VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

IMPLEMENTACE CAN BUS KOMUNIKACE
IMPLEMENTATION OF CAN BUS COMMUNICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Ján Vavrovič
AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Spáčil
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadaní bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Ján Vavrovič**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**Ing. Tomáš Spáčil**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Implementace CAN bus komunikace

Stručná charakteristika problematiky úkolu:

CAN komunikace je v průmyslovém prostředí hojně využívaný standard. Cílem této práce je vytvořit vzorovou úlohu pro další potenciální zájmce o tuto technologii. Pro zajištění co nevětší škály aplikací je cílem první fáze práce zprovoznění převodníku SPI CAN a posléze realizace CAN komunikace na samostatném mikrokontroléru.

Cíle bakalářské práce:

1. Zprovoznění modulu s MCP2515 pro CAN komunikaci na Rasberry Pi.
2. Vyčítání CAN zpráv pomocí mikrokonrtroleru dsPIC33FJ128MC804 a přeposílání vybraného obsahu na UART.
3. Vytvoření vzorové aplikace pro komunikaci Rasberry Pi s mikrokontrolerem (uC), kdy uC bude odpovídat předem definovanými zprávami na sadu vzorových zpráv.
4. Knihovna v C pro mikrokontroler dsPIC33FJ128MC804.

Seznam doporučené literatury:

VALÁŠEK, Michael. Mechatronika. Dot. 1. vyd. Praha: České vysoké učení technické, 1996. ISBN 80-01-01276-X.

ZÁHLAVA, Vít. Návrh a konstrukce desek plošných spojů. Vyd. 1. Praha: Česká technika - nakladatelství ČVUT, 2005. ISBN 80-01-03351-1.

SKALICKÝ, Jiří. Teorie řízení 1. Vyd. 1. Brno: Vysoké učení technické, 2002. Učební texty vysokých škol. ISBN 80-214-2112-6.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Podstatou této práce je vytvoření CAN sítě mezi Raspberry Pi a mikrokontrolerem dsPIC. Po zprovoznění modulů potřebných pro CAN komunikaci je vytvořena aplikace za účelem demonstrace provozu na sběrnici a také knihovna, která umožňuje jednoduché používání CAN periferie mikrokontroleru.

Summary

Essence of this thesis is to create CAN network between Raspberry Pi and microcontroller dsPIC. After initial setup of modules needed for CAN communication, an application is built that demonstrates traffic on the bus and a library is written that allows convenient use of microcontroller's CAN peripheral.

Klíčová slova

CAN, Raspberry Pi, mikrokontroler, dsPIC33FJ128MC804, MCP2515, jazyk C

Keywords

CAN, Raspberry Pi, microcontroller, dsPIC33FJ128MC804, MCP2515, C language

Bibliografická Citace

VAVROVIČ, J. *Implementace CAN bus komunikace*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2021. 39 s., Vedoucí bakalářské práce: Ing. Tomáš Spáčil.

Prohlašuji, že jsem bakalářskou práci „Implementace CAN bus komunikace“ vypracoval samostatně s použitím zdrojů, které jsou v práci citované a jsou uvedené v seznamu použitých zdrojů.

Ján Vavrovič

Brno . . . .

Děkuji vedoucímu práce Ing. Tomáši Spáčilovi a zastupujícímu vedoucímu Ing. Martinu Formánkovi za rady a pomoc při práci.

Ján Vavrovič

Obsah

1 Úvod

Tato práce se zabývá komunikací prostřednictvím sběrnice CAN. Komunikovat budou dvě zařízení, které je potřeba pro tuto komunikaci nastavit. Prvním zařízením je počítač Raspberry Pi, na kterém je nasazena rozšiřující CAN deska (HAT). Druhé zařízení je mikrokontroler dsPIC, který má přímo v sobě CAN periferii.

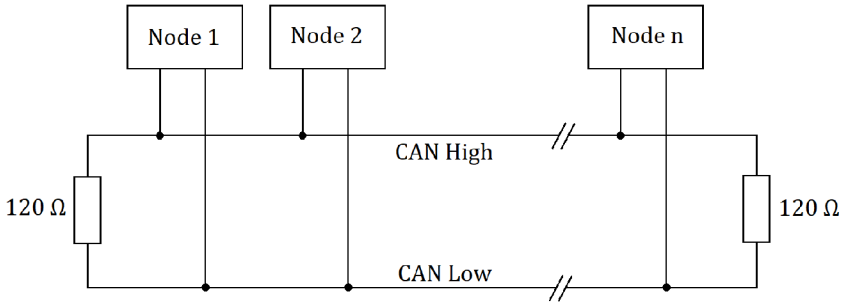
Na začátek je potřeba seznámit se s CAN protokolem – o jaký typ komunikace se jedná, z čeho se sběrnice skládá, jak jsou zařízení ke sběrnici připojené, jaké jsou pravidla pro vysílání a příjem, jaké jsou druhy zpráv, jaká je struktura zpráv, jak se nastavuje časování bitu a podobně.

Mikrokontroler dsPIC je umístěn na vývojové desce, na které jsou prvky jako potenciometr nebo akcelerometr. Po zprovoznění základní komunikace se předvede příklad přenosu údajů z těchto prvků přes sběrnici CAN.

Jednotlivé příkazy, kterými se nastavoval mikrokontroler, a kterými se realizovala komunikace, se pak zobecněně zapíšou do funkcí, z kterých se vytvoří knihovna.

2 CAN protokol

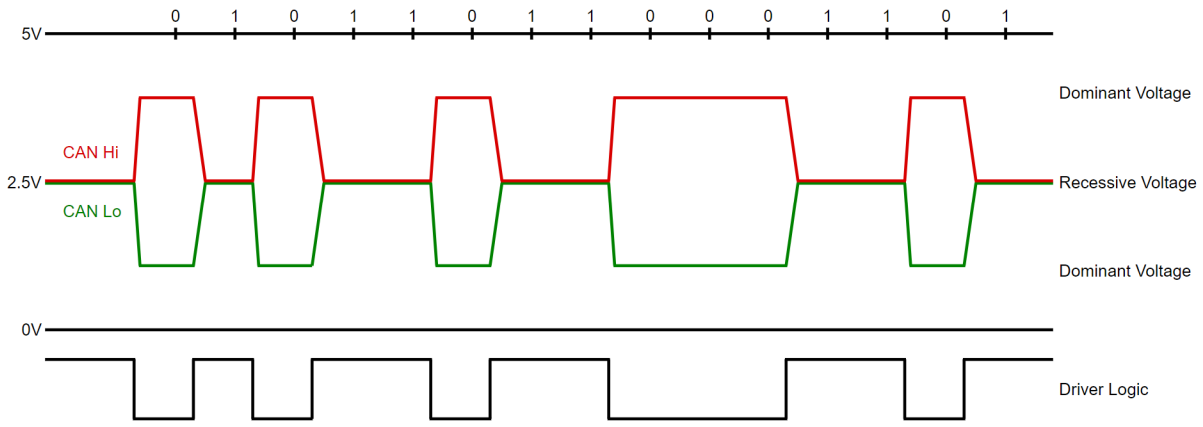
CAN (Controller Area Network) protokol byl představen v roce 1986 společností Bosch. CAN je vhodný hlavně do aplikací, kde je potřeba zajistit rychlou a bezpečnou komunikaci za přítomnosti zdrojů značného rušení. Použití sahá od osobních automobilů a nákladních vozidel přes vlaky, lodě, průmyslové stroje, medicinskou techniku až po výtahy a jeřáby. Nejvíc se používá ve zmíněném automobilovém průmyslu, kde je potřeba zajistit propojení jednotlivých elektronických prvků jako např. řídicí jednotka motoru, jednotky zajišťující stabilitu vozidla, senzory, jednotky ovládající světla, okna, dveře, klimatizaci a jiné. Použití CAN sběrnice zde zabezpečí poměrně malé množství potřebné kabeláže. V současné době se v jednom vozidle nachází i víc než jedna CAN síť. [1][2]



Obrázek 2.1: Nákres CAN sběrnice

Komunikační protokol CAN je popsaný v normách ISO 11898. Na obr. 2.1 je schematicky znázorněna CAN sběrnice. Všechny jednotky neboli uzly komunikují přes jeden kanál, který je tvořen kroucenou dvojlinkou. Ta se skládá ze dvou vodičů CAN High a CAN Low o charakteristické impedanci 120 Ω. Na sběrnici můžou být dvě logické úrovně: dominantní nebo recesivní. Lze si představit, že výstupy uzlů jsou zapojeny do logického členu AND. To znamená, že když alespoň jeden uzel vysílá logickou 0 (dominantní úroveň) bude na sběrnici logická 0. Logická 1 (recesivní úroveň) bude na sběrnici jen tehdy, když budou výstupy všech uzlů v logické 1. [1][3]

CAN komunikace je asynchronní a sériová. Na dvojlinku se přivádí diferenciální signál. Příklad signálu je zobrazen na obr. 2.2. Během recesivní úrovně je na obou vodičích stejné napětí, tedy rozdílové napětí je 0 V. Při dominantní úrovni se na vodiči CAN High napětí zvýší, na vodiči CAN Low se napětí sníží. Na lince tak vznikne rozdílové napětí. Na každém konci je linka zakončena 120 Ω rezistorem, který potlačuje odraz signálu od konce vedení. Po skončení dominantní úrovně tyto rezistory také zabezpečují vrácení napětí na recesivní úroveň. Standardně je maximální rychlost přenosu 1 Mbit/s. V případě protokolu CAN Flexible Data-Rate ale může být i vyšší. [1][3][5]



Obrázek 2.2: Signál na CAN sběrnici [4]

2.1 Pravidla vysílání

CAN sběrnice je multi-master, takže když je sběrnice volná, kterýkoliv uzel může zahájit vysílání. Vždy vysílá jen jeden uzel a všechny ostatní uzly ho poslouchají. Zpráva tedy obecně nemá jednoho konkrétního příjemce. [1]

Součástí zprávy je její identifikátor dlouhý 11 nebo 29 bitů. Tento identifikátor označuje, co je obsahem dané zprávy. Každý uzel pak rozhodne, jestli je daná zpráva pro něj podstatná nebo ne. Může se stát, že by chtělo začat vysílat víc uzlů najednou. V takovém případě o tom, kdo získá právo vysílat, rozhodne arbitráž. [3]

Arbitráž

Arbitráž je rozhodování o tom, kdo získá přístup k vysílání. Identifikátor, který se vysílá na začátku zprávy, zároveň slouží i na určení priority. Všechny uzly, které se podílí na CAN komunikaci, nepřetržitě sledují logickou úroveň na sběrnici. Platí to i pro uzel, který vysílá. Když uzel vysílá identifikátor a ním vysílaná úroveň se neshoduje s úrovní na sběrnici, přestane vysílat a stane se přijímačem. Tenhle uzel prohrál arbitráž. Po uplynutí všech 11 nebo 29 bitů identifikátoru zůstane vysílat právě jeden uzel. Tento uzel vyhrál arbitráž a pokračuje ve vysílání zbylých částí zprávy. Když jeho zpráva skončí, tak uzly, které prohrály arbitráž, se znovu pokusí o vyslání svojí zprávy. Jelikož se bity identifikátoru vysílají směrem od nejdůležitějšího bitu po nejméně důležitý a dominantní úroveň (logická 0) se vždy prosadí, tak platí, že čím má zpráva nižší hodnotu identifikátoru, tím má vyšší prioritu. [1][3]

2.2 Přenos zpráv

CAN zpráva má podobu rámce s předepsanou strukturou. Rámec CAN zprávy může mít jeden ze dvou formátů [1]:

- Standardní rámec (Standard Frame) – obsahuje 11 bitový identifikátor
- Rozšířený rámec (Extended Frame) – obsahuje 29 bitový identifikátor

CAN protokol rozlišuje čtyři typy rámců [1]:

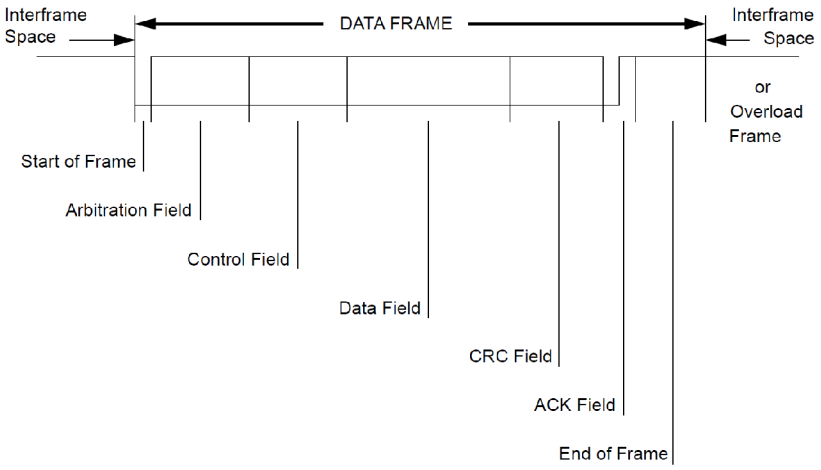
- Datový rámec (Data Frame)
- Žádost o rámec (Remote Frame)
- Chybový rámec (Error Frame)
- Rámec přetížení (Overload Frame)

Datový rámec

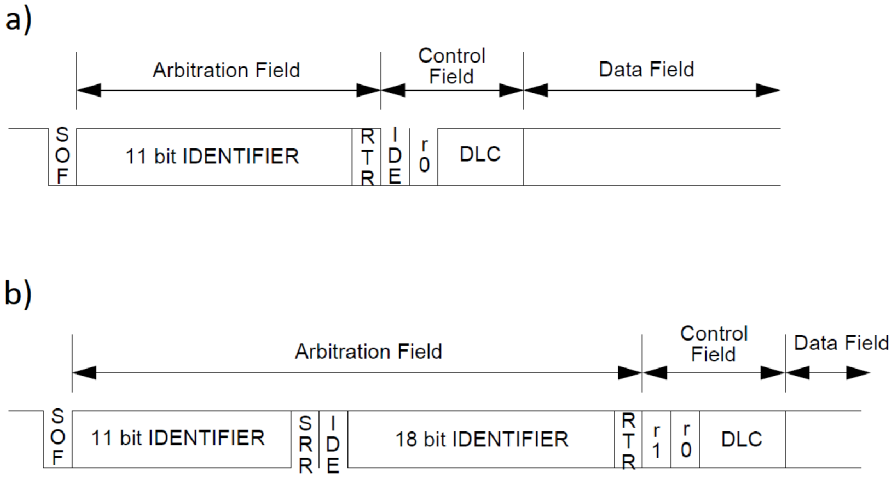
Datový rámec slouží k přenosu dat. Jeho struktura je znázorněna na obr. 2.3. Datový rámec obsahuje následovná pole [1]:

- Začátek rámce (Start of Frame) – jeden dominantní bit. Označuje začátek vysílání.
- Arbitrážní pole – liší se podle formátu:
  - Standardní formát (obr. 2.4a):
    - Identifikátor (Identifier) – 11 bitů
    - RTR (Remote Transmission Request) – dominantní bit pro datový rámec, recesivní pro žádost o rámec
  - Rozšířený formát (obr. 2.4b):
    - 11 bitový základní identifikátor – nejdůležitější bity 29 bitového identifikátoru (28-18)
    - SRR (Substitute Remote Request) – recesivní bit
    - IDE (Identifier Extension) – recesivní bit
    - 18 bitový rozšířený identifikátor – nejméně důležité bity 29 bitového identifikátoru (17-0)
    - RTR (Remote Transmission Request) – dominantní bit pro datový rámec, recesivní pro žádost o rámec
- Řídicí pole (Control Field) – liší se podle formátu:
  - Standardní formát (obr. 2.4a):
    - IDE (Identifier Extension) - dominantní bit
    - r0 – rezervovaný bit
    - DLC (Data Length Code) – 4 bity, které určují počet datových bytů
  - Rozšířený formát (obr. 2.4b):
    - r1 – rezervovaný bit
    - r0 – rezervovaný bit
    - DLC (Data Length Code) – 4 bity, které určují počet datových bytů
- Datové pole (Data Field) – přenášená data. Délka může být od 0 do 8 bytů.

- Pole CRC (Cyclic Redundancy Check Field) - kontrolní mechanismus na zjišťování chybně přenesených dat. Obsahuje:
  - Sekvence CRC (CRC Sequence) – výsledek CRC výpočtu
  - Oddělovač CRC (CRC Delimiter) – recesivní bit
- Pole ACK (Acknowledgement Field) – potvrzení o přijetí zprávy. Obsahuje:
  - ACK Slot – vysílač zde vyšle recesivní bit a uzly, které zprávu zaznamenaly, přepíšou tento bit na dominantní. Stane se tak nezávisle od toho, jestli zpráva prošla přes filtr příjemce nebo ne.
  - Oddělovač ACK (ACK Delimiter) – recesivní bit
- Konec rámce (End of Frame) – 7 recesivních bitů



Obrázek 2.3: Datový rámec [1]



Obrázek 2.4: a) Standardní formát, b) Rozšířený formát [1]

Žádost o rámec

Žádost o rámec vyšle uzel, který žádá o data od jiného uzlu. Struktura je shodná s datovým rámcem, s následujícími výjimkami [1]:

- RTR bit je recesivní, což značí, že se jedná o žádost
- Pole DLC odpovídá počtu bytů vyžádané datové zprávy
- Datové pole je zde vynecháno

Chybový rámec

Chybový rámec vyšle uzel, který zaznamenal chybu. Obsahuje dvě pole [1]:

- Příznak chyby (Error Flag) – může mít dvě formy:
  - Aktivní příznak chyby (Active Error Flag) – 6 po sobě vyslaných dominantních bitů. Vyšle je uzel, který zaznamenal chybu a je v aktivním stavu. 6 dominantních bitů naruší pravidlo vkládání bitů. Všechny uzly tedy zaznamenají chybu a začnou vysílat svoje příznaky chyby. Délka, kterou bude mít výsledný příznak chyby na sběrnici bude od 6 do 12 dominantních bitů.
  - Pasivní příznak chyby (Passive Error Flag) – 6 recesivních bitů. Vysílá je uzel, který zaznamenal chybu a je v pasivním stavu. Uzel chybu detekoval, ale nesděluje tuto informaci jiným uzlům.
- Oddělovač chyby (Error Delimiter) - následuje po ukončení příznaku chyby (ukončení sekvence dominantních bitů a přechod sběrnice do recesivní hodnoty). Obsahuje 8 recesivních bitů.

Rámec přetížení

Vyslání rámce přetížení můžou způsobit tři události [1]:

- Přijímací uzel není připravený na příjem nebo vysílání další zprávy. V tomto případě uzel začne vysílat rámec přetížení během prvního bitu mezery mezi rámci. Na zpoždění dalšího přenosu na sběrnici se můžou vyslat maximálně 2 takové rámce za sebou.
- Detekce dominantního bitu během mezery mezi rámci (kromě posledního bitu)
- Detekce dominantní hodnoty při posledním bitu oddělovače chyby nebo oddělovače přetížení.

Rámec přetížení obsahuje dvě pole [1]:

- Příznak přetížení (Overload Flag) – 6 dominantních bitů, tedy stejná forma jako aktivní příznak chyby
- Oddělovač přetížení (Overload Delimiter) – následuje po příznaku přetížení a podobně jako oddělovač chyby obsahuje 8 recesivních bitů

Rámec přetížení od chybového rámce odlišuje doba, kdy nastal. Rámec je vyhodnocen jako rámec přetížení, a ne jako chybový rámec, pokud nastal během mezery mezi rámci, na konci chybového rámce nebo na konci rámce přetížení.

2.3 Prostor mezi rámci

Chybové rámce jsou vysílány okamžitě, jak je chyba zaznamenána. Rámce přetížení nastávají v okamžicích popsaných na straně 13. Tyto dva typy rámců nejsou odděleny od předchozích rámců. V případě datového rámce a žádosti o rámec, uzel, který chce vysílat, vždy čeká na uplynutí jisté minimální doby od skončení předešlého vysílání. Tato mezera mezi rámci (Intermission) je tvořena třemi recesivními bity. [1]

2.4 Vyhodnocování chyb

Existuje pět událostí, které způsobí chybu, a tedy vyslání chybového rámce [1]:

- Bitová chyba (Bit Error) – nastane, když uzel zaznamená na sběrnici jinou hodnotu, než kterou vysílá. Výjimkou je ACK bit a pasivní příznak chyby.
- Chyba vkládání (Stuff Error) – nastane při zaznamenání šesti bitů stejné hodnoty po sobě
- Chyba CRC (CRC Error) – nastává, když se výsledek CRC výpočtu příjemce neshoduje s hodnotu, kterou obdržel ve zprávě od vysílače
- Chyba formátu (Form Error) – nastane, když v přijímané zprávě ty části rámce, které mají pevně danou logickou úroveň, obsahují nesprávnou úroveň
- Chyba potvrzení (Acknowledgment Error) – nastane, když během ACK bitu vysílací uzel zaznamená recesivní hodnotu. To značí, že nijaký uzel jeho zprávu nezaznamenal.

Dle počtu zaznamenaných chyb může být uzel v jednom ze tří stavů [1]:

- Aktivní (Error Active) – může vysílat, při zaznamenání chyby vyšle aktivní příznak chyby
- Pasivní (Error Passive) – může vysílat, při zaznamenání chyby vyšle pasivní příznak chyby
- Odpojený (Bus Off) – nemůže vysílat žádné zprávy

Zpočátku jsou všechny uzly v aktivním stavu. Detekování chyby způsobí inkrementování počítadla chyb podle pravidel popsaných v [1] na stranách 61-63. Když bude počítadlo větší než 127, uzel přejde z aktivního do pasivního stavu. Po přesáhnutí hodnoty 255 se uzel od sběrnice odpojí. [1]

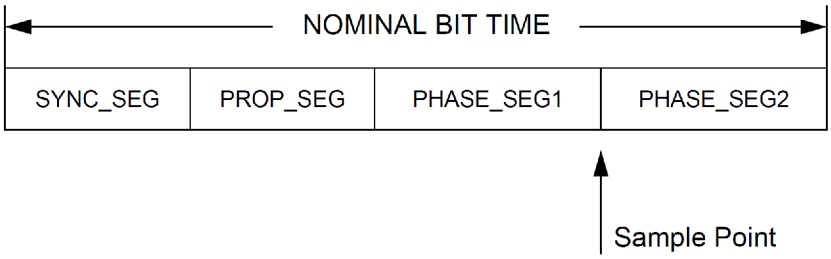
2.5 Vkládání bitů

Při vysílání na CAN sběrnici se používá NRZ (Non Return to Zero) kódování, tedy jednotlivé bity na sebe spojitě navazují. Aby byla zaručena synchronizace mezi vysílačem a příjemci, vkládají se do rámce dodatečné bity. Když vysílač zaznamená 5 vyslaných bitů stejné hodnoty, automaticky vloží jeden bit opačné hodnoty. Části rámce, které se takhle upravují jsou začátek rámce, arbitrážní pole, řídicí pole, datové pole a CRC sekvence. Do částí oddělovač CRC, pole ACK a konec rámce se bity nevkládají. Příjemce pak tyto vložené bity ze zprávy odstraní. [1]

2.6 Časování bitu

Bitový čas (Bit Time) je doba jednoho bitu na sběrnici. Je roven převrácené hodnotě přenosové rychlosti (Bit Rate). Jak je znázorněno na obr. 2.5, bitový čas se skládá ze čtyř částí [1]:

- Synchronizační segment (Synchronization Segment) – v této části je při správné synchronizaci hrana
- Propagační segment (Propagation Segment) – část, která kompenzuje zpoždění signálu
- Fázový segment 1 (Phase Segment 1) – část, která může být prodloužena při resynchronizaci
- Fázový segment 2 (Phase Segment 2) – část, která může být zkrácena při resynchronizaci



Obrázek 2.5: Části bitového času [1]

Vzorkovací bod (Sample Point) je okamžik, při kterém dojde k zaznamenání logické úrovně na sběrnici. Nastane hned po skončení fázového segmentu 1. Jeho polohu je potřeba vhodně nastavit. Signál se od jednoho uzlu k druhému dostane s určitým zpožděním, které způsobují vodiče a elektronika. Nastavení vzorkovacího bodu je důležité pro arbitraci, protože tehdy může vysílat víc uzlů najednou. Když nastane vzorkování příliš brzo, může se stát, že uzel vyčte sběrnici dřív, než se k němu dostane signál od jiného uzlu. Když nastane vzorkování příliš pozdě, může dojít zaznamenání následující úrovně. [1][6]

Celkový bitový čas je rozdělený na menší časové jednotky zvané časová kvanta. Každému segmentu se přiřadí doba v těchto jednotkách. Doba synchronizačního segmentu je vždy 1 časové kvantum, doba ostatních segmentů je programovatelná. [1]

2.7 Synchronizace

Při CAN komunikaci má každý uzel svůj vlastní oscilátor, není přítomen žádný společný hodinový signál. Frekvence oscilátorů mají konečnou přesnost, následkem čeho vznikají fázové chyby. Je tedy nevyhnutné zajišťovat synchronizaci mezi vysílačem a přijímači. [7]

V CAN protokolu jsou popsané dva druhy synchronizace [1]:

- Tvrdá Synchronizace (Hard Synchronization)
- Resynchronizace (Resynchronization)

Řekněme, že je sběrnice volná, a některý uzel zahájí vysílání. Jako první se vyšle bit „začátek rámce“. Nastane přechod z recesivní úrovně na úroveň dominantní. Při této hraně proběhne tvrdá synchronizace. Projeví se tak, že příjemce restartuje svůj bitový čas. Během vysílání při všech přechodech z recesivní úrovně na dominantní nastane resynchronizace. Dostatečný počet těchto přechodů zaručí vkládání bitů. Resynchronizace se projeví tak, že přijímač zkrátí svůj fázový segment 1 nebo prodlouží fázový segment 2. [1]

Příjemce může zaznamenat hranu ve třech místech svého bitového času:

a) v synchronizačním segmentu

b) za synchronizačním segmentem a před vzorkovacím bodem.

c) za vzorkovacím bodem předešlého bitového času a před synchronizačním segmentem aktuálního bitového času

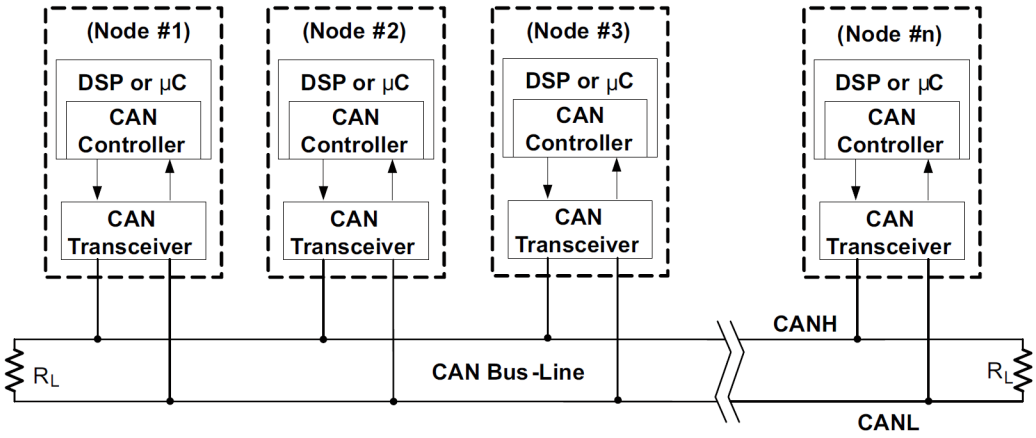
V případě a) resynchronizace není potřebná. V případě b) přijímač prodlouží svůj fázový segment 1. V případě c) přijímač zkrátí svůj fázový segment 2. Maximální hodnota, o kterou je možné zkrátit nebo prodloužit fázové segmenty se označuje jako šířka resynchronizačního skoku (Rasynchronization Jump Width). [1]

3 Použitý hardware a software

Součástí každého uzlu na CAN sběrnici je [8]:

- Mikroprocesor nebo mikrokontroler
- CAN řadič (CAN Controller) – implementuje CAN protokol
- CAN budič (CAN Transceiver) – spojuje uzel se sběrnicí, přizpůsobuje napěťově úrovně

Pro zvýšení odolnosti se může mezi řadič a budič přidat galvanický oddělovač, případně použít takový budič, který má už oddělovač v sobě integrovaný. [6]



Obrázek 3.1: Struktura uzlů [8]

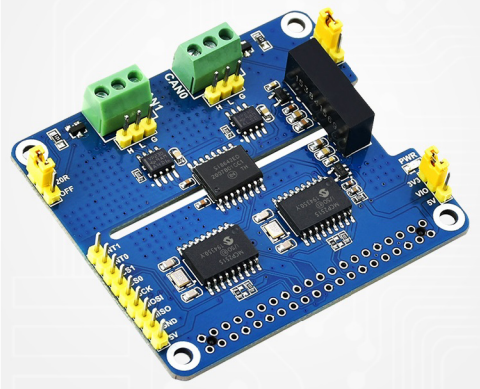
3.1 Raspberry Pi

První zařízení, na kterém se bude komunikace realizovat je jednodeskový počítač Raspberry Pi 3 Model B+ (obr. 3.2) od nadace Raspberry Pi Foundantion. Jeho základní parametry jsou [9]:

- Procesor: Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
- Paměť: 1GB LPDDR2 SDRAM
- Úložiště: Micro SD
- Komunikační rozhraní: 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2, Gigabit Ethernet, 4 × USB 2.0

- Audio a video rozhraní: HDMI, MIPI DSI display port, MIPI CSI camera port, stereo output, composite video port
- Další rozhraní: 40-pin GPIO
- Napájení: 5V/2.5A DC via micro USB, 5V DC via GPIO, Power over Ethernet

Obrázek 3.2: Raspberry Pi 3 Model B+ [9]



Obrázek 3.3: Rozšiřující deska [10]

K Raspberry Pi je připojena rozšiřující deska Waveshare 2-CH CAN HAT (obr. 3.3). Jedná se o HAT (Hardware Attached on Top). Deska se tedy umístí navrch počítače Raspberry Pi a komunikuje s ním přes GPIO piny. Komunikace probíhá prostřednictvím rozhraní SPI. Deska má dvě samostatné CAN jednotky. Součástí každé jednotky je [10]:

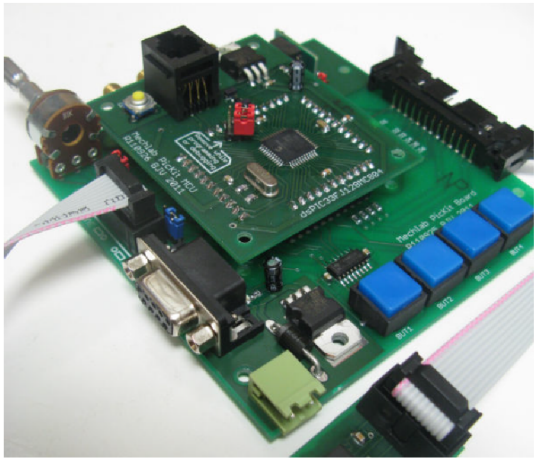
- CAN řadič MCP2515 od firmy Microchip
- CAN budič SN65HVD230 od firmy Texas Instruments

Každý řadič MCP2515 má vlastní krystal s frekvencí 16 MHz. Dále je na desce digitální oddělovač signálu Si8642ED-IS a DC/DC měnič B0505LS-1W. Počítač Raspberry Pi bude tedy galvanicky oddělen od CAN sběrnice. Na desce je také možnost přes propojku připojit zakončovací rezistor. [10]

3.2 Vývojová deska

Druhé zařízení, na kterém se bude realizovat CAN komunikace je vývojová deska (obr. 3.4) vyvinuta v mechatronické laboratoři na FSI VUT. Deska obsahuje mikrokontroler dsPIC33FJ128MC804 od společnosti Microchip. Základní parametry mikrokontroleru jsou [11]:

- Architektura: 16-bit
- Nejvyšší frekvence CPU: 40 MHz
- Programová paměť: 64 KB
- Paměť SRAM: 16 KB
- 8 DMA kanálů
- Napájení: 3 až 3.6 V
- 44 IO pinů
- Periferie: 2 x komparátor, A/D převodník 9x12-bit, 2 x D/A převodník 2x16-bit, 2 x UART, 2 x SPI, I2C, CAN modul, 12 PWM výstupů, Motor Control PWM, časovače 5 x 16-bit nebo 2 x 32-bit, 4 x Input Capture, 4 Output Compare kanály, 2 x Quadrature Encoder Interface

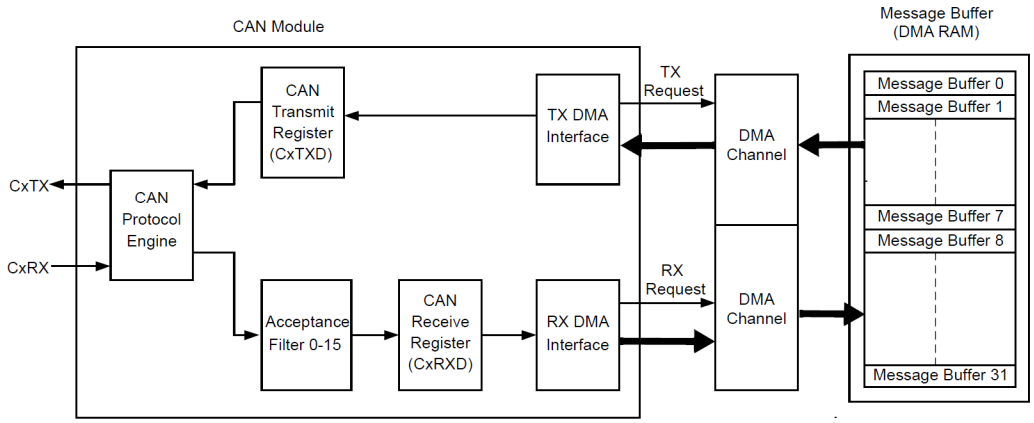


Obrázek 3.4: Vývojová deska

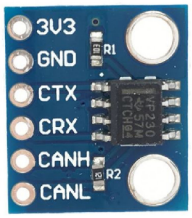
Na nahrávání a ladění programu mikrokontroleru dsPIC se použil MPLAB PICkit 4 Debugger.

Součástí mikrokontroleru je CAN periferie (obr. 3.5), která plní funkci CAN řadiče. Periferie má jen jeden vysílací a jeden přijímací registr. Jinak pracuje s DMA pamětí mikrokontroleru, kterou používá jak pro vysílání, tak pro příjem zpráv. [11]

Na buzení CAN sběrnice je použitý budič SN65HVD230 od firmy Texas Instruments (obr. 3.6). Spolu s ním je na malé desce i zakončovací rezistor.



Obrázek 3.5: Blokové schéma CAN periferie [12]



Obrázek 3.6: Budič SN65HVD230 [13]

Ve vzorové aplikaci se dále využívají tyto periferie mikrokontroleru:

- A/D převodník
- Output Compare modul (PWM)
- UART periferie
- SPI periferie

Kromě mikrokontroleru jsou na vývojové desce umístěny různé prvky. Ve vytvořené aplikaci se používají:

- LED diody
- D/A převodník DAC7715U
- Převodník UART na USB FT231X
- Tlačítka
- Potenciometr
- Akcelerometr MMA3204D

Dále byl k pinům na vývojové desce připojen:

- Bzučák
- DC motor s tranzistorem a diodou
- Voltmetr
- Joystick

3.3 Software

Pro Raspberry Pi je použitý operační systém Raspberry Pi OS. Tento operační systém je založený na linuxové distribuci Debian, optimalizované pro Raspberry Pi. Poskytuje ho přímo Raspberry Pi Foundation. [14]

Součástí linuxového jádra je soubor ovladačů a síťových nástrojů nazývaný Socket-CAN. Toto rozhraní umožňuje konfigurovat a používat CAN zařízení jako síťové zařízení. [15]

Dále je na Raspberry Pi nainstalovaný balík can-utils, který obsahuje uživatelské nástroje na zobrazování a vytváření provozu na sběrnici. [16]

Program pro Raspberry Pi je napsaný v jazyku python s použitím knihoven tkinter, os, math, can a threading. Program pro mikrokontroler dsPIC je napsaný v jazyku C.

V programu pro mikrokontroler se používají knihovny stdint, stdio a knihovna xc. Na psaní se použilo vývojové prostředí MPLAB X IDE. Použitý kompilátor je MPLAB XC16.

4 Zprovoznění CAN komunikace

4.1 Konfigurace CAN modulu na Raspberry Pi

Rozšiřující deska byla připojena k Raspberry Pi. Po nastartování počítače byl otevřen soubor „/boot/config.txt“. Na jeho konec se přidaly příkazy [17]:

- dtparam=spi=on - zapni SPI řadič
- dtoverlay=mcp2515-can0 - vyber zařízení mcp2515-can0
- dtparam=oscillator=16000000 - frekvence oscilátoru daného zařízení je 16 MHz
- dtparam=interrupt=23 - vyber GPIO pin 23 pro signál SPI přerušení
- dtoverlay=mcp2515-can1 - vyber zařízení mcp2515-can1
- dtparam=oscillator=16000000 - frekvence oscilátoru daného zařízení je 16 MHz
- dtparam=interrupt=25 - vyber GPIO pin 25 pro signál SPI přerušení

Počítač byl následně restartován. Dále se nastavila síťová zařízení can0 a can1 pomocí příkazů [15]:

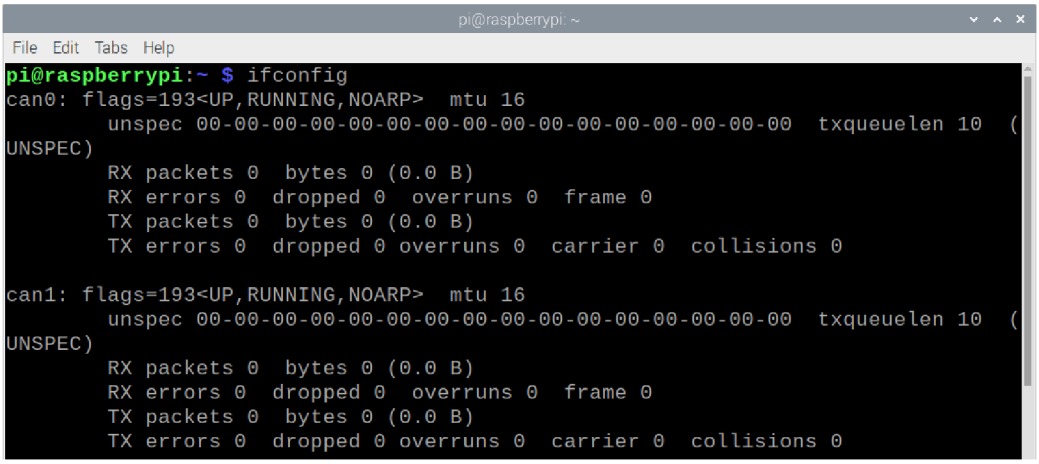
- sudo ip link set can0 type can bitrate 1000000 - nastav can0 jako CAN zařízení a nastav bitovou rychlost 1 Mbit/s
- sudo ip link set can0 up - zapni can0
- sudo ip link set can1 type can bitrate 1000000 - nastav can1 jako CAN zařízení a nastav bitovou rychlost 1 Mbit/s
- sudo ip link set can1 up - zapni can1

Pro kontrolu, jestli bylo nastavení úspěšné, byl zadán příkaz „ifconfig“, který zobrazí informace o síťových zařízeních. Na obr. 4.1 je vidno, že inicializace obou zařízení proběhla úspěšně.

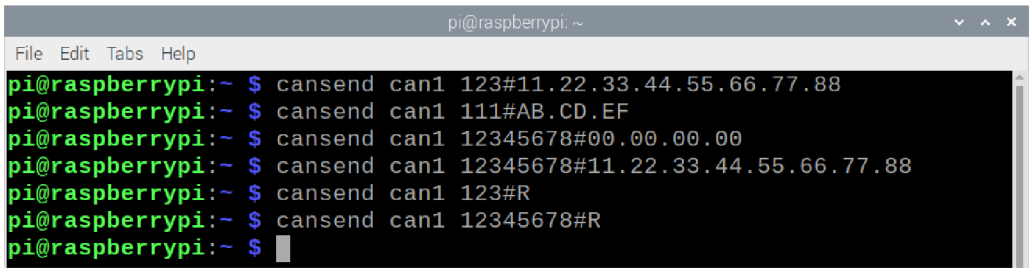
Na otestování rozšiřující CAN desky se vyzkoušela komunikace mezi jednotkami can0 a can1. Na Raspberry Pi byl nainstalován balík „can-utils“, který obsahuje nástroje na testování CAN sběrnice z příkazového řádku. V této práci se budou používat příkazy „cansend“, který se používá na vyslání zprávy, a „candump“, který se používá na monitorování sběrnice. [16]

Na rozšiřující desce (obr. 3.3) byl vodičem propojen výstup „H“ na terminálu can0 s výstupem „H“ na terminálu can1. Rovněž byl propojen výstup „L“ na terminálu can0 s výstupem „L“ na terminálu can1. Pomocí propojky se připojily zakončovací rezistory. Tím se vytvořila CAN síť.

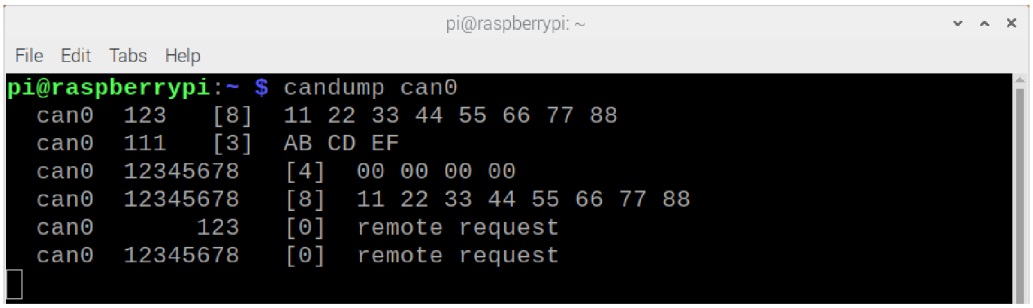
Na jednom příkazovém řádku byl zadán příkaz „candump can0“. Na tomto terminálu se zobrazuje provoz na sběrnici, který zaznamená zařízení can0. Na druhém příkazovém řádku bylo pomocí příkazu „cansend“ vysláno několik testovacích zpráv ze zařízení can1 (obr. 4.2). Tyto zprávy byly úspěšně zaznamenané zařízením can0 (obr. 4.3).



Obrázek 4.1: Výpis příkazu „ifconfig“



Obrázek 4.2: Vysílání zpráv



Obrázek 4.3: Monitorování sběrnice

4.2 Konfigurace CAN periferie mikrokontroleru

Konfigurace časování bitu

Časování bitu se nastavuje v závislosti na použitém hardwaru. V našem případě se na zpoždění signálu podílí CAN budič na rozšiřující desce Raspberry Pi, oddělovač signálu, vodiče a CAN budič připojený k mikrokontroleru. Délka vodičů je kolem 20 cm. Při uvážení typické hodnoty zpoždění 5ns/m vychází zpoždění na vodičích 1 ns. [6]

Oba budiče jsou SN65HVD230, které mají maximální zpoždění z datového vstupu na datový výstup 135 ns. [18]
Zpoždění digitálního oddělovače Si8642ED-IS je 13 ns. [19]

Časování bitu je potřeba nastavit s ohledem k největšímu možnému zpoždění. To je dvojnásobek doby cesty signálu mezi dvěma nejvzdálenějšími uzly. V našem případě vychází maximální zpoždění 298 ns. [6]

Frekvence mikrokontroleru je 40 MHz. Rychlost přenosu byla zvolena 1 Mbit/s. Počet časových kvant v jednom jmenovitém bitovém čase byl zvolen 20. Z toho vychází z rovnice 6-2 uvedené v [12] celočíselná hodnota děličky v CAN periferii, takže daná konfigurace je možná. Pak podle rovnice 6-1 uvedené v [12] je doba jednoho časového kvanta TQ = 50 ns. Propagační segment musí mít podle vztahu (9), který je uvedený v [7] minimální délku 6 TQ. Dále platí, že čím je větší šířka resynchronizačního skoku a čím je větší minimum z fázového segmentu 1 a fázového segment 2, tím větší je tolerance frekvence oscilátoru. [7]

Na základě zmíněných úvah byly zvoleny hodnoty:

- Doba propagačního segmentu = 7 TQ
- Doba fázového segmentu 1 = 6 TQ
- Doba fázového segmentu 2 = 6 TQ
- Šířka resynchronizačního skoku = 4 TQ

Konfigurace bufferu pro CAN zprávy

CAN periferie mikrokontroleru nemá vyhrazenou vlastní paměť, ale využívá přímý přístup do RAM paměti (Direct Memory Access). Tato funkce umožňuje přenos dat mezi periferií a RAM pamětí bez zásahu procesoru. Přenos je zprostředkovaný DMA řadičem. Samotná periferie má jen jeden registr (o velikosti 1 slovo) pro vysílaná data a jeden pro přijatá data. [11]

Velikost bufferu, s kterým periferie pracuje je možné nastavit od 4 do 32 zpráv. Buffery 0 až 7 je možné nastavit pro vysílání nebo příjem. Ostatní buffery umožňují jen příjem. V rámci bufferů je možné pro příjem zpráv vyhradit FIFO oblast. [12]

Byla zvolena následovná nastavení:

- Velikost bufferu byla nastavena na 8 zpráv
- Buffery 0 až 3 byly nastaveny pro vysílání, buffer 4 pro příjem zpráv
- Začátek FIFO bufferu byl nastaven na buffer 5

V této práci byly potřeba dva kanály DMA řadiče. Jeden se využil pro přenos dat z RAM paměti do vysílacího registru periferie, druhý pro přenos z přijímacího registru periferie do RAM paměti. Na určení toho, kde v paměti se mají data zapsat, nebo odkud se mají data vyčíst je potřeba znát adresu. DMA řadič v použitém mikrokontroleru umožňuje tři způsoby adresování [20]:

- Nepřímé adresování registrem, s post-inkrementem – adresa je inkrementována po každém přenosu
- Nepřímé adresování registrem, bez post-inkrementu – adresa není inkrementována
- Nepřímé adresování periferií – periferie poskytuje adresu

Při nepřímém adresování periferií je pro kanál určena základní adresa, která musí mít určitý počet nejméně důležitých bitů nulových. Když periferie požádá o přenos, poskytne zároveň DMA kanálu tyto poslední bity. Operací OR se pak vytvoří skutečná adresa. Periferie tedy poskytuje posunutí od počáteční adresy. Kolik nulových koncových bitů je potřeba zajistit v základní adrese závisí od velikosti bufferu. [20]

Velikost bufferu je 8 CAN zpráv. Každá CAN zpráva má 8 slov. To je dohromady 128 bytů. Periferie bude generovat posunutí od 0 do 127, takže posledních 7 bitů základní adresy musí být 0. Toho se docílí tak, že základní adresa bude násobkem 128. [20]

DMA kanál může být nastaven do jednorázového nebo kontinuálního režimu. Při jednorázovém režimu se kanál po přenosu nadefinovaného bloku vypne. Při kontinuálním režimu se adresa po každém přenosu inkrementuje a po posledním přenosu bloku se resetuje zpět na počáteční adresu. Velikost tohoto bloku možné nastavit. [20]

Je možné nastavit i Ping-Pong mód, při kterém se střídavě využívají dva buffery. S jedním může pracovat DMA kanál, zatím co druhý může zpracovávat procesor. Tento mód zde ale nebude potřebný. [20]

Při konfiguraci bylo potřeba nastavit tyto parametry:

- Velikost jednoho přenosu byla nastavena na 1 slovo (2 byty)
- Směr přenosu:
  - Kanál 0: z RAM do periferie
  - Kanál 1: z periferie do RAM
- Způsob adresování byl nastaven na nepřímé adresování periferií
- Operační režim kanálu byl nastaven na kontinuální, bez Ping-Pong módu
- Spouštěč DMA přenosu:
  - Kanál 0: přerušení vyvolané žádostí o vyslání zprávy
  - Kanál 1: přerušení způsobené tím, že je k dispozici přijatá zpráva
- Počet DMA přenosů v jednom bloku byl nastaven na 8

- Registr periferie, s kterým kanál pracuje:
  - Kanál 0: registr pro vysílaná data (C1TXD)
  - Kanál 1: registr pro přijatá data (C1RXD)
- Počáteční adresa byla nastavena na adresu předem vytvořené proměnné

Konfigurace příjmového filtru

Zpráva je přijata tehdy, jestli se určité bity identifikátoru zprávy shodují s bity filtru. O tom, které bity se budou porovnávat rozhoduje maska filtru. Byla provedena následovná nastavení:

- Bity masky byly nastaveny na hodnotu 0, budou se tedy přijímat všechny zprávy
- Bity filtru byly nastaveny na hodnotu 0
- Pro filtr byla nastavena daná maska
- Byl zvolen buffer 4 jako úložiště pro zprávy přijaté filtrem

Obsluhy přerušení

V souvislosti s činností CAN periferie se využívají přerušení od následovných zdrojů:

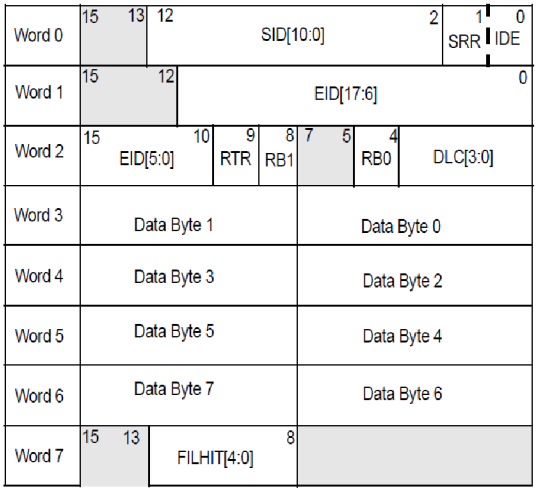
- CAN událost
- DMA kanál 0
- DMA kanál 1

Přerušení od CAN události je způsobeno dokončením vysílání, zapsáním přijaté zprávy do bufferu a dalšími událostmi. V obsluze přerušení od CAN události je potřeba podívat se, který příznak je nastaven na 1, tedy co způsobilo dané přerušení. V této práci se vyhodnocuje TBIF – příznak, který značí, že zpráva byla vyslána a RBIF – příznak, který značí, že zpráva byla zapsána do bufferu. Tyto příznaky je potřeba v obsluze vynulovat. Když je příznak RBIF vyhodnocen jako 1, je potřeba zkontrolovat jednotlivé příznaky RXFULx, aby se zjistilo, který buffer je plný. Pak se daný buffer zpracuje a jeho příznak se vynuluje. [12]

Přerušení od DMA kanálů 0 a 1 jsou generovány tehdy, když byl dokončen přenos jednoho bloku (jedné zprávy) z/do periferie. V obsluze přerušení je potřeba příslušné příznaky vynulovat. [20]

Vysílání a příjem zpráv

Buffer jedné zprávy má velikost 8 slov. Obsah těchto slov je znázorněn na obr. 4.4. Toto rozložení platí jak pro buffer, z kterého se bude zpráva vysílat tak i pro buffer do kterého byla zapsána přijatá zpráva.



Obrázek 4.4: Obsah jednoho bufferu [12]

Význam jednotlivých částí je následovný [12]:

- Slovo 0:
  - Bity standardního identifikátoru nebo bity 28-18 rozšířeného identifikátoru
  - SRR bit - V standardním formátu má bit označený jako SRR funkci RTR bitu. Tedy SRR bude 0 pro datový rámec, 1 pro žádost o rámec. V případě rozšířeného formátu musí být SRR = 1.
  - IDE bit - je 0 pro standardní, 1 pro rozšířený formát zprávy.
- Slovo 1:
  - Bity 17 až 6 rozšířeného identifikátoru. Při standardním formátu se tyto bity nevyužívají.
- Slovo 2:
  - Bity 5 až 0 rozšířeného identifikátoru - při standardním formátu se nevyužívají
  - RTR bit - Při standardním formátu se RTR bit nevyužívá. Když se jedná o rozšířený formát, RTR je 0 pro datový rámec, 1 pro žádost o rámec.
  - RB1 bit - musí být vždy 0
  - RB0 bit - musí být vždy 0
  - DLC bity - určují počet datových bytů, které zpráva obsahuje. Maximální počet datových bytů v jedné zprávě je 8.
- Slova 3 až 6:
  - Datové byty
- Slovo 7:
  - FILHIT bity - označují číslo filtru, který zprávu přijal. V případě vysílacího bufferu se tyto bity nevyužívají.

Připravená zpráva se vyšle nastavením TXREQx bitu příslušného bufferu na 1. Po vyslání se tento bit automaticky vynuluje. [12]

Přijatá zpráva se vyčte v obsluze přerušení. Následně je potřeba vynulovat bit RXFULx příslušného bufferu, což označí daný buffer jako zpracovaný a volný pro další zápis. [12]

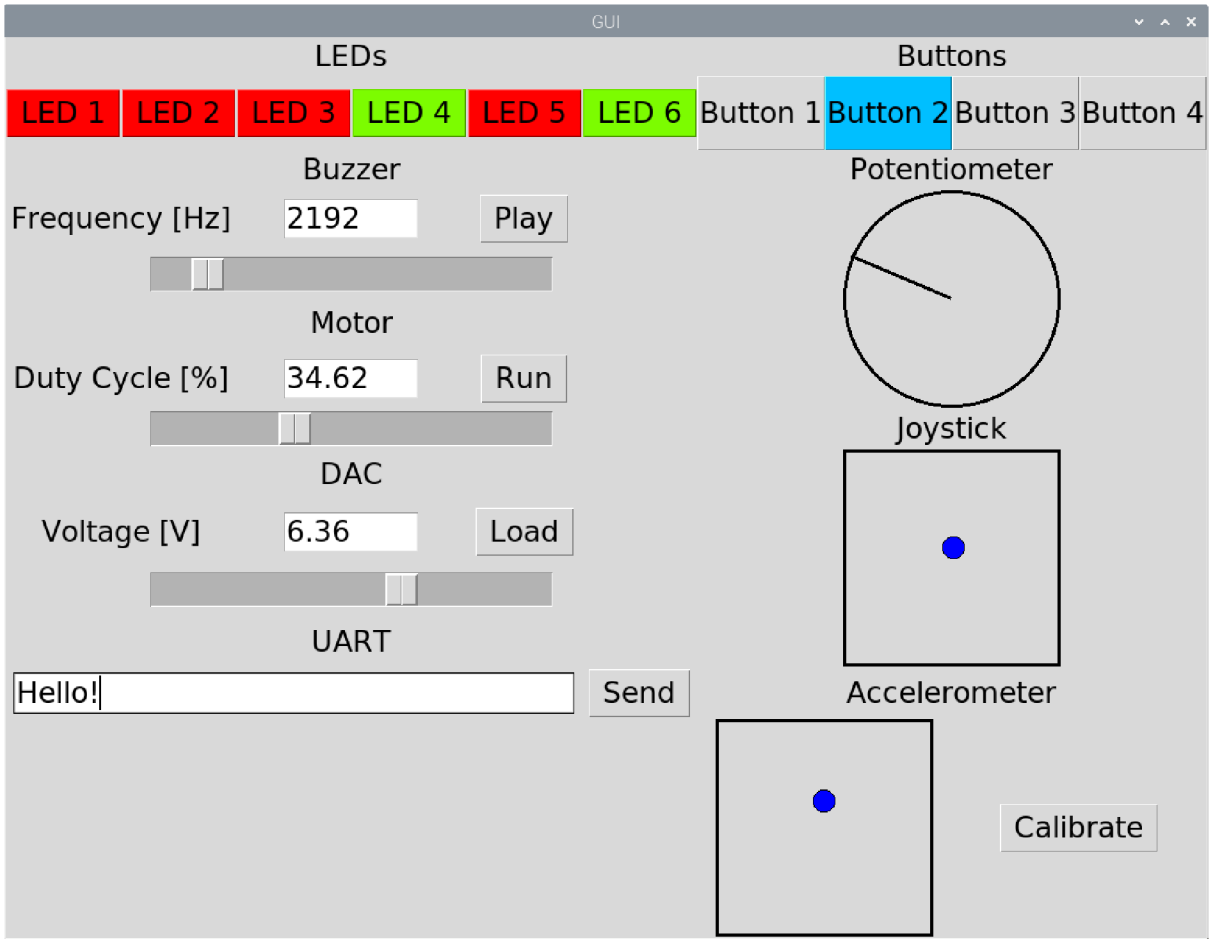
5 Aplikace

Program aplikace pro Raspberry Pi je napsaný v jazyku python. Na vytvoření grafického rozhraní byl použitý modul Tkinter. Grafické uživatelské rozhraní aplikace (obr. 5.1) obsahuje:

- Tlačítka pro LED diody 1 až 6
- Vstupní pole pro zadávaní frekvence bzučáku
- Tlačítko pro zapnutí/vypnutí bzučáku
- Vstupní pole pro zadávaní střídy pro DC motor
- Tlačítko pro spuštění/vypnutí motoru
- Vstupní pole pro zadávaní napětí na výstupu DA převodníku
- Tlačítko pro zapnutí/vypnutí DA převodníku
- Vstupní pole pro zadávaní UART zprávy
- Tlačítko pro vyslání UART zprávy
- Štítky pro tlačítka 1 až 4
- Obrázek pro potenciometr
- Obrázek pro joystick
- Obrázek pro akcelerometr
- Tlačítko kalibrace

Na levé straně GUI je možno z Raspberry Pi ovládat prvky na vývojové desce, tedy zprávy se vysílají z Raspberry Pi. Na pravé straně se zobrazují stavy prvků na vývojové desce, tedy zprávy se vysílají z mikrokontroleru. Všechna komunikace probíhá prostřednictvím CAN sítě. Program vytvořené aplikace je dodaný k práci jako elektronická příloha - viz soubory „app_RPi.py“ a „app_uC.c“. Aplikace umožňuje:

- Ovládat LED diody
- Ovládat bzučák
- Ovládat DC motor
- Vytvořit žádané napětí pomocí D/A převodníku



Obrázek 5.1: Aplikace

- Vyslat UART zprávu

V programu mikrokontroleru jsou pro tyto činnosti napsané funkce. Je tam tedy zvlášť funkce pro LED diody, zvlášť pro bzučák atd. Vstupy těchto funkcí jsou data přijaté CAN zprávy. Když mikrokontroler přijme zprávu, vyvolá to přerušení. V obsluze přerušení se přijatá zpráva vyčte a na základě identifikátoru se zavolá příslušná funkce.

Aplikace dále zobrazuje:

- Stav tlačítek – pomocí barvy štítků
- Natočení potenciometru – pomocí ručky v obrázku
- Polohu joysticku – pomocí kuličky v obrázku
- Natočení akcelerometru – pomocí kuličky v obrázku

V programu aplikace na Raspberry Pi běží v samostatném vlákně nekonečný cyklus, který čeká na přijatí zprávy. Když Raspberry Pi přijme zprávu, podívá se na identifikátor a zavolá funkci týkající se příslušného prvku. Je tedy napsaná zvlášť funkce pro tlačítka, zvlášť pro potenciometr atd.

5.1 Prvky

LED diody

Po kliknutí na jedno z LED tlačítek se v programu Raspberry Pi vykoná následovné:

- Na základě proměnné, která obsahuje informaci o stavu LED diody (zapnutá nebo vypnutá), se zjistí aktuální stav dané LED diody
- Do proměnné se zapíše opačná hodnota
- Změní se barva tlačítka příslušné LED diody buď ze zelené na červenou, nebo z červené na zelenou
- Vyšle se CAN zpráva obsahující informaci o tom, která LED dioda se má změnit a jak se má změnit

Program mikrokontroleru:

- Z přijaté zprávy se vyčte, která LED dioda má změnit svůj stav a jak ho má změnit
- Zavolá se funkce, která buď zapne nebo vypne příslušnou LED diodu

Bzučák, DC motor a D/A převodník

Když se klikne na tlačítko bzučáku, tak v programu Raspberry Pi se vykoná následovné:

- Program se podívá na proměnnou obsahující informaci o stavu bzučáku (zapnutý nebo vypnutý)
- Hodnota dané proměnné se změní na opačnou
- Změní se text na tlačítku z Play na Stop nebo ze Stop na Play
- Frekvence, kterou bude mít bzučák se určí vyčtením hodnoty ze vstupního pole
- Vyšle se CAN zpráva obsahující informaci o žádaném stavu a frekvenci bzučáku

Pro aktualizaci frekvence je možné po napsaní frekvence do vstupního pole stisknout enter, nebo potáhnout sliderem. Číslo ve vstupním poli a poloha slideru se synchronizují. Když se do vstupního pole zadá neplatná hodnota, objeví se messagebox, který oznámí chybu.

Program mikrokontroleru:

- Z přijaté zprávy se vyčte, jaký má být stav bzučáku a jakou má mít bzučák frekvenci
- Buď se zapne nebo vypne časovač, který ovládá bzučák
- Registr periody časovače se nastaví na hodnotu odvozenou z frekvence

Pro DC motor je program Raspberry Pi i program mikrokontroleru podobný jako při bzučáku, až na tyto výjimky:

- Text na tlačítku je buď Run nebo Stop
- Do vstupního pole se zadává střída PWM

Pro D/A převodník je program Raspberry Pi i program mikrokontroleru podobný jako při bzučáku a motoru, s těmito výjimkami:

- Text na tlačítku je buď Load nebo Unload
- Do vstupního pole se zadává požadované napětí na výstupu DA převodníku
- V programu mikrokontroleru se místo nastavování časovače nebo PWM zde vyšle zpráva do DA převodníku prostřednictvím rozhraní SPI

UART

Když se klikne na tlačítko „Send“, tak se v programu Raspberry Pi vykoná následovné:

- Vyčte se text ze vstupního pole
- Vyšle se CAN zpráva, přičemž jednotlivé datové byty zprávy jsou znaky textu (hodnoty znaků z ASCII tabulky)

V případě, že je do vstupního pole zadaný text delší než 8 znaků, se vyšle víc CAN zpráv těsně za sebou.

Program mikrokontroleru:

- Z přijaté CAN zprávy se vyčtou jednotlivé znaky
- Znaky se přepošlou na UART

Tlačítka

Program mikrokontroleru:

- V programu mikrokontroleru je puštěný časovač, který každých 5 milisekund vygeneruje přerušení
- V obsluze přerušení se volá funkce, která zjišťuje stav tlačítek. V této funkci jsou definované statické proměnné, které obsahující aktuální stavy tlačítek, a taky proměnné, které obsahují stavy tlačítek v čase předešlého volání funkce. Tyto proměnné slouží na ošetření zákmitů při stlačení nebo odtažení tlačítka (debouncing)
- Když se zjistí změna stavu tlačítka, aktualizuje se příslušná proměnná obsahující stav daného tlačítka
- Následně se vyšle CAN zpráva, která obsahuje číslo tlačítka, které změnilo svůj stav, a nový stav tlačítka

Program Raspberry Pi:

- Z přijaté zprávy se vyčte, které tlačítko změnilo stav a jaký je jeho nový stav
- Změní se barva štítku příslušného tlačítka z šedé na modrou nebo z modré na šedou

Potenciometr, joystick a akcelerometr

Program mikrokontroleru:

- Stejný časovač, který se používá pro tlačítka, se používá taky pro potenciometr, joystick a akcelerometr
- V obsluze přerušení od časovače se spustí série pěti A/D konverzí: jedna pro potenciometr, dvě pro joystick a dvě pro akcelerometr
- Výsledky konverzí se zapíšou do DMA bufferu
- Když je přenos všech výsledků do DMA bufferu dokončen, vygeneruje se přerušení od DMA kanálu
- V obsluze přerušení se zavolají tři funkce, jedna pro potenciometr, jedna pro joystick a jedna pro akcelerometr
- Ve funkcích se výsledky konverzí zpracují
- Vyšlou se CAN zprávy obsahující informace o natočení potenciometru, poloze joysticku a natočení akcelerometru

Program Raspberry Pi:

- Vyčítá se obsah z přijatých CAN zpráv
- Na základě obsahu se upraví obrázky, které znázorňují natočení potenciometru, polohu joysticku a natočení akcelerometru

Vedle obrázku akcelerometru je umístěno tlačítko kalibrace, které slouží na umístění tečky do středu obrázku při aktuálním natočení akcelerometru.

6 Knihovna

Jedním z cílů této práce je vytvoření knihovny v jazyku C pro mikrokontroler dsPIC33FJ128MC804, která by obsahovala základní funkce potřebné pro CAN komunikaci. Knihovna i hlavičkový soubor knihovny jsou k práci dodány jako elektronické přílohy - viz „canlib.h“ a „canlib.c“.

V hlavičkovém souboru knihovny jsou pomocí maker definovány parametry, za základě kterých je možno provést základní nastavení časování bitu a nastavení bufferu pro CAN periferii. Je to pro případ, kdy by uživatel nepotřeboval přesné vlastní nastavení. Hlavičkový soubor dále obsahuje deklaraci bufferu a prototypy funkcí knihovny. Samotný C soubor knihovny obsahuje definici bufferu a definice funkcí. Při definici bufferu jsou specifikovány vlasntnosti [21]:

- space(dma) - tato vlastnost umístní proměnnou do DMA paměti
- aligned() - tato vlastnost zaručí potřebný počet nulových nejméně důležitých bitů

Pro praktickou manipulaci s CAN zprávami je v hlavičkovém souboru definována vlastní struktura pro CAN zprávu. Členy této struktury obsahují informace o identifikátoru zprávy, o typu zprávy a o datech zprávy.

6.1 Funkce

canTimingConf

Funkce canTimingConf nakonfiguruje časování bitu. Při volání této funkce se do vstupu zadají parametry odpovídající požadované konfiguraci. Těmito parametry jsou délky jednotlivých segmentů bitového času, počet časových kvant, na které je rozdělený bitový čas a požadovaná bitová rychlost sběrnice. Dále se do vstupu ještě zadává, jestli se má sběrnice vzorkovat jednou nebo třikrát.

canBufConf

Funkce canBufConf nakonfiguruje DMA buffer, s kterým bude periferie pracovat. Při volání této funkce se do vstupu zadají parametry odpovídající požadované konfiguraci. Těmito parametry jsou velikost bufferu, označení začátku fifo bufferů, a kód pro rozdělení bufferů 0-7 na vysílací a přijímací.

dmaCanTxInit

Funkce dmaCanTxInit inicializuje DMA kanál pro přenos dat z bufferu do vysílacího registru CAN periferie. Do vstupu se zadá číslo kanálu, který se má použít.

dmaCanRxInit

Funkce dmaCanRxInit inicializuje DMA kanál pro přenos dat z přijímacího registru CAN periferie do bufferu. Do vstupu se zadá číslo kanálu, který se má použít.

canDefaultInit

Funkce canDefaultInit slouží na inicializaci periferie s použitím základních parametrů definovaných v hlavičkovém souboru. Tato funkce inicializuje časování bitu, buffer i oba DMA kanály.

maskConfig

Funkce maskConfig nakonfiguruje masku. Uživatel do vstupu zadá číslo masky, která se má nakonfigurovat, typ masky, a samotnou masku. Typ masky může byt standardní nabo rozšířený.

filterConfig

Funkce filterConfig nakonfiguruje filtr. Uživatel do vstupu zadá číslo filtru, který se má nakonfigurovat, masku, kterou bude filtr používat, typ filtru (standardní nebo rozšířený), samotný filtr a číslo bufferu, do kterého filtr zapíše přijaté zprávy.

enableFilter

Funkce enableFilter zapne filtr. Do vstupu se zadá číslo filtru, který se má zapnout.

disableFilter

Funkce disableFilter vypne filtr. Do vstupu se zadá číslo filtru, který se má vypnout.

sendCanMsg

Funkce sendCanMsg zapíše zprávu do vysílacího bufferu a následne ji vyšle. Do vstupu se zadá zpráva, která se má vyslat a buffer, přes který se má zpráva vyslat.

readCanMsg

Funkce readCanMsg vyčte CAN zprávu z bufferu a uloží ji do uživatelem zadané proměnné. Uživatel zadá do vstupu číslo bufferu, z kterého se má zpráva vyčíst a adresu proměnné, do které se má vyčtená zpráva zapsat.

7 Závěr

První část práce byla seznámení se s CAN protokolem. Na základě nadobudnutých znalostí se pak nastavil CAN modul připojený k Raspberry Pi a CAN periferie mikrokontroleru. Po úpsěšném zprovoznění CAN komunikace se vytvořila vzorová aplikace, při které si zmíněné zažízení mezi sebou vyměňují informace. Aplikace byla pro ukázku provozu na CAN sbernici rozšírena o další prvky, které se nacházely na vývojové desce, nebo které měl autor po ruce. Součástí výsledné aplikace je grafické uživatelské rozhraní na Raspberry Pi, z kterého je možno ovládat LED diody, bzučák, motor, vytvořit napětí pomocí D/A převodníku a poslat text na UART. Dále aplikace zobrazuje stav tlačítek na vývojové desce, polohu potenciometru, joysticku a akcelerometru.

Přínosem práce je vytvořená knihovna, kterou je možné do programu naimportovat. Funkce, které knihovna obsahuje umožňují nastavení CAN periferie podle požadavek a jednoduchou manipulaci s vysílanými i přijímanými zprávami.

Seznam použitých zdrojů

[1] *CAN Specification* .Ver. 2.0. Robert Bosch, 1991.

[2] History of CAN technology. *can-cia.org* [online]. CAN in Automation, ©1992-2021 [cit. 2021-05-10]. Dostupné z: www.can-cia.org/can-knowledge/can/can-history

[3] The CAN Protocol Tutorial. *kvaser.com* [online]. Kvaser Europe, ©2021 [cit. 2021-05-10]. Dostupné z: www.kvaser.com/can-protocol-tutorial

[4] *ISO11898-2.svg* [online]. Wikimedia Foundation, 2017 [cit. 2021-05-10]. Dostupné z: commons.wikimedia.org/wiki/File:ISO11898-2.svg

[5] *CAN with Flexible Data-Rate Specification*. Ver. 1.0. Robert Bosch, ©2011.

[6] WATTERSON, Conal. Configure Controller Area Network (CAN) Bit Timing to Optimize Performance. *Analog Dialogue* [online]. 2014, vol. 48, no. 3, pp. 29-31 [cit. 2021-05-10] Dostupné z: www.analog.com/media/en/analog-dialogue/volume-48/number-3/articles/volume48-number3.pdf

[7] ROBB, Stuart. *AN1798: CAN Bit Timing Requirements* [online]. Rev. 4. Freescale Semiconductor, ©2004 [cit. 2021-05-10]. Dostupné z: www.nxp.com/docs/en/application-note/AN1798.pdf

[8] CORRIGAN, Steve. *Introduction to the Controller Area Network (CAN)* [online]. Rev. B. Texas Instruments, ©2018 [cit. 2021-05-10]. Dostupné z: www.ti.com/lit/an/sloa101b/sloa101b.pdf

[9] *Raspberry Pi 3 Model B+ product brief* [online]. Raspberry Pi Foundation, [cit. 2021-05-10]. Dostupné z: static.raspberrypi.org/files/product-briefs/200206+Raspberry+Pi+3+Model+B+plus+Product+Brief+PRINT&DIGITAL.pdf

[10] 2-Channel Isolated CAN Expansion HAT for Raspberry Pi. *waveshare.com* [online]. Waveshare Electronics, [cit. 2021-05-10]. Dostupné z: www.waveshare.com/2-CH-CAN-HAT.htm

[11] *dsPIC33FJ32MC302/304, dsPIC33FJ64MCX02/X04, and dsPIC33FJ128MCX02/X04 Data Sheet* [online]. Rev. G. Microchip Technology, ©2007-2012 [cit. 2021-05-10]. Dostupné z: ww1.microchip.com/downloads/en/DeviceDoc/70291G.pdf

[12] *dsPIC33F/PIC24H FRM, Enhanced Controller Area Network (CAN)* [online]. Rev. D. Microchip Technology, ©2020 [cit. 2021-05-10]. Dostupné z: ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33F-PIC24H-FRM-Enhanced-Controller-Area-Network-CAN-DS70000185D.pdf

[13] *SN65HVD230_0-550x550h.JPG* [online]. Pájeníčko, ©2020 [cit. 2021-05-10]. Dostupné z: pajenicko.cz/image/cache/catalog/komponenty/prevodniky/SN65HVD230/SN65HVD230_0-550x550h.JPG

[14] Raspberry Pi OS - Raspberry Pi Documentation. *raspberrypi.org* [online]. Raspberry Pi Foundation, [cit. 2021-05-10]. Dostupné z: www.raspberrypi.org/documentation/raspbian

[15] HARTKOPP, Oliver, et al. *Readme file for the Controller Area Network Protocol Family (aka SocketCAN)* [online]. Linux Foundation, ©1997-2014 [cit. 2021-05-10]. Dostupné z: www.kernel.org/doc/Documentation/networking/can.txt

[16] *can-utils README* [online]. GitHub, ©2021 [cit. 2021-05-10]. Dostupné z: github.com/linux-can/can-utils/blob/master/README.md

[17] *Device Tree overlays README* [online]. GitHub, ©2021 [cit. 2021-05-10]. Dostupné z: github.com/raspberrypi/firmware/blob/master/boot/overlays/README

[18] *SN65HVD23x 3.3-V CAN Bus Transceivers datasheet* [online]. Rev. O. Texas Instruments, ©2020 [cit. 2021-05-10] Dostupné z: www.ti.com/lit/ds/symlink/sn65hvd230.pdf

[19] *Si864x Data Sheet Low-Power Quad-Channel Digital Isolators* [online]. Rev. 2.15. Silicon Laboratories, 2019 [2021-05-10] Dostupné z: www.silabs.com/documents/public/data-sheets/si864x-datasheet.pdf

[20] *dsPIC33F/PIC24H Family Reference Manual - Section 38. Direct Memory Access (DMA) (Part III)* [online]. Rev. C. Microchip Technology, ©2007-2012 [cit. 2021-05-10]. Dostupné z: ww1.microchip.com/downloads/en/DeviceDoc/70215C.pdf

[21] *MPLAB XC16 C Compiler User's Guide* [online]. Rev. K. Microchip Technology, ©2012-2020 [cit. 2021-05-10]. Dostupné z: ww1.microchip.com/downloads/en/DeviceDoc/50002071K.pdf

Seznam příloh

app_RPi.py

app_uC.c

canlib.h

canlib.c