ID S2

The CAN BUS Hands-On

Required Knowledge

- Lab ID S1 (function and characterization of ultrasound sensor)
- Differential measurements with a two-channel oscilloscope; calibration of probes
- CAN bus - physical Layer: CANHigh, CANLow, data rate
- CAN bus – communication/logic layer: CAN-telegrams
- Bus-arbitration on a CAN-bus

Material

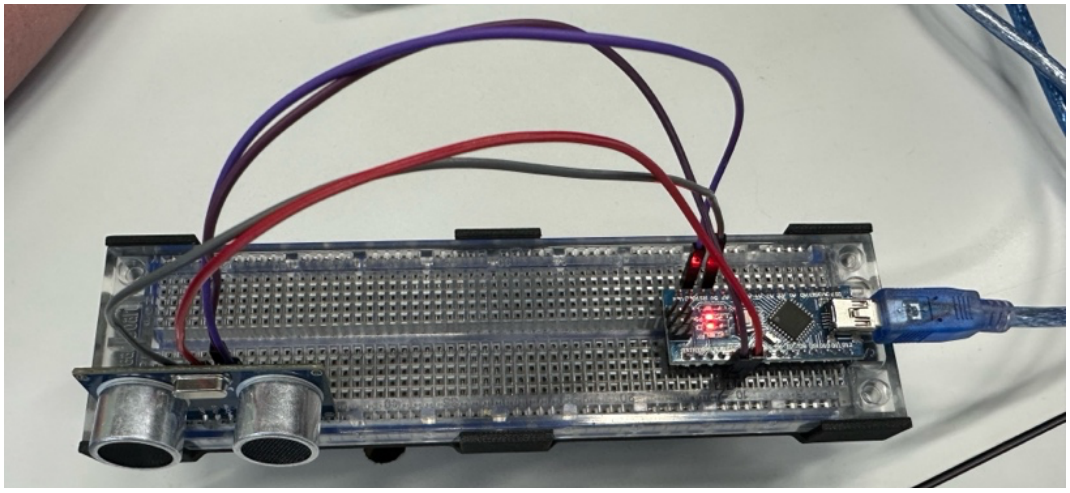
- Material from experiment ID S1 (Arduino, HCSR04)
- Program code from ID S1 (HCSR04)
- USB-CAN interface TRU-Components USB-CAN (https://asset.conrad.com/media10/add/160267/c1/-/en/002368701DS00/datenblatt-2368701-tru-components-tc-9474804-can- converter-usb-can-bus-sub-d9-not-galvanically-isolated-5-vdc-1-st.pdf)
- Male/male cables

Duration approx. 4h

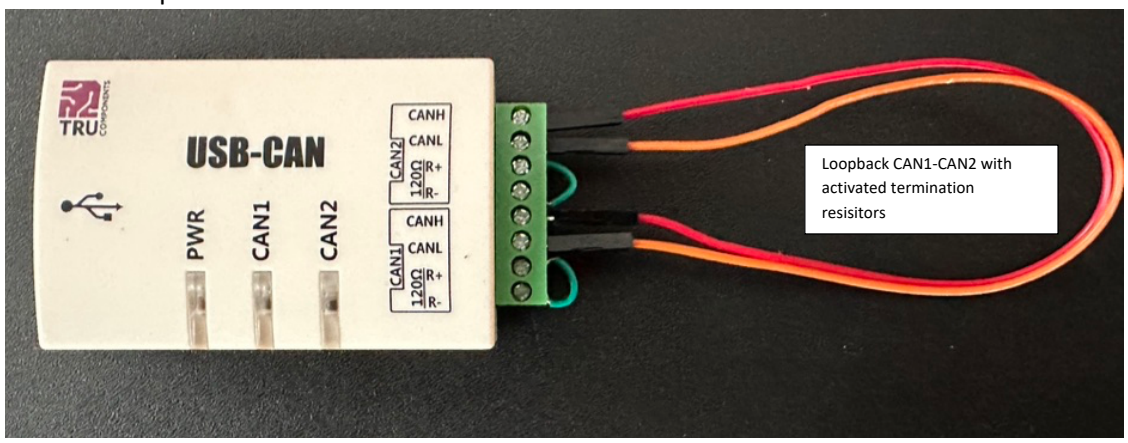
Setup

Hardware:

- HCSR04 as setup in lab exercise S1



- TRU-components USB-CAN



Software:

USB-Driver:

- Windows: Install USB-Drivers with ZADIG (https://zadig.akeo.ie); map USB-CAN-Interface to libusb32; Driver will be automatically downloaded and installed
- OSX: Install Homebrew (OSX-Packet-Manager) with these commands:

- /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
- echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zprofile
- brew install libusb

For emergencies (badly translated windows software):

- https://asset.conrad.com/media10/add/160267/c1/-/gl/002368701DL00/download-2368701-tru-components-tc-9474804-can-umsetzer-usb-can-bus-sub-d9-nicht-galvanisch-getrennt-5-vdc-1-st.zip

Python:

- Anaconda/Spyder with the library „catalystii“ („pip install canalystii“)
- Prepared programmes (Python)
- Your programs from ID S1 (HCSR04)

Exercise 1 – CAN-Interface Setup und Test

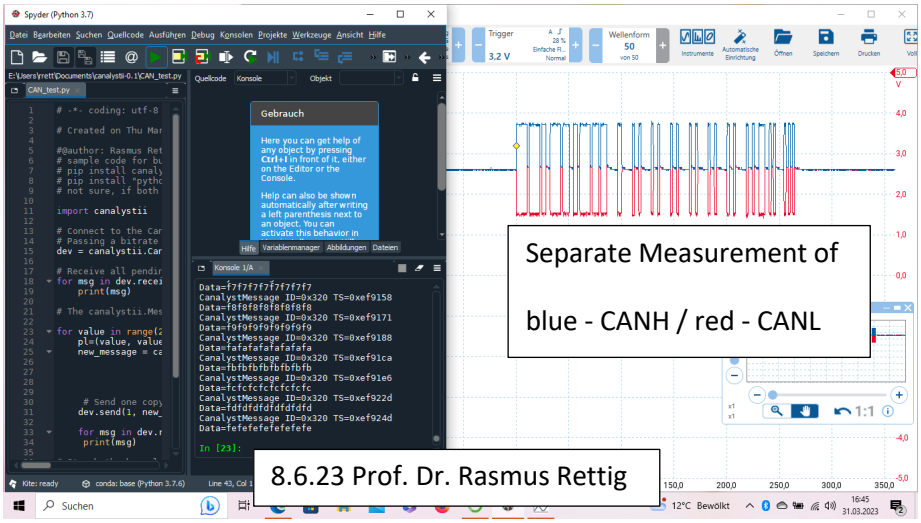
- Install the USB CAN interface according to the instructions above
- Connect the two CAN interfaces (CANH-CANH and CANL-CANL) and activate the terminating resistors by wire bridges between R+ and R- for both interfaces
- Load the "CAN_test.py" program and analyze the function. Modify the program if required.
- Change the parameters and payload in the transmission (bitrate; can_id, remote, extended, data_len, data); document your investigations systematically and completely

Deliverables:

- CAN_test: Briefly describe the function of the program and draw a diagram of the data flow.
- Measurement results for 5 different settings (bitrate; can_id, remote, extended, data_len, data)

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

- Calibrate 2 probes
- 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.
- Measure both CANH and CANL separately and use the math function to calculate CANH-CANL
- Create 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.



Deliverables:

- Photo of your setup
- Screen dump: Single CAN telegram with the sections marked (250kbaud, 500kbaud)

- Screen dump: Several CAN telegrams sequentially on one screen shot (250kbaud, 500kbaud)
- Analysis and description of the differences between the symbol rates

Exercise 3 – Setup sensors and transmit data to the CAN-Bus (Loopback CAN1-CAN2)

- Setup HCSR04 according to lab exercise ID S1.
- Modify the python program so that 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.
- Check whether the sensor data corresponds to the data read back on the CAN bus (CAN2).
- Document your tests as examples.
- Draw the data flow diagram. How do you get the data onto the CAN bus, how do you read it back?

Deliverables:

- Photo of your setup
- Diagram of the data flow including all components. Clarify how sensor data is readback
- Test results, number of cycles and number of errors

Exercise 4 – Automated error detection (Loopback CAN1-CAN2)

- Automate the process of exercise 3 so that deviations are automatically detected and reported (displayed on the screen).
- Run the test with at least 100 messages. Are messages lost? Do you observe transmission errors?

Deliverables:

- Test results: Total number of messages, number of errors, number of correct transmissions; pls. check consistency
- Include the documented source code with your lab report.

Exercise 5 – Large(r) CAN-Network (CAN1 to common bus, please disable termination resistors BEFORE connecting to the common bus lines)

- Coordinate your activities with the other teams in the lab: Which bit rate do you want to use? Who uses which identifiers?
- Document the result in a table.
- Describe the coding of the sensor data in your CAN messages using your python code.
- Record the entire CAN network with all participants.
- Transfer your data to the bus and whether all groups receive it correctly. Carry out the test in both directions. Document the results.

Deliverables:

- Table with the assignment of identifiers to groups and their payload.
- Graphical documentation of the entire CAN network
- Reading: Which groups' identifiers do you see? What is their payload?
- Writing: Which groups can confirm correctly receiving your message and payload?