and structure of the sensor data in your CAN messages using your python code

for the encoding of our messages the following python module was created:

```
import struct

class Message:
    def __init__(self, chuck, checksum : int, counter : int):
        self.chuck = chuck
        self.checksum = checksum
        self.counter = counter
    def get_struct(self):
        #covert chuck to bytes
        if type(self.chuck) == str:
            #convert to ascii
            self.chuck = self.chuck.encode('ascii')
        return struct.pack('4sHH', self.chuck, self.checksum, self.counter)

    @staticmethod
    def fromBytesToMessage(raw_message):
        return Message(*struct.unpack('4sHH', bytes(raw_message)))


    def __str__(self):
        return f'Chuck: {self.chuck}, Checksum: {self.checksum}, Counter:
{self.counter}'

class MessageConstructor:
    def __init__(self):
        self.counter = 0

    def checksum(self, message):
        return sum(ord(c) for c in message) % 2**16 # 16 bit checksum

    def create_message(self, message):
        chunks = [message[i:i+4] for i in range(0, len(message), 4)]# every 4
letters
        m_array = []
        for chunk in chunks:
            self.counter = (self.counter + 1) % 2**16 # 16 bit counter
            cs = self.checksum(chunk)
            #print(f'Sending chunk: {chunk}, Checksum: {cs}, Counter:
{self.counter}')
            m = Message(chunk, cs, self.counter).get_struct()
            m_array.append(m)
        return m_array
```

As it can be seen, the message is encoded in the following way:

The `Message` class is used to represent a single message that will be sent over the CAN network. Each message consists of a chunk of data, a checksum, and a counter. The chunk of data is a string of up to 4 characters, the checksum is a 16-bit integer, and the counter is also a 16-bit integer.

The `get_struct()` method is used to convert the message into a format that can be sent over the CAN network. This is done using the struct.pack function, which takes a format string and a sequence of values and returns a bytes object. The format string '4sHH' indicates that the data should be packed as a string of 4 bytes followed by two unsigned short integers.

The `fromBytesToMessage` static method is used to convert a bytes object back into a Message object. This is done using the struct.unpack function, which takes a format string and a bytes object and returns a tuple of values.

The `MessageConstructor` class is used to create a sequence of Message objects from a string of data. The create_message method splits the data into chunks of 4 characters, calculates the checksum for each chunk, increments the counter, and creates a Message object for each chunk.

The checksum is calculated by summing the ASCII values of the characters in the chunk and taking the remainder when divided by 2^16.

The counter is incremented for each chunk and wraps around to 0 when it reaches 2^16.

In this way, the data is encoded into a sequence of messages that can be sent over the CAN network. Each message contains a chunk of the original data, a checksum to verify the integrity of the data, and a counter to keep track of the order of the messages.

The message as seen on `get_struct()` is encoded in the following way:

4 bytes	**2 bytes**	**2 bytes**
4s	H	H
chunk	checksum	counter

- Record the entire CAN network with all participants.
- Transfer your data to the bus and check whether at least one group receives correctly. Carry out the test in both directions. Document the results

The group composed by Catherine and Alisa were able to receive our messages.

Group	Id	From us	To us
(us)	0x300		
Alisa and Catherine	0x1EE	received	received

message captured :

```
[CanalystMessage ID=0x1ee TS=0x0 Data=0x4a7f25933b1d6a8b]
[CanalystMessage ID=0x1ee TS=0x0 Data=0x7b930e589d2c1651]
[CanalystMessage ID=0x1ee TS=0x0 Data=0x6f3d19a8b7c56e2d]
[CanalystMessage ID=0x3a7 TS=0x0 Data=0x512b6748e9f7c0ad]
[CanalystMessage ID=0x1ee TS=0x0 Data=0x9085b4d72e6c37a1]
[CanalystMessage ID=0x3a7 TS=0x0 Data=0x1bf6d54397a2e80c]
[CanalystMessage ID=0x1ee TS=0x0 Data=0x39c74a8e0d5b6f21]
[CanalystMessage ID=0x3a7 TS=0x0 Data=0x5e7340af8219c8b6]
[CanalystMessage ID=0x1ee TS=0x0 Data=0x23d46eacfc18975b]
```

Appendix

Part 3: stdout Source

Please refer to the document `part3.txt`.

Part 4: stdout Source

Please refer to the document `part4.txt`.