Interfacing a Throttle Pedal with CAN for Roadrunner's Solar Car

Stefan Susic, S164352, Mirac Ali Sözlü S194037, Anas Alam, S194016

Abstract*— *For DTU's solar car project a system is designed and implemented for interfacing a throttle pedal with the car's CAN bus. Two PCB's have been designed and tested, a transmitter PCB which sends the signal from the pedal onto the CAN bus, and a receiver PCB which reads the message from the CAN bus and outputs a signal corresponding to how much the pedal is pressed. Special emphasis was placed on fault-safe mechanisms and reduced power consumption.

The total power consumption (of both transmitter and receiver) was reduced from 1.032 W to 0.708 W by sending MCU's in sleep. A rate-limiter was implemented to make sure the speed reference does not have sudden and rapid changes. One fault-safe mechanism was implemented in case of transmit or receive errors on the bus.

A PWM signal whose duty cycle corresponds to how much the pedal is pressed is low passed filtered, using a first order RC filter, to obtain the the final output. Due to the time constant of the RC filter (τ= 0.2s) the current design may limit the signal too much, whether this is the case is best tested by making real world test on the car itself. If the filter's time constant proves to be too much, an alternative is to use a DAC.

The used throttle pedal has two outputs, by comparing the two outputs a fault safe mechanism could be implemented in case of failures in the pedal. However this could not be implemented due to not being able to measure the second output.

I. Introduction

DTU Roadrunners is a project developing a car powered from solar cells. The goal is to reduce the number of recharges required by an electric car. Recharge-time for electric cars makes it difficult to travel long distances, this is one of the reasons to why people hesitate to buy electric cars.

The car will join the Bridgestone world solar challenge 2023, which is a competition located in Australia, where universities and companies will compete about which solar powered car will perform the best. The best car will be chosen from different parameters, like how many recharges the car needs, how many people can be in the car and how fast the car will finish the given distance.

The car utilizes a CAN bus (Controller Area Network) to communicate with various devices on the car. CAN is a communication protocol, which especially is used in the car industry. In this paper a system is presented for interfacing a throttle pedal with the car's CAN bus.

II. Problem Statement

Because the throttle pedal is a safety critical component of the car there is focus on safety and fail safe mechanisms. Because the car is powered from solar-cells there is major focus on reducing power consumption, without impacting performance. Below are the specific aspects of the project

- How can a system be designed and implemented that will send the throttle pedal's signal to the motor driver, utilizing CAN bus
- How does the system prevent sudden changes in the speed reference from the throttle pedal
- How can the power consumption of the system be optimized?
- How can the system be secured against electrical and mechanical faults?

III. Overview

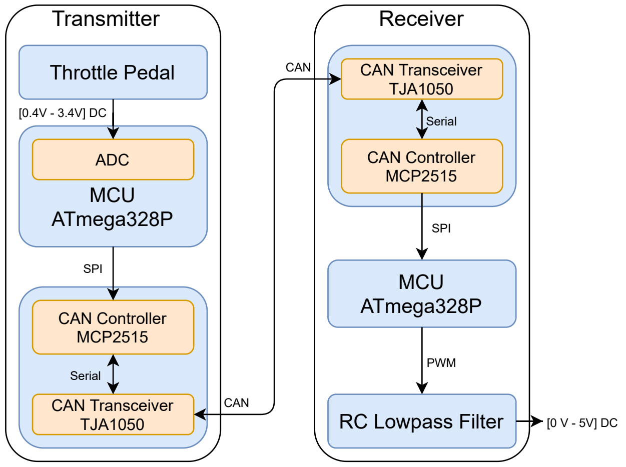


Figure 1. Block diagram showing the signal path from the throttle pedal to output signal going to motor driver

Figure 1 is a block diagram showing the final system (excluding how each component is powered). The system consists of two main parts, a transmitter PCB and receiver PCB. The transmitter PCB samples the signal from the throttle pedal and sends it to the CAN bus. The receiver reads the signal from the CAN bus and outputs a signal in the range $[0V, 5V]$ corresponding to how much the throttle pedal is pressed. This range can be changed in software as will be discussed later.

IV. Power

All nodes/devices on the solar car are powered from $12V$. Since the MCU, CAN transceiver and controller all require $5V$ a circuit for converting $12V$ to $5V$ must be implemented. To do this a linear voltage regulator is used. The regulator part for both receiver and transmitter circuits has a standard Capacitor regulator design, which provides a steady 5 volt DC signal. The used capacitors has values of $220\mu F$ and $22\mu F$, which are provided by electrolytic capacitors and 2 ceramic capacitors with the value of $100nF$. The circuit diagram is shown in next figure:

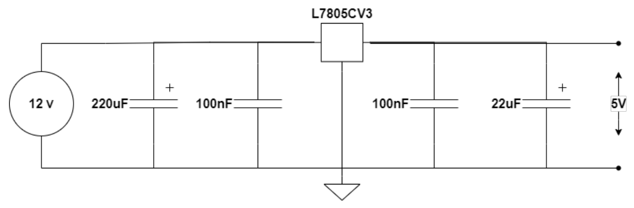


Figure 2. $5V$ regulator

Another option is to use a SMPS-topology, however linear voltage regulators have low cost with respect to these. They cover less space and offer steady output voltage. They also have very little output noise, which is desired in vulnerable circuits.

V. Pedal

The throttle pedal used in the project is from Ford. It has six pins and provides two outputs, one analog output in the range $[0.4V, 3.4V]$, and another PWM output.
Unfortunately the PWM output could not be measured in spite of driving the speeder according to our technical instructor and a schematic found online for a Ford model car [3, page 49]. While the system still can be realized without the PWM signal, the absence of it means that a fail-safe mechanism cannot be implemented for the pedal.
By having two output signals from the pedal the MCU could monitor and compare the two signals, if one signals is significantly different from the other it can be concluded a failure has occurred. After detecting a failure the MCU can fx. override the pedal signal and output a signal corresponding to zero speed until it detects the two signals are equal

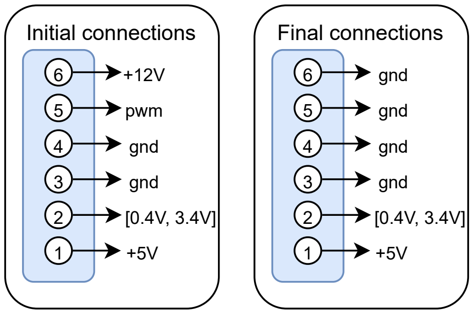


Figure 3. Left: Initial connections as shown in technical reference for Ford car [3, page 49], Right: The final connections

VI. CAN - Controller Area Network

CAN, Controller-Area-Network is a serial multi-master communication protocol. It uses differential signals: CAN-high and CAN-low. These two signals can can be in two states, dominant and recessive. In dominant state $CANH \geq CANL$ and in recessive state $CANH \leq CANL$. The wires are a twisted pair with a nominal characteristic impedance of 120Ω. Logic 0 is encoded by a recessive state and logic 1 is encoded by a dominant state.

Various versions of the CAN protocol exitsts, most notably are high-speed CAN which supports speeds up to $1\frac{MBit}{s}$, and low speed which supports speeds up to $125\frac{KBit}{s}$. The specification also defines two message formats, standard and extended format. The standard format allows 11 bit identifiers where the extended allows for 29 bit identifiers [2, chapter 1]. The solar car uses low-speed CAN at $125\frac{Kbit}{s}$ and standard format.

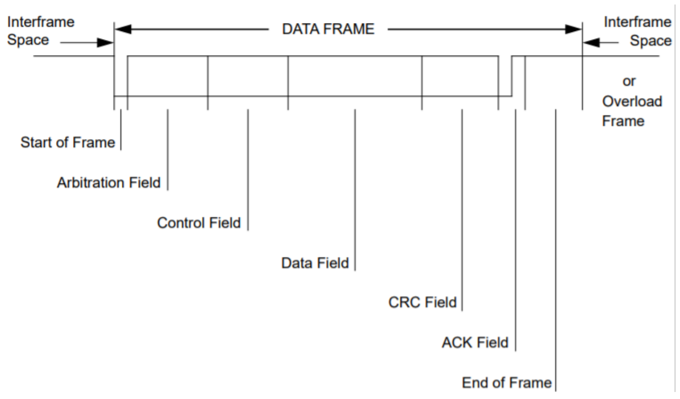


Figure 4. CAN message protocol, figure taken from [2, page 10]

The specification defines five different error types during communication, which can be detected by the CAN-controller [2, chapter 6]:

1) Bit Error: Because a transmitted message is received by all nodes on the bus (also the transmitter node), the transmitter can monitor if a bit is different that the bit that was sent; if that is the case an error is detected
2) Stuff Error: The specification says that six consecutive bits of same logic level is an error. The controller will insert a stuff bit after five consecutive bits with same level, but if this stuff bit is not sent an error is detected
3) CRC Error: Both transmitter and receiver calculates a CRC (cyclic redundancy checksum) for the data sent and received respectively. If the calculated CRC's are not equal an error is detected
4) Form Error: The message protocol has a number of bits that are fixed or reserved, if any of those fixed bits do not have the correct level an error is detected
5) Acknowledgment error: If a dominant state is not detected by the transmitter in the ACK field an error is detected, this is simply a way of letting the transmitter know that a message has been received

By detecting these errors one can implement fault-safe mechanisms in case of errors on the bus.

A. Interfacing with CAN bus

To interface/talk with a CAN bus three components are required, a CAN-transceiver, CAN-controller and a MCU (in some cases the controller is built-in the MCU as a peripheral, however not in ours).

The transceiver is responsible for actually decoding any incoming message on the CAN bus or encoding a message which needs to be sent on the CAN bus. The controller is the middle link between the transceiver and MCU. The controller can generate interrupts whenever a message is received, only allow certain messages with certain ID and provide a standard serial protocol for easy interfacing with MCU.

This project uses a finished PCB with both tranceiver and controller. The transceiver is TJA1050 and controller is MCP2515. The transceiver uses asynchronous serial protocol to communicate, where TX is the data that is to be sent on the CAN bus and RX is the data received from the CAN bus.

The chosen controller MCP2515 uses SPI-protocol (serial peripheral interface) to communicate with MCU. SPI is a synchronous serial communication interface with one master and multiple slaves. MOSI, master-out-slave-in is data from master to slave and MISO, master-in-slave-out is data from slave to master, data is sampled at each rising or falling edge of the clock depending on the SPI mode. Lastly CS, chip-select selects which slave is currently active [4].

Fot this project there is only one slave which is the CAN controller and the master is the MCU. The used MCU (Atmega328P) has a peripheral for interfacing with a SPI-bus. The Atmega328P datasheet provides which IO pins can be used to interface with SPI-bus:

- Pin D11 - MOSI
- Pin D12 - MISO
- Pin D13 - Serial clock
- Pin D10 - CS

B. Setting up the CAN Controller

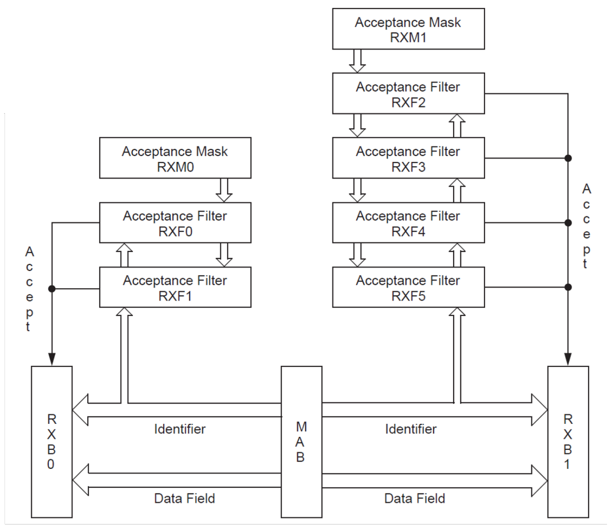
Instructions are sent to the controller via the SPI-bus. The instructions will set various bits in various registers to configure the controller. For this project the CAN controller is set-up to following:

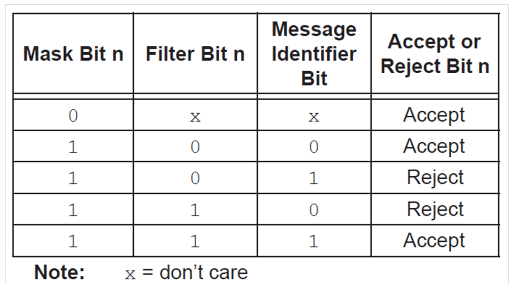
1) Transmit data at $125KBPS$
2) Transmitting one byte of data
3) Generate an interrupt (active low) whenever a message is received
4) Generate an interrupt (active low) whenever an error is detected
5) Only receive messages whose ID match the speeder ID

The controller has two receive buffers, $RXB0, RXB1$. The controller puts the received message in $RXB0$, in case there already is a message in $RXB0$ the message is rolled over to $RXB1$.
The controller will be set-up such that an interrupt is generated whenever a message is received in either $RXB0$ and $RXB1$. If this is not done some messages from the pedal may be missed in the final design because other messages coming from nodes with higher priority already are present in $RXB0$.

To only allow messages with a certain ID the controller provides some mask and filter registers for each receive buffer. Each bit in an incoming message is compared to the corresponding bit in the filter register, only if the two bits match *and* the corresponding mask bit is high will the message bit be accepted. If all message bits are accepted the message is inserted to one of the receive buffers [5, page 23].

Figure 5 shows two figures from the controllers datasheet, which explains the concept. In our case the filter will be applied to all bits, thus all bits in $RXM0$ and $RXM1$ are high. Acceptance filters $RXF0$ is used for the first receive buffer and $RXF2$ for the second receive buffer. The filter registers are set to be the chosen ID for the transmitter.





Mask Bit n	Filter Bit n	Message Identifier Bit	Accept or Reject Bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Note: x = don't care

Figure 5. Figures from MCP2515 datasheet explaining acceptance masks and filters. Figure taken from [5, page 25 and page 33]

As mentioned the CAN specification specifies five sources of error which should be detected by the CAN controller. The chosen controller can be set-up to generate an interrupt upon errors. The current set-up then has three sources of interrupts: from message received in $RXB0$, message received in $RXB1$ and upon any of the five types of errors earlier discussed. To check where the interrupt came from the controller has a register $CANINTF$ which holds various interrupt flags corresponding to the interrupt source [5, page 54].

VII. Software

The chosen microcontroller for this project is Atmega328P on Arduino Nano board. The main criteria for the MCU are:

- ADC, minimum 8-bit
- SPI interface

- PWM output or DAC
- Has a supported library for MCP2515 controller
- Can disable unused peripherals and go to sleep mode

Many microcontrollers fulfill these criteria but a too powerful MCU is not needed because it does not perform any heavy computations. In light of these criteria and the high amount of documentation around Arduino Nano it was selected. The Nano does not have any internal DAC, so to provide the final signal PWM and a low pass filter is used (alternatively an external DAC can be used).

A. *Transmitter*

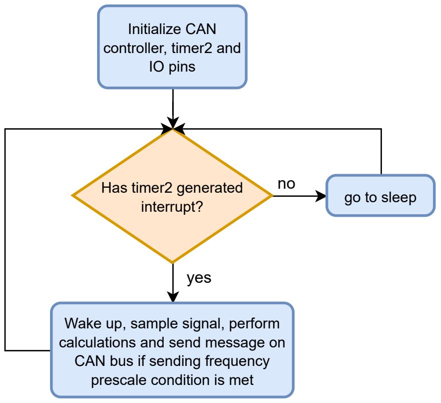

Initialize CAN controller, timer2 and IO pins
Has timer2 generated interrupt?
no
go to sleep
yes
Wake up, sample signal, perform calculations and send message on CAN bus if sending frequency prescale condition is met

Figure 6. Flow diagram for code on transmitter MCU

Figure 6 shows a general flow diagram of the code running on the transmitter MCU. To minimize power consumption and have a defined level for unused IO pins all IO pins are initialized as inputs with a pull-up resistor. Timer2 on MCU is initialized to generate interrupts at the desired sampling frequency.

To find the sampling frequency following method was used:

1) Probe the output of the speeder on an oscilloscope in single shot mode, and trigger on rising edge
2) Press the speeder down as hard as a person realistically would press it down; this gives a maximum (realistic) rate change, which can be measured on the scope
3) Define the sampling interval/period to be 10 times the amount of time it took to go from 0% to 100% on the speeder

Above method resulted in a sampling rate of $330Hz$.
To reduce power consumption the MCU goes in sleep when it is not running the interrupt handler. The Atmega328P has a variety of sleep-modes, for this case the power-save mode is used. This mode disables clock for CPU, flash, IO and ADC. Power-save mode has a variety of wake-up sources, one of which is timer2 [1, page 34].

The rate of the signal is calculated by:

$$\Delta y = \frac{y_1 - y_2}{\Delta t}$$

Where Δt is the sampling interval, and y_1 is the current value of the speeder signal and y_2 is the previous value. If the rate change is larger than the max-rate, which is defined in $volt/sec$ as a constant in software, the new value is calculated with following:

$$y_1 = \Delta y_{max} \cdot \Delta t + y_2$$

The maximum rate can easily be changed by simply changing a constant in the code.

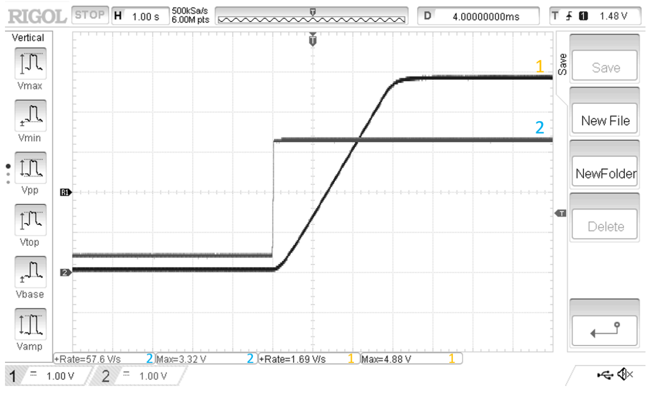


Figure 7. Rate limiter test, max rate is $1\frac{V}{s}$ channel 2 is input, channel 1 is output. Both channel has $1\frac{V}{div}$ and $1\frac{s}{div}$

Sending messages on the CAN bus with same frequency as the sampling rate might be too much, especially when there will be many other nodes on the bus. Therefore the sending frequency can be decreased with the help of a prescaler. The prescaler is also defined as a constant in software, and can easily be changed. The sending frequency is then $f_{send} = \frac{f_{sample}}{prescaler}$

The final output of the system is a DC signal which tells how much the pedal is pressed down, this signal will go to the car's motor driver. As mentioned the final output range is configurable in software, this is done by having two constants defining the lower and upper limit of the range. The sampled value is then mapped between these two ranges.

B. *Receiver*

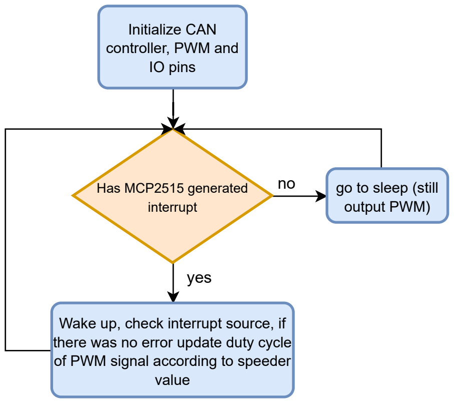

Initialize CAN controller, PWM and IO pins
Has MCP2515 generated interrupt
no
go to sleep (still output PWM)
yes
Wake up, check interrupt source, if there was no error update duty cycle of PWM signal according to speeder value

Figure 8. Flow diagram for code on receiver MCU

The receiver MCU outputs a PWM signal whose duty-cycle corresponds to how much the speeder is pressed down. Because this PWM signal will be low-pass filtered to get the final DC signal, the PWM signal should have as high a frequency as possible. To do this timer0 is set-up in fast-PWM mode on the MCU, where a frequency of $975Hz$ is achieved.

Figure 8 shows flow diagram for software on receiver MCU. The receiver uses a similar concept as the transmitter. The controller generates an interrupt whenever a message is received with correct ID, and the interrupt handler can read the data from the bus.

When the MCU is not in the interrupt handler it goes to sleep. Since the receiver needs to continuously output a PWM signal, the clock for IO should be active while in sleep. For this reason the MCU is put in idle sleep-mode. In this mode only clocks for CPU and flash are deactivated. The wake-up source is an external interrupt (coming from CAN controller) on pin D2 [1, page 34].

In the interrupt handler the duty cycle for PWM signal is updated. As mentioned the controller has three interrupt sources, the PWM signal must only be updated if the source of the interrupt is from a received message in either of its two receive buffers, if that is not the case, the interrupt triggered due to an error and the duty cycle is set to 0.

By sending the MCU's in sleep mode whenever they are not transmitting or receiving a message the total power consumption is reduced from $1.032W$ to $0.708W$.

VIII. Output Filter

The filter is the final part of the system. It takes as input the PWM signal, and outputs the DC component of the PWM signal. The PWM signal has a DC component because it is not centered around x-axis, and instead starts form 0V. It is a 1.st order RC filter. The main drawback of using such a filter is the high rise time associated with it. Because the signal has to be rate-limited anyway, the rise time is not a problem. Therefore the RC filter will act as a rate-limiter whose maximum rate of change is determined by its time constant. The filter's time constant is selected such that the it will determine the maximum allowed rate-change, then the software rate-limiter acts as a secondary rate-limiter which gives more fine control.

The cut-off frequency of a first order RC low-pass filter is given by

$$f_c = \frac{1}{2\pi RC}$$

The time constant tells how long it takes for the capacitor to reach 63.2% of the final value. The time constant of a series resistor and capacitor is RC.
Thus a higher time constant gives a longer rise time, but a too small time constant will increase the cut-off frequency and thus the signal will have low frequency noise.

Following values for R and C to give a reasonable balance between time constant and cut off frequency.

$$R = 4.32K\Omega \quad C = 47\mu F$$

These value gives following parameters:

$$\tau = 0.203s$$

$$V_{ripple} = 230mV (20 \text{ MHz BW-limited scope})$$

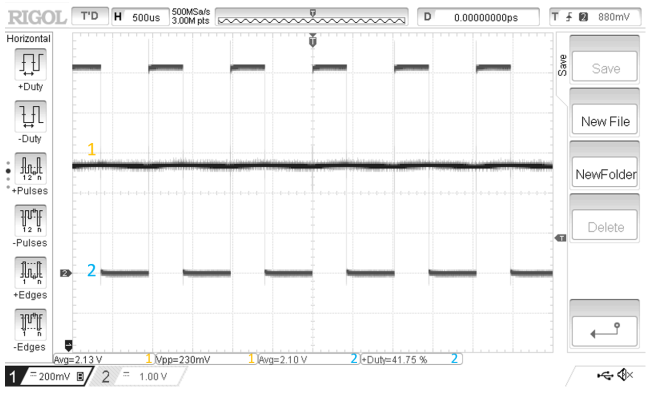


Figure 9. Channel 2: Input to filter ($1\frac{V}{div}$, $500\frac{\mu s}{div}$), Channel 1: Output from filter ($200\frac{mV}{div}$, $1\frac{s}{div}$)

Figure 10 shows the output signal (yellow) and the input signal (blue) where the software rate-limiter is disabled.

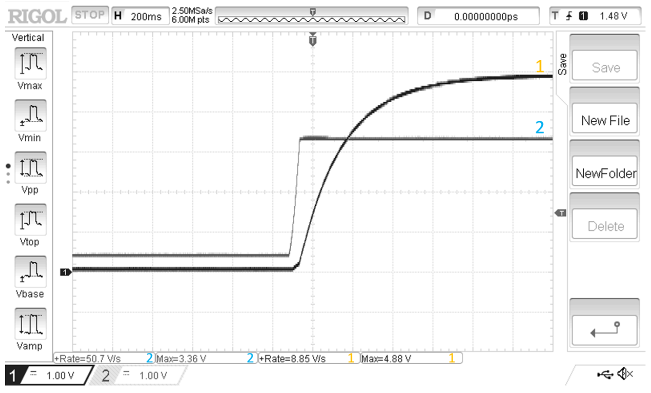


Figure 10. Channel 2: Input from pedal, Channel 1: Output from the system Both channels have $1\frac{V}{div}$, $200\frac{ms}{div}$

The figure shows an undesirable effect of the RC filter, which is the non-linearity, due to charging of the capacitor. If the software rate-limiter is set such that the signal will reach 63.2% in *more* time than the filter time constant, one would see more linearity, because the output would be overridden by the rate-limiter in software.

It could be the case that having a time constant of $0.2s$ is too much, this is best tested by using the system on a test-drive once the car is ready. If this is the case (that $0.2s$ is too much) one could decrease either R or C to get a smaller time constant, however this should be done while monitoring the noise on the output signal. Alternatively and external DAC should be used.

IX. PCB

After developing a prototype on both breadboard and perfboard, the process of designing a PCB began. By designing and using a PCB, allows the circuit to be more compact, as well as keeping the signal stable and less impacted by interference. Since the entire circuit consists of a transmitter and a receiver, two designs were developed. The circuits themselves are fairly simple, so a two-layer PCB would suffice. Furthermore, the orientation and placements of the components, were set as such to minimize area for the linear regulator, while having the USB connector and screw-terminal for the Arduino and CAN-bus, respectively, facing outwards to allow connection once a protective 3D printed box is placed around it. Due to the chosen orientation, some of the traces would overlap due to the pin configuration, which was solved by having certain traces run on the back side, which can be seen on figure 11. Another thing that becomes apparent on figure 11, is the placement of a lot via's. These are grounded via's connecting top to bottom side, and are placed to decrease interference, as well as decreasing the impedance of the traces.

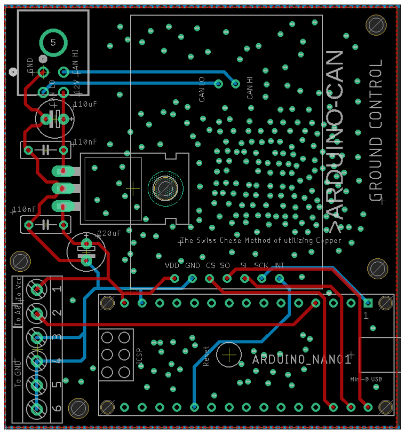


Figure 11. Screenshot of board design for the transmitter circuit. red traces are on the top-layer, blue traces are on the bottom-layer. The several green dots are grounded via's.

The dimensions of the PCB's are '$57.40mm \times 61.50mm$' for the transmitter PCB and '$52.10mm \times 70.60mm$' for the receiver PCB. Compared to the size of our perfboard size measuring '$45.00mm \times 108.00mm$' for each of the circuit parts, it seems that the PCB option is much more compact. One modification made to the used PCB, was to connect the V_{in} pin to the $5V$ pin, by soldering a trace on. This has been done due to the Arduino having an internal regulator, which causes the $5V$ supply to drop to $4V$. By supplying the chip through $5V$ pin the internal regulator of the Arduino is bypassed.

When first tested on the perfboard, the Arduino would be constantly connected by USB port, and the Arduino would select the $5V$ supply from the USB instead. There exists also the solution to order a new PCB, where the supply connects the $5V$ pin, which will omit the internal regulator.

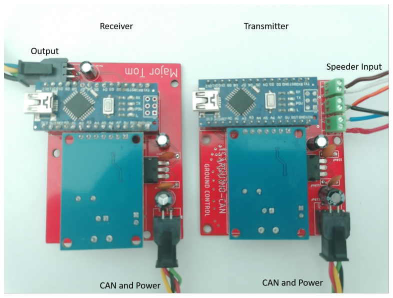


Figure 12. Final PCBs

X. Conclusion

The goal of this project was to design and implement a system a throttle pedal system for DTU's solar car project. The signal had to be sent over CAN-bus, and special focus was on reducing power consumption and implementing faul-tolerant mechanisms.

Such a system has been constructed, furthermore the power consumption has been reduced from $1.032W$ to $0.708mA$ and one fail-safe mechanism which override the throttle signal in case of errors on the CAN-bus was implemented.

That being said some improvements must be implemented before implementing this system on the final car. These are:

- Have a minimum of two outputs from the throttle pedal such that each output can be monitored and detect failure in the speeder if the two outputs differ
- Using a DAC instead of a low-pass filtering a PWM signal. This is not necessary. But the current design is rather limiting due to the $0.2s$ time constant of the output filter. Whether the current design is sufficient will have to be tested on the final car
- Use individual IC's for CAN- transceiver and controller
- The ADC on Atmega328P has 10-bit resolution, but currently it is scaled down to an 8-bit value. Had timer1 been used on the receiver MCU all 10-bits could be utilized, because timer1 is a 16-bit timer. This would give increased resolution on the output

References

[1] *Atmega328P datasheet*. Atmel, 2015. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[2] *CAN specification*. Version 2. Bosch, 1991. URL: http://esd.cs.ucr.edu/webres/can20.pdf.

[3] *entire_schematics_ph8m5t-700000-ad*. Ford, 2010. URL: https://www.fordownersclub.com/applications/core/interface/file/attachment.php?id=42087.

[4] *SPI Interface Specification*. Mouser, 2005. URL: https://www.mouser.com/pdfdocs/tn15_spi_interface_specification.PDF.

[5] *Stand-Alone CAN Controller with SPI Interface*. Microchip, 2003. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf.

Projektplan
31015 Fagprojekt Elektroteknologi

s194016 - Anas
s194037 - Mirac
s164352 - Stefan Susic

Marts 2021

Gruppe 11 - MEK02
Speederpedal til Solbil

Introduktion til emnet

DTU Roadrunners kører et projekt med at bygge en elektrisk bil der er drevet af solceller. Målet er at reducerer antallet af opladninger for en elektrisk bil, og dermed give en længere rækkevidde uden opladninger. Opladningstiden for en elektrisk bil gør det svært at køre lange ture, og er et af årsagerne til at mange folk tøver med at købe elektriske biler.

Bilen skal deltage i Bridgestone World Solar Challenge 2023, som er en konkurrence der afholdes i Australien, hvor universiteter og virksomheder dyster om hvis solbil er bedst. Den bedste bil bestemmes ud fra en score der beregnes fra en række forskellige parametre, såsom hvor mange opladninger bilen skal bruge, hvor mange personer der kan sidde i bilen og hvor lang tid det tager at dække distancen.

Bilens enkelte dele bruger en CAN bus (Controller Area Network) til at kommunikere med hinanden. CAN er en kommunikations protokol der specielt anvendes i bilindustrien. I dette projekt skal der arbejdes med bilens CAN bus.

Problemformulering

Formålet med dette projekt er at danne et system for en speeder pedal til solbilen. Speederen skal sende sit signal over bilens CAN bus til bilens motor driver. Da pedalen regulerer hastigheden for bilen, er der visse aspekter der vil sættes fokus på. Derfor er det største fokuspunkt for dette projekt, hvordan batteriforbruget kan mindskes, uden at det påvirker bilens evne til at køre optimalt. Et andet fokuspunkt vil være sikkerheden. Derfor vil der udbygges en sikkerhedsklausul, der sikre at signalet kan afbrydes fra motoren, i tilfælde af usikre forhold.

De konkrete aspekter der vil undersøges og implementeres:

- Hvilke speeder teknologier findes der? Og hvilken er mest robust og sikker?
- Hvordan kan der implementeres et system der sender speederens signal til bilens motor driver, ved hjælp af bilens eksisterende CAN bus?
- Hvordan kan der sørges for at motoren ikke får pludselige hastighedsændringer?
- Hvordan kan strømforbruget af systemet optimeres?
- Hvordan kan systemet sikres mod elektriske og mekaniske fejl?

Afgrænsning af Projektet

For dette projekt, er der kigget på den overordnet funktion af en speederpedal, hvoraf der er taget beslutning til hvilke felter og fokuspunkter der er mest relevante for udviklingen af speederpedalen. Således er dette indsnævret til to fokuspunkter, som vil beskrives følgende:

Det første fokuspunkt, blev naturligvis besluttet på baggrund af projektets hovedmål, hvilket er en speederpedal for en elektrisk bil, opladt af solpaneler, der skal konkurrere i et langdistance løb. Derfor var det oplagt at fokusere på at gøre pedalen så effektiv som muligt. Da pedalen er et mellemled imellem batteriet og motoren, er det essentielt at funktionaliteten af hastighedsregulering bevares, i og med at systemets belastning på batteriet minimeres.

En stor afgrænsning vi laver er at vi ikke køber en rigtig speeder pedal. Årsagen til dette er at de speeder pedaler der findes fremstilles til specifikke bilmodeller, og de medfølger sjældent datablade. Derfor kan vi komme i en situation hvor vi bestille en speeder pedal og skal finde ud af dens detaljer uden et datablad, dette er ikke en god ide da vi har begrænset tid til projektet. Desuden har vi fundet mange speeder pedaler fra Kina, her er problemet at der er lang leveringstid hvilket vi ikke har tid til. Vores fokus er derfor på at bygge systemet således at når man en dag køber en speeder kan den nemt kobles til systemet under en række betingelser som vi vil definere senere.
Istedet for at købe en speeder vil vi enten bruge et normalt potentiometer, eller et hall effect rotary encoder. Dermed implementerer vi kun speeder sensoren, og ikke den mekaniske konstruktion af speederen. Vi vil undersøge fordele og ulemper ved potentiometer og hall effect rotary encoder og lave en vurdering af hvilken vil fungerer bedst i vores tilfælde.

Målgruppe

Der er to målgrupper til projektet

- Medstuderende der arbejder på ROAST. Dette omfatter medstuderende i elektro, software og mekanik gruppen
- Personer der kommer til at køre solbilen

For vores medstuderende der arbejder på ROAST er det vigtigt at vi leverer et robust system, som fremtidige studerende kan arbejde videre på. Derfor er det vigtigt at vi undervejs projektet husker på at i fremtiden er kan der være folk som skal opgraderer noget i vores system eller reparerer noget i vores system, derfor er dokumentation vigtigt. Endvidere er det vigtigt at vi opretholder god kodestil, dvs. kode der så vidt muligt selv forklarer hvad det gør, og de steder hvor dette ikke er tilfældet er koden forklaret i form af kommentarer. Af samme årsag skal de to PCB'er vi udvikler have en vis overskuelighed, én måde

hvorpå dette kan opnås er ved at have labels forskellige steder på PCB'et.
For de personer der kommer til at kører bilen kan vi ikke forvente at vedkommende er faglærte. Derfor er det vigtigt at speederen er intuitiv og virker som man ville forvente.

Løsningsmetode

Forneden er en liste med de specifikke ting der skal implementeres

1. Undersøg om det er bedst at bruge potentiometer eller hall effect rotary encoder
2. Speeder pedal hvis signal sendes på bilens CAN bus, dette kræver et PCB
3. Et PCB der læser fra CAN bussen og giver et analogt signal til bilens motor driver
4. Rate-limiter algoritme, der sørger for at der ikke kommer pludselige ændringer i hastighedsreferencen
5. Implementér fejlhåndtering i software

Specifikke trin for at implementere ovenstående

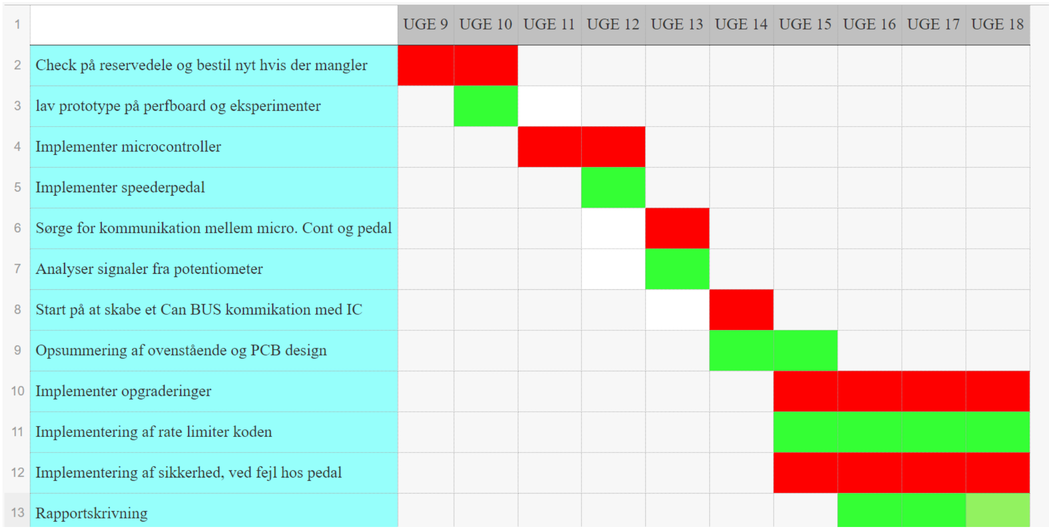
1. Undersøg om det er bedst at bruge potentiometer eller hall effect rotary encoder
   (a) Dette vil gøres eksperimentelt ved at dreje på begge sensorer og måle deres output. Ved at sammenligne en række parametre vil vi tage en beslutning om det resterende projekt vil dannes med potentiometer eller hall effect encoder. Disse parametre kan være: Linearitet, støj, mekanisk robusthed. I det resterende af løsningsmetoden bruger vi speeder sensor til at henvise til enten potentiometer eller hall effect encoder
2. Speeder pedal hvis signal sendes på bilens CAN bus, dette kræver et PCB
   (a) Først laves en prototype med breadboard og perfboard
   (b) Opsættelse af MCU, vi har valgt en arduino til vores projekt, og dets toolchain
   (c) Opsættelse af speeder sensor
   (d) Læs speeder sensors signal med MCU'ens indbyggede ADC
   (e) Opsættelse af CAN transceiver modul (MCP2515) med MCU, og lav nogle test skrivninger til og læsninger fra CAN bus
   (f) Send speeder sensor signal på CAN bussen
   (g) Tegn et kredsløbsdiagram og design PCB af dette

3. Et PCB der læser fra CAN bussen og giver et analogt signal til bilens motor driver
    (a) Først laves en prototype med breadboard og perfboard
    (b) Opsæt nyt par af MCU og CAN transceiver på samme måde som før
    (c) Læs speeder sensorens signal fra CAN bussen
    (d) Brug MCU'ens indyggede DAC til at sende analogt signal fra speeder sensor
    (e) Tegn kredsløbsdiagram og design PCB af dette
4. Rate-limiter algoritme, der sørger for at der ikke kommer pludselige ændringer i hastighedsreferencen
    (a) Undersøg nærmere om de rate-limiter algoritmer der findes
    (b) Lav pseudokode for algoritme
    (c) Implementer algoritmen på MCU'en
5. Implementér fejlhåndtering i software
    (a) Implementer flere speeder sensorer, således man kan sammenligne signalet fra hvert sensor
    (b) Implementering af fejl koder der sendes på CAN bussen, således at hvis speederen fejler ved resten af bilen det også

Ressourcer

- PCB perfboard:
  PCB Ali express
  PCB RS
- Can BUS transiever IC: CAN Bus RS RS
- Male headers:
  Male Header Ali Express
- PCB connectors:
  PCB Connector AliExpress
- resistor:
  Resistor AliExpress
- LED: LED AliExpress
- Rotary encoder: rotary encoder digikey

Aktivitetsplan



1		UGE 9	UGE 10	UGE 11	UGE 12	UGE 13	UGE 14	UGE 15	UGE 16	UGE 17	UGE 18
2	Check på reservedele og bestil nyt hvis der mangler	🟥	🟥								
3	lav prototype på perfboard og eksperimenter		🟩								
4	Implementer microcontroller			🟥	🟥						
5	Implementer speederpedal				🟩						
6	Sørge for kommunikation mellem micro. Cont og pedal					🟥					
7	Analyser signaler fra potentiometer					🟩					
8	Start på at skabe et Can BUS kommikation med IC						🟥				
9	Opsummering af ovenstående og PCB design						🟩	🟩			
10	Implementer opgraderinger							🟥	🟥	🟥	🟥
11	Implementering af rate limiter koden							🟩	🟩	🟩	🟩
12	Implementering af sikkerhed, ved fejl hos pedal							🟥	🟥	🟥	🟥
13	Rapportskrivning								🟩	🟩	🟩

Figur 1: Tidsplan

Referenceliste

- Jae-Yong Lee, ”A novel design of active accelerator pedal using linear electromagnetic actuator”, Journal of Mechanical Science and Technology, 2010
- Ravishankar Rao, ”battery Optimization cs Energy Optimization: Which to choose and when”, 2005 International Conference on Computer-Aided Design
- Warren Gay, ”CAN Bus”, Beginning Stm32, Chapter 18 ”CAN Bus”, 2018
- Lukas Cronje/Louise Sternholdt-Sørensen, ”Implementation of CAN bus in a concept solar car”, DTU 2020
- Daler Rakhmatov, ”Energy Management for Battery-Powered Embedded Systems”, Embedded Computing Systems Vol.2 No.3, 2003