# CAN BUS FOR VEHICLE APPLICATIONS*

1st Tom Alexander Bateson
*School of engenering*
*University of the west of England (UWE)*
**Bristol**, England
Tom@TomBateson.com
Student number: 19005205

***Abstract*—Modern vehicles utilise an increasingly growing amount of digitally controlled devices for sensing, actuation, control and other purposes. All of these devices need to communicate with each other to facilitate their operation and many modern vehicles facilitate this communication through the use of CAN (controller area network) bus. This level of sophistication however has not been present in classical vehicles that are now being converted to electric drivetrains. This paper presents the development and testing of a clean slate CAN communication protocol tailored to implementation in cars converted to electric drivetrains devoid of pre-existing digital control systems for their body electronics. The project will document the creation of both a software stack for the control unit and peripherals associated with the demonstration system as well as the electronic hardware in which the software will run. The performance of this system will then be tested rigorously and the suitability of the system for its intended purpose will be evaluated. By documenting this process and making the designs and code freely available, this paper aims to provide insights into the challenges and guidance for any future EV conversion projects or a basis for their CAN communication system should they wish to utilise the protocol developed in this research endeavour.**

### *A. Abbreviations and Acronyms*

CAN (Controller Area Network) MCU (micro controller unit) IC (integrated circuit)

## I. Introduction

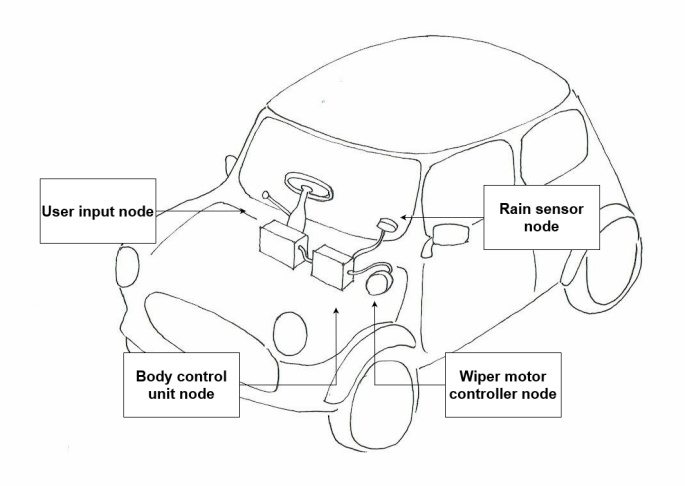


Fig. 1. In situ example of the demonstration system

the question that this research will attempt to answer is: can a can bus protocol be created to suit a converted electric vehicle?

to answer this question the first step is to conduct a detailed review of prior literature to both determine that the research is novel and that there is not already an answer to this question. if it is determined that the research is novel then this initial review can then be broadened to begin learning the topic and what will be required to begin work on developing this project. this research will be focused on articles about the can bus and existing protocols developed for it as well as current standards used for vehicle control. The leanings from this process are detailed in the following literature review section of this report.

in order to develop the can bus protocol, that is the subject of this research, hardware will also have to be produced to run the software that is developed. university technicians suggested that a portion of the whole cars body electronics system should be selected and recreated in a bench top form that would have a system present with sensing, control and actuation to enable a full piece of demonstration software to be developed on it.

The objective of this research then is to produce a functioning fore node demonstration of an automatic wiper system. A mock-up this is shown in Figure 1, consisting of a rain sensor node, user input node, a wiper motor controller node and a body control module node to coordinate the activities of the other nodes.

To achieve its functionality this system must provide low latency between nodes without missing any commands and not over-saturating the CAN bus which would increase latency. The communication scheme between all nodes in the system is shown in Figure 2. This shows the flow of information from the user input to the body control unit which deals with deciding whether or not input from the rain sensor is required to set the state of the wiper controller. The CAN system to be developed will also aim to be able to communicate between modules without requiring the body control unit while not interrupting functionality to increase the flexibility of the protocol.

### *A. Literature review*

The three sources I looked at initially to gain an understanding of the topic were several web articles around the CAN open protocol(in Automation  CiA) as well as the protocol's GIT hub(in Automation  CiA), a paper written

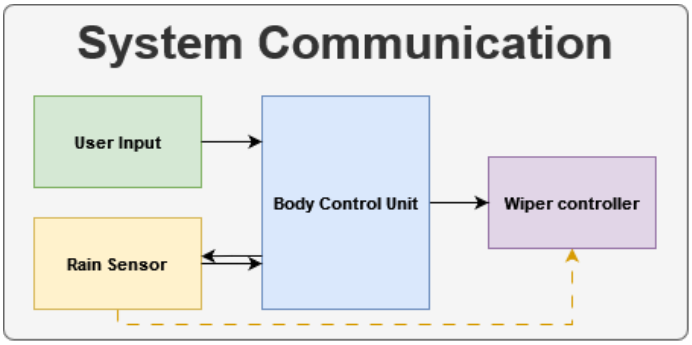


Fig. 2. Proposed information flow between nodes

about the development of a can bus system for CubeSat applications(Scholz et al. 2018) and another paper about the development of a can bus communication structure for electric vehicles(Ran et al. 2010). These sources sources give me a good understanding of how systems have been done in the past and give me some starting points for working on my system, such as having a state engine to govern the primary behaviour of each individual node in the system.

## II. Development

### A. Hardware Requirement Analysis

This project is focused on the development a CAN protocol as well as the supporting software to implement said protocol and demonstrate its functionality, therefore hardware selection must balance two factors predominantly. Firstly to ease development and enable rapid development time, the project should stick to a common development environment which will have wide support to enable easy prototyping and secondly appropriate selection of controllers for the automotive environment.

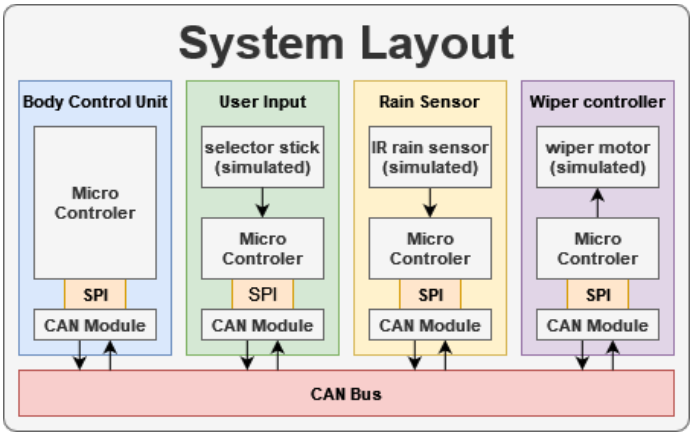


Fig. 3. system block diagram

IO requirements will not be very large for this project, as to meet the requirements The selected microcontroller must support only a small number of general input and output GPIO connections and a single SPI connection to the CAN module. In the final implementation of the system other connections such as one or more i2c busses may be called for but due to the tools being used for tis project, namely platform io, converting the code to run on another microcontroller with grater hardware support will not pose any major challenge.

### B. Hardware Review

The three suppliers identified were Espressif, Arduino and STMicroelectronics.

The ESP 32 microcontroller from Espressif is an especially interesting offering as it has an onboard CAN bus controller (Systems 2024) that would remove the necessity of a separate entire can module but would still need a transceiver module such as a TJA1050(Semiconductors 2024). The ESP32 can be programmed with the Arduino environment as well as Espressif's native ESP-IDF giving it the same wide range of software compatibility as an Arduino.

STMicroelectronics offers a range of microcontrollers specialised for different purposes and some of these are designed to support automotive applications such as the stm8 and SPC5 microcontrollers(STMicroelectronics 2024*a*).

Arduino offers a large range of microcontrollers suited to a variety of purposes. The development environment is also very widely supported by large amounts of software and hardware. This can be leveraged to increase the pace of development. It is also possible that software developed in the Arduino and development environment may be ported to other more appropriate microcontrollers that are not suited to this low-cost development project.

For the selection of a CAN bus module, there was one obvious choice due to its abundance and support. This module features an MCP2515(Technology 2024) CAN controller and a TJA1010(Technology 2024) CAN transceiver two widely supported pieces of hardware. The module features support for multiple development environments including Arduino, ESP IDF and STM cube IDE. It will be compatible with any of the considered microcontrollers.

### C. Hardware Selection

An Arduino nano was chosen as the most appropriate microcontroller for this project. This decision was made because the Arduino offers all of the functionality required to accomplish this project scope and its familiarity and software support will aid in project swiftness.

The ESP32 was not chosen as utilising an Arduino or STM32 and a separate CAN bus breakout board would allow for greater ease of development due to existing stand-alone integrated CAN controller and transceiver modules having greater existing software support.

the STM32 modules were rejected as they were both cost-prohibitive for the budget of this project and using automotive-rated hardware is of less importance at this stage in the project. Additionally, in the case of the SPC5, it uses a non-standard architecture that would significantly increase development and testing times.

The stm8 however should be evaluated for future work as it is rated for automotive applications while also being compatible with the Arduino development environment. meaning

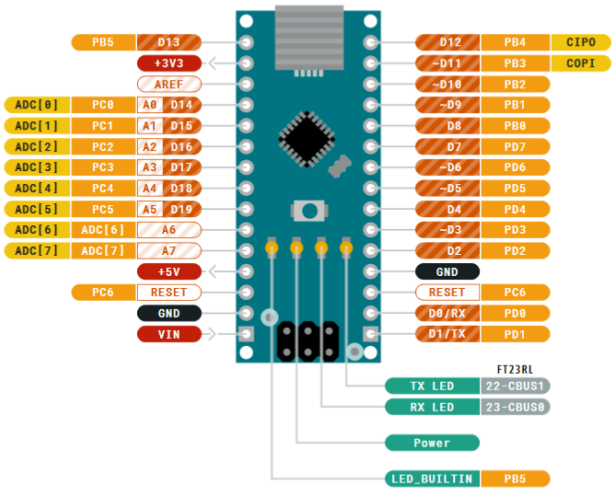


Fig. 4. Arduino pin out

that with minimal porting it could be compatible with the software developed for this project in the Arduino development environment(STMicroelectronics 2024*b*).

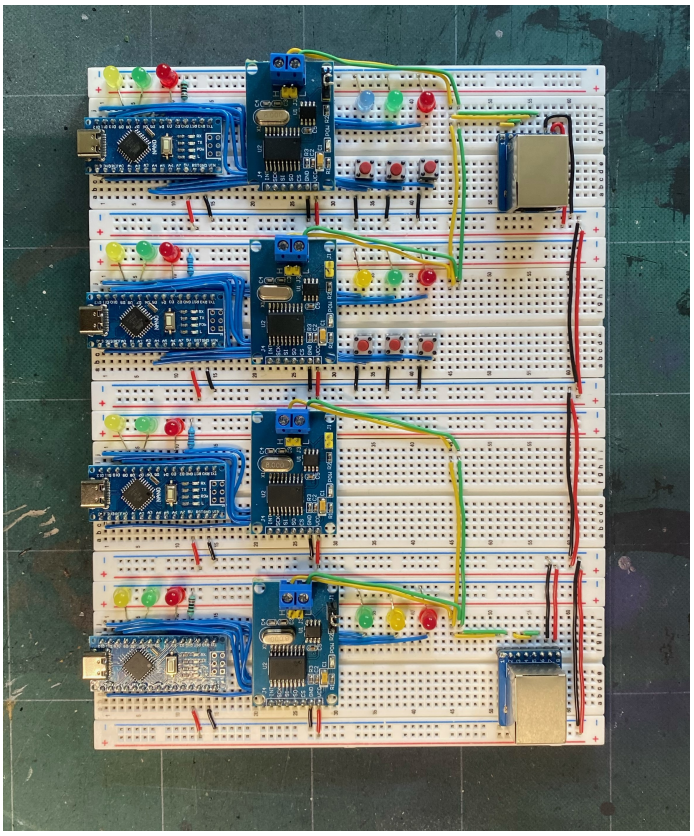

Fig. 5. Test hardware shown without Ethernet cable completing the CAN bus

### *D. supporting hardware and Assembly of the demonstration system*

with all hardware selection now made the focus turns to the implementation of the hardware in to a functioning demonstration system. To do this all the electronics will be brought together on bread boards. Bread boards are a tool that allows for easy and rapid prototyping of electrical systems using a pin grid system. 4 bread joined together as can be seen in (FIG) will be used for this project. Each breadboard in the prototype will represent an individual CAN node.

### *E. Programming environment*

Development was undertaken using three main pieces of software. Firstly, backup and version control were handled using GitHub. Software development was done on the VS code development environment with the platform IO extension to enable compiling for and interfacing with the micro controller.

Version control is an imperative part of any software development project. Cloud-based backup and access to previous versions of the code allow for easier review and debugging.

git hub allowed for the easy creation of a branch of software to be created for each of the 4 different nodes in the demonstration system being built. allowing the separation of all the changes from the program's core to be easily managed.

Platform IO is an excellent tool for hardware-focused development involving micro controllers. It supports a very wide range of micro controllers and development environments. All three of the micro controllers considered for this research, the ESP32 STM32 and Arduino Nano, have their native development environment supported, ESP-IDF for the ESP32 and cube IDE for the STM32, as well as the Arduino development environment for all three.

As well as this it is also able to provide tools such as software and hardware debugging, serial interfacing library management and compilation with easy search functionality. It also enables switching between micro controllers which has been very useful in this project, as development was started with a single Arduino Uno before specific hardware was acquired to support all four CAN modules. This required porting the software from the Arduino Uno to the Arduino Nano which was all seamlessly accomplished through the use of platform IO.

with the chosen hardware the best language to program in is the native Arduino language. this language is a derivative from c/c++ that has been modified for ease of use and intuitiveness as well as had hardware compatibility added to it allowing the user to interact with the devices IO and internal hardware such as the EAPROM (ref for ardunio langage)

the git hub (section that talks about open scorsing project an the benafitsh ther off) (ref to git hub)

### *F. Programming Strategy*

The above figure shows the structure of the CAN data messages used in this protocol. For this research only the standard 11-bit Identifier CAN messages and not the extended 29-bit Identifier messages have been used.

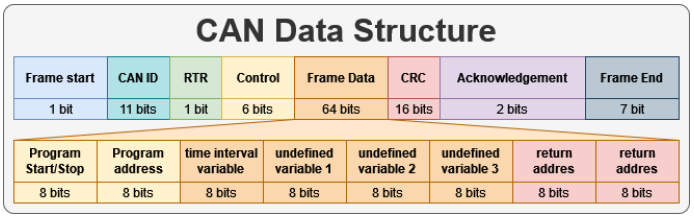


Fig. 6. layout of the can frame and the structure of the data within it

In this protocol, the CAN ID is used by the CAN module to identify whether a message put on the CAN bus is relevant to a specific node. This is done using the filtering and masking capabilities of the MCP2515 CAN controller IC. The protocol uses a mask to check the first three bits of the 11-bit CAN ID and then only accepts a message and stores it in the buffer if it has one of the two CAN IDs associated with the node.

The message frame is broken into 3 distinct sections. The first section of two bytes deals with identifying the node command that the message is related to and whether the command should be started or suspended. The next section has 4 bytes. These four bytes are the function settings. These will be passed into a function when it's run and may affect what it does. The first bytes are reserved for a time interval variable for node commands that use it this time interval variable will determine the interval between messages sent to the CAN bus. Finally, the last section of two bytes, which is also passed into the function, is a return address. This may be used by a function to specify the address to send a message if the destination for the message is not fixed.

### G. Program Review

Review of relevant code with programming examples

```
std::map<byte, FunctionPtr> functionDictionary={
    {0x10, setState},
    {0x20, testInterface_wiperMotorOnce},
    {0x21, testInterface_wiperMotorSeting},
    {0x30, exampleNodeFunction}
};

void runNodeCommand(unsigned char command[8]){
    auto funcId=functionDictionary.find(command[1]);
    if(funcId!=functionDictionary.end()&&command[0]==0x01){
        unsigned char functionSetings[6];
        memcpy(functionSetings, command+2, 6);
        FunctionPtr func=funcId->second;
        func(functionSetings);
    }
}
```

Listing 1. Function dictionary and the Node function running system C++ code 1

This is an example of a function dictionary and the function running system in the core of the node programs. In this case, it is the wiper motor node. The dictionary shows functions and their associated addresses. These are called node commands.

When a message is received by a node, it is passed into the runNodeCommand function. This function then strips out the first two bytes of the message. Bite one indicates whether the node command should be run or not and the second bite indicates what program function should be run by specifying the address of a function in the function dictionary.

The runNodeCommand function then strips away the first two bytes of information and passes through the remaining 6 bytes as the function settings.

This scheme of operation allows for one controller to easily handle running multiple functions only when they are required without using extensive switch cases or "else if" statements. It also increases the ease at which a program can be modified, as instead of editing a large switch case or other method of dealing with incoming information, the user must only write their function and add it to the function dictionary. Then it will automatically be run and dealt with whenever a message calling it is received by the node it has been implemented on.

```
case RUN_ACTIVE:
  receiveCAN(message);
  runNodeCommand(message);
  break;
```

Listing 2. Active response system state C++ code 1

```
case RUN_REACTIVE:
  if (receiveCAN(message)){
    runNodeCommand(message);
  }
  break;
```

Listing 3. Reactive response system state C++ code 1

The use of the runNodeCommand function discussed previously enables a very streamlined main program loop. The only things that are done in main.cpp are initialising the locally used variable for received CAN messages setting the initial state for the state machine and using the setup function in the setup state such as enabling serial output as well as the beginning of the can bus and configuring hardware-specific settings. Finally, switching the state to RUN when it is finished.

The main program loop will then only have to include two functions. firstly, "receiveCAN()" the function that checks for a new message every loop and "runNodeCommand()" the function that will take the message respond appropriately to it.

There are however two RUN states RUN ACTIVE and RUN REACTIVE the code for which is shown in Figures X and Y. Where RUN ACTIVE runs a node command in the CAN buffer every process cycle and RUN REACTIVE only runs a node command when a new command is received. These two states can be toggled between by using the "setState()" node command should the function of the node call for it.

## III. Testing

### A. Discussion of useful testing metrics

In the testing portion of this project there are two aspects of the system that should be tested to ensure that the system will work in the application that it is envisioned for. Firstly, the behavior of the system can be tested as the system is relatively simple and there are only a small number of possible input and output states. Secondly the ability of the system to function over the distances of cable found in a vehicle should be tested and quantified to ensure that the system will still function when installed in a vehicle.

### B. validating system behavior

During the development of the demonstration can system, incremental testing had been involved to verify that the components of the system are working as they should be. The system as a whole has also been tested to ensure its integrated behavior has been correct. However to ensure that there are no erroneous behaviors a rigorous testing of inputs and outputs must be conducted to ensure that the system behaves exactly how we have set out in the system design. To do this we will conduct a trial where all possible inputs will be tested and there outputs recorded. We will record not only the states of the system inputs and outputs but also the debug LEDs to ensure behavior on the CAN bus is as expected.

### C. validation and quantification of CAN signal integrity over vehicle relevant distances

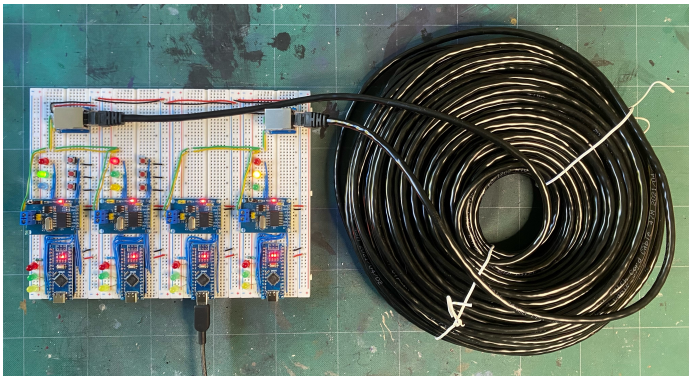

Fig. 7. Test system shown with the Ethernet cable used to test the systems performance over distance

To ensure that this bus system will work with vehicle relevant distances, part of the testing scheme is to transmit the can information between controllers separated by a large amount of cable. To accomplish this, the signal is run back and forward through a 47 metre Ethernet cable three times giving a total maximum run length of approximately 141 metres. It is suggested the typical maximum run length of a vehicle wiring harness is up to 22(Olbrich & Lackinger 2022). Although this is based on high voltage wires, we would expect control cables to be of similar a length so if the system still functions under these conditions we can expect that the system will function in its intended environment with a very large safety margin. We can also assume that there will be no issues induced by the length of the wiring harness for any larger road vehicles such as trucks.

There are two tests that we will perform, firstly the system will be connected over Ethernet in progressively longer lengths; 47m, 94m and 141m and functionality will be tested. Assuming the system remains functional at the maximum test length the test will then move on to the second section. This second section involves the quantification of performance. The CAN bus will be connected to an oscilloscope and the signal voltage will be measured at a range of lengths. 0m*, 47m, 94m and 141m. Three measurements of the peek to peek


Fig. 8. an example of the data gathering approach using an oscilloscope

voltage will be taken at each length and plotted. This will allow the calculation of the attenuation of the signal as well as extrapolation of the voltage drop to calculate where it will drop bellow the minimum signalling voltage thus suggesting a maximum buss length using the CAT5 Ethernet cable being used in these tests.

*it is important to note that in this context the 0m refers to 0 meters of Ethernet cable. however, there will still be a small amount of solid core bread board wire connecting parts of the can bus although due to its very short length its effect will be negligible and thus will be ignored.

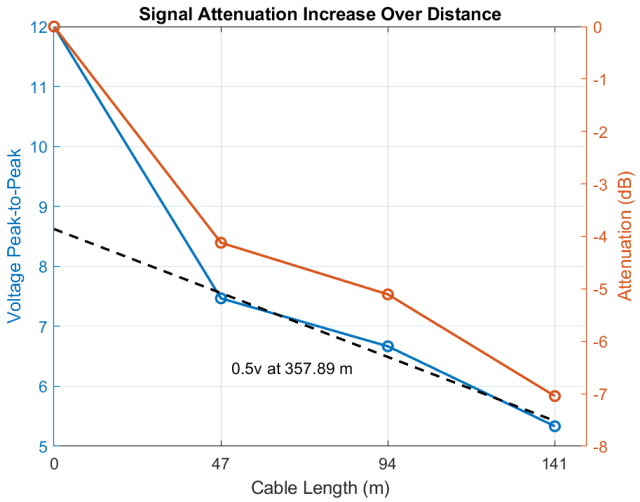


Fig. 9. Graph showing the peak to peak voltage and attenuation in DB of the CAN signal over a range of cable lengths

To test the effects of length on the signal strength measurements were taken of the peak to peak voltage of the signal transmitted on the X line on the other side of the Ethernet cable run from the transmitting module. An average of 10 measurements per cable length were taken of the peak peak voltage and then plotted. As well as this, the attenuation in decibels was calculated at each cable length and also plotted.

## IV. Deduction of further investigation

The inception of this research project came about because a project was ongoing at UWE to convert a classic Mini Cooper to an electric drive train. As a part of this project the aim was also to install modern body electronics such as automatic wipers electrically driven and controlled windows and other modern digitally controlled body electronics such as indicators, headlamps, ETC… This project aimed to provide a CAN protocol for this system to run on, as well as creating a desktop prototype of a single full system in, this case automatic wipers. However, the scope was limited to use simulated inputs and outputs to help with project timelines and costs. Now that the system has been proven out, further work can be done to demonstrate the systems functionality with actual automotive relevant hardware.

This can take two forms. Firstly, modifying the current system to interface with the actual automotive components that will be used in a final system, the wiper motor rain sensor and car wiper selector control. This will increase the confidence in the system and prove out further work such as moving on to creating more final prototype electronics that could be integrated into an actual vehicle.

## References


in Automation (CiA), C. (2024*a*), 'Canopen'.
**URL:** *https://www.can-cia.org/canopen/*

in Automation (CiA), C. (2024*b*), 'Canopen'.
**URL:** *https://www.can-cia.org/canopen/*

Olbrich, S. & Lackinger, J. (2022), 'Manufacturing processes of automotive high-voltage wire harnesses: State of the art, current challenges and fields of action to reach a higher level of automation', *Procedia CIRP* **107**, 653–660. Leading manufacturing systems transformation – Proceedings of the 55th CIRP Conference on Manufacturing Systems 2022.
**URL:** *https://www.sciencedirect.com/science/article/pii/S2212827122003250*

Ran, L., Junfeng, W., Haiying, W. & Gechen, L. (2010), Design method of can bus network communication structure for electric vehicle, *in* 'International Forum on Strategic Technology 2010', IEEE, pp. 326–329.

Scholz, A., Hsiao, T.-H., Juang, J.-N. & Cherciu, C. (2018), 'Open source implementation of ecss can bus protocol for cubesats', *Advances in Space Research* **62**(12), 3438–3448.

Semiconductors, N. (2024), 'Tja1050 high speed can transceiver'.
**URL:** *https://www.nxp.com/docs/en/data-sheet/TJA1050.pdf*

STMicroelectronics (2024*a*), 'Automotive microcontrollers'.
**URL:** *https://www.st.com/en/automotive-microcontrollers.html*

STMicroelectronics (2024*b*), 'Stm8af series - pdf documentation'.
**URL:** *https://www.st.com/en/microcontrollers-microprocessors/stm8af-series/documentation.html*

Systems, E. (2024), 'Esp-idf v3.3 - can api reference'.
**URL:** *https://docs.espressif.com/projects/esp-idf/en/release-v3.3/api-reference/peripherals/can.html*

Technology, M. (2024), 'Mcp2515 family data sheet'.
**URL:** *https://ww1.microchip.com/downloads/aemDocuments/documents/Family-Data-Sheet-DS20001801K.pdf*

# V. Appendix

## A. Matlab code used to process signal data

```
LA0  = [12,12.8,11.2];
L47  = [8,7.2,7.2];
L94  = [7.2,6.4,6.4];
L141 = [5.6,5.6,4.8];

% Define cable lengths
cable_lengths = [0, 47, 94, 141]; % Cable lengths in meters

% Define arrays of voltage peak-to-peak values for each cable length
voltage_peak_to_peak_0m = [12,12.8,11.2]; % Array of voltage peak-to-peak values for 10 meters
voltage_peak_to_peak_47m = [8,7.2,7.2]; % Array of voltage peak-to-peak values for 20 meters
voltage_peak_to_peak_94m = [7.2,6.4,6.4]; % Array of voltage peak-to-peak values for 30 meters
voltage_peak_to_peak_141m = [5.6,5.6,4.8]; % Array of voltage peak-to-peak values for 40 meters

% Calculate the average voltage peak-to-peak values
voltage_peak_to_peak_average = [
    mean(voltage_peak_to_peak_0m), ...
    mean(voltage_peak_to_peak_47m), ...
    mean(voltage_peak_to_peak_94m), ...
    mean(voltage_peak_to_peak_141m)
];

% Calculate attenuation assuming 10m is the input voltage
attenuation = voltage_peak_to_peak_average / voltage_peak_to_peak_0m(1);

% Calculate attenuation in dB
attenuation_dB = 20 * log10(voltage_peak_to_peak_average / voltage_peak_to_peak_0m(1));

% Plot the graph
figure;

% Plot voltage peak-to-peak values on the left y-axis
yyaxis left;
plot(cable_lengths, voltage_peak_to_peak_average, '-o', 'LineWidth', 1.5); % Plot the voltage peak-to-peak values
xlabel('Cable Length (m)'); % X-axis label
ylabel('Voltage Peak-to-Peak'); % Left y-axis label
xticks(cable_lengths); % Set x-axis tick positions
grid on; % Show grid

% Add a trend line for the voltage peak-to-peak values using only the last 3 data points
hold on;
last_three_lengths = cable_lengths(2:end); % Last three cable lengths
last_three_averages = voltage_peak_to_peak_average(2:end); % Last three average voltage peak-to-peak values
p = polyfit(last_three_lengths, last_three_averages, 1); % Fit a linear polynomial to the last three points
f = polyval(p, cable_lengths); % Evaluate the polynomial over all cable lengths for the trend line
plot(cable_lengths, f, '--k', 'LineWidth', 1.5); % Plot the trend line in black

% Calculate where the trendline intercepts the x-axis for a specific y-value (arbitrary y-value of 0)
y_value = 0.5;
intercept_x = (y_value - p(2)) / p(1); % Calculate the x-value where y equals the arbitrary value (0 in this case)

% Display the intercept value under the trendline
text(50,polyval(p,50)- 1,sprintf('0.5v at %.2fm',intercept_x), 'HorizontalAlignment','left','VerticalAlignment','top','Margin',5);

hold off;

% Plot attenuation in dB on the right y-axis
yyaxis right;
plot(cable_lengths, attenuation_dB, '-o', 'LineWidth', 1.5); % Plot the attenuation in dB
ylabel('Attenuation (dB)'); % Right y-axis label

title('Signal Attenuation Increase Over Distance'); % Title
```