ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická
Katedra elektroniky a informačních technologií

DIPLOMOVÁ PRÁCE

Formula Student – jednotka snímání pozice pedálů

Autor práce: **Adam Trojan**
Vedoucí práce: **Ing. Martin Zavřel**

2022

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Adam TROJAN**
Osobní číslo:	**E20N0055P**
Studijní program:	**N0714A060013 Elektronika a informační technologie**
Specializace:	**Elektronika**
Téma práce:	**Formula Student – jednotka snímání pozice pedálů**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

1. Proveďte rešerši platných pravidel a doporučení pro snímání pozice pedálů v soutěžích Formula Student.
2. Na základě provedené rešerše zvolte koncepci snímání pedálů pro Formula Student ZČU a detailně ji popište.
3. Proveďte návrh obvodového schéma jednotky snímání polohy pedálů
4. Proveďte návrh DPS jednotky snímání polohy pedálů s ohledem na geometrii rámu a pedálů Formula Student ZČU. Zohledněte také potřebné pouzdření a krytí jednotky.
5. Proveďte oživení jednotky snímání polohy pedálů.

Rozsah diplomové práce: **40 – 60**
Rozsah grafických prací: **dle doporučení vedoucího**
Forma zpracování diplomové práce: **elektronická**

Seznam doporučené literatury:

1. Formula Student Germany: International Design Competition [online]. 2019 [cit. 2019-04-15]. Dostupné z: fsg.one/rules
2. Senzory a měřící obvody / Stanislav Ďaďo, Marcel Kreidl. – 1. vyd. – Praha : ČVUT, 1996. – 315 s. : 351 obr., lit.. – ISBN 80-01-01500-9
3. Analogové elektronické systémy. 1. část / Jiří Pinker, Václav Koucký. – 3. vyd. – Plzeň : Západočeská univerzita, 2004. – 142 s. : il.. – ISBN 80-7043-284-5 (1.část)
4. Analýza signálů a soustav / Pavel Nevřiva. – 1. vyd.. – Praha : BEN, 2000. – 671 s. : il., grafy ; 21 cm. – Přehled základních použitých matematických vzorců. – ISBN 80-7300-004-0
5. Literatura a podklady týmu Formula Student ZČU

Vedoucí diplomové práce: **Ing. Martin Zavřel**
Research and Innovation Centre for Electrical Engineering

Datum zadání diplomové práce: **8. října 2021**
Termín odevzdání diplomové práce: **26. května 2022**

Západočeská univerzita v Plzni * Fakulta elektrotechnická *

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 8. října 2021

Abstrakt

Tato diplomová práce se zaměřuje na vývoj elektronického snímače polohy akceleračního pedálu budoucí studentské elektrické formule s důrazem na pravidla FSAE a FSG. Snímač je řešený pomocí magnetu a dvojice Hallovo sond. Metoda bipolárního buzení sond zajišťuje snímači linearitu. Navržená deska plošných spojů je uložena v navrženém pouzdru vyrobeném pomocí 3D tisku. Snímač jako celek je napájen z napětí 12 V DC a komunikuje po sběrnici CAN.

Klíčová slova

snímač, senzor, pedál, studentská formule, bezkontaktní snímaní, Hallovo sonda, permanentní magnet, bipolární buzení

Abstract

This diploma thesis focuses on the development of an electronic accelerator pedal position sensor of the future student electric formula with emphasis on the rules of the FSAE and FSG. The sensor is solved using a magnet and a pair of Hall probes. The bipolar excitation method of the probes ensures linearity of the sensor. The designed printed circuit board is situated in a designed case made by 3D printing. The sensor is powered by 12 V DC and communicates via the CAN bus.

Key Words

sensor, pedal, student formula, contactless sensing, hall probe, permanent magnet, bipolar excitation

Poděkování

Tímto bych rád poděkoval vedoucímu této diplomové práce Ing. Martinovi Zavřelovi a konzultantovi Ing. Luboši Streitovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce. Dále děkuji Ing. Zdeňkovi Kubíkovi, Ph.D. za věcné připomínky při návrhu vstupních filtrů.

Obsah

Seznam symbolů a zkratek

Značka	Popisek	Jednotka
B	Magnetická indukce	(T), (G)
C	Elektrická kapacita	(F)
f	Frekvence	(Hz)
I	Elektrický proud	(A)
L	Elektrická indukčnost	(H)
R	Elektrický odpor	(Ω)
U	Elektrické napětí	(V)

Zkratka	Popisek
ADC	Analog to digital converter
CAN	Controller Area Network
DC	Direct current
DLC	Data Length Code
DPS	Deska plošných spojů
EEPROM	Electrically Erasable Programmable Read-Only Memory
ESD	Electrostatic discharge
FIFO	First In, First Out
FMEA	Failure Modes and Effects Analysis
FSAE	Formula SAE
FSG	Formula Student Germany
GPIO	General-purpose input/output
IIC	Inter-Integrated Circuit (datová sběrnice)
JTAG	Joint Test Action Group
LDO	Low-Dropout
LED	Light-Emitting Diode
MCU	Microcontroller unit
PWM	Pulse Width Modulation
SAE	Society of Automobile Engineers
SPI	Serial Peripheral Interface
THT	Through-hole technology
TVS	Transient-voltage-suppression

Úvod

V rámci spolupráce Fakulty strojní a Fakulty elektrotechnické Západočeské univerzity v Plzni je vyvíjena plně elektrická studentská formule. Soutěže studentských formulí jsou již tradičními, a prestižními událostmi, kterých se účastní řada univerzit z celého světa.

Snímač pozice pedálů je nutnou součástí každého elektrovozidla. Má za úkol snímat míru stlačení pedálu a předávat informace do nadřazené jednotky. Celé zařízení je napájeno z nízkonapěťové části vozidla o napětí 12 V. Komunikovat s nadřazenou jednotkou má pomocí sběrnice CAN. Zařízení musí splňovat určitá pravidla (normy), podobná těm automobilovým, konkrétně FSAE [1] a FSG [2]. Pravidla mají zejména bezpečnostní charakter, tedy mají zajistit bezpečnost jak řidiče, tak i okolí vozidla. Jejich splnění se kontroluje na technických přejímkách, jimiž musí vozidlo před účastí na zmiňovaných závodech vždy projít.

Prvotnímu návrhu se věnovala má bakalářská práce [3] z akademického roku 2019/20. V této práci byla provedena rešerše metod, které je možné použít pro snímaní pozice pedálu a v rámci této práce vznikl i první prototyp snímače využívající dvojici Hallovo sond jako senzorů polohy. Tento prototyp je funkční, ovšem má řadu nedostatků. Především je to nešťastné umístění senzorů za sebe v řadě, což vyžaduje složitější zpracování signálů, a společné napájecí cesty, což není v souladu s pravidly FSAE [1] a FSG [2]. Díky těmto nedostatkům není vhodné tento prototyp použít na vozidle, nicméně dobře posloužil k získání důležitých poznatků v oblasti snímání polohy magnetickou cestou. I proto se konstrukce nového typu bude ubírat tímto směrem, tedy snímáním polohy pomocí dvojice Hallovo sond.

Cílem této práce je definovat vhodné geometrické uspořádání, které nemá zmiňované nectnosti, navrhnout mechanickou část snímače, elektrické schéma i desku plošných spojů, která bude v rámci této diplomové práce i oživena a naprogramována. Celý projekt bude řešen s velkým důrazem na shodu s pravidly FSAE a FSG.

1 Pravidla FSAE a FSG

Jak již bylo naznačeno v úvodu této práce, konstrukce elektronického snímače polohy pedálu elektroformule podléhá, stejně jako ostatní prvky vozidla, pravidlům FSAE [1] a FSG [2]. Tato kapitola se proto věnuje rešerši části těchto pravidel platících pro snímání polohy pedálu. Z této části pravidel bude vycházet následná konstrukce.

1.1 Zpráva FMEA

Před účastí na závodech je nutné v čas odevzdat dokumentaci k celému vozidlu pořadatelům soutěže [1], zvláště pak tzv. „Failure Modes and Effects Analysis“ – zkráceně FMEA. Jedná se o tabulku, do které se musí vyplnit veškeré chyby, které mohou kdekoli na vozidle nastat, jejich důsledky, důvody, a jak je jim předcházeno [1].

1.2 Plynový pedál

Senzory akcelerace je nutné ovládat nožním pedálem. Dráha pedálu se udává v procentech, kde plně sešlápnutý pedál odpovídá 100 % a uvolněný 0 %. Pedál se musí sám vracet do nulové (počáteční) polohy pomocí nejméně dvou vratných pružin, přičemž jedna pružina musí mít takovou sílu, aby i ona sama spolehlivě vrátila pedál do nulové polohy. Případné pružiny, které obsahuje snímač polohy, se jako vratné neuznávají. Taktéž musí být pedálová konstrukce opatřena dorazy, aby se nemohly nijak poškodit snímací prvky, nebo se zabránilo přejetí mimo snímací dráhu. [1] [2]

1.3 Senzory polohy plynového pedálu

Pro snímání polohy pedálu je nutné použít nejméně dva nezávislé senzory polohy [1]. Senzory nesmí mít společné signálové ani napájecí cesty [2]. Každý senzor musí mít rozdílnou strmost odezvy na stlačení pedálu (offset), přičemž musí mít pozitivní směrnici, tzn. musí se stlačením růst [1]. Výstupní charakteristiky se navíc nesmí protínat [2]. Tyto požadavky mají zajistit, že i při zkratu signálových linek mezi sebou dojde ke spolehlivé detekci poruchy a zastavení vozidla.

1.4 Vyhodnocení věrohodnosti signálů

Signály z jednotlivých senzorů polohy je nutno neustále kontrolovat a porovnávat. Tím se ověřuje jejich věrohodnost. Nevěrohodnost je v pravidlech [1] definována jako více než

10% rozdíl v polohách změřených jednotlivými senzory. Použití větší tolerance je možné, ale je nutné ji řádně zdůvodnit ve zprávě FMEA, přičemž není jisté, že bude schválena [1].

Taktéž je nutno detekovat jakékoli poruchy signálové linky typu zkrat na kostru, zkrat na napájecí zdroj, či rozpojení obvodu [1] [2]. Pokud jsou senzory analogové, je to nutno řešit tak, že každý senzor pracuje v určitém rozsahu napětí (např. 0,5 až 4,5 V). V případě vzniku některé ze zmiňovaných poruch, signál ujede mimo tento pracovní rozsah, což je ihned detekovatelné.

Pro případ rozpojení obvodu musí být zařazen pull-up či pull-down rezistor, který opět zajistí snadnou detekci [1].

Nevěrohodnost, která trvá déle než 100 ms, musí vyústit v poruchu a okamžité vypnutí motorů, kdy není nutné vypínat kompletní pohonný systém, stačí, když měniče nedodávají výkon [1] [2].

Pokud jsou využívány tři senzory polohy, je možné při nevěrohodnosti signálu z jednoho z nich, při souhlasu zbylých dvou (10% tolerance), třetí senzor ignorovat a provozovat vozidlo jen při využití senzorů dvou [1] [2].

Bezpečnostní mechanismy ověřující správnou funkci senzorů musí být možné zkontrolovat během technické inspekce vozidla [2]. Proto musí být v cestě signálů zařazen konektor, jehož rozpojením je možno otestovat zmiňovaná bezpečnostní opatření, nebo musí být v cestě signálů umístěna krabice s rozpojovacími prvky z téhož důvodu [1].

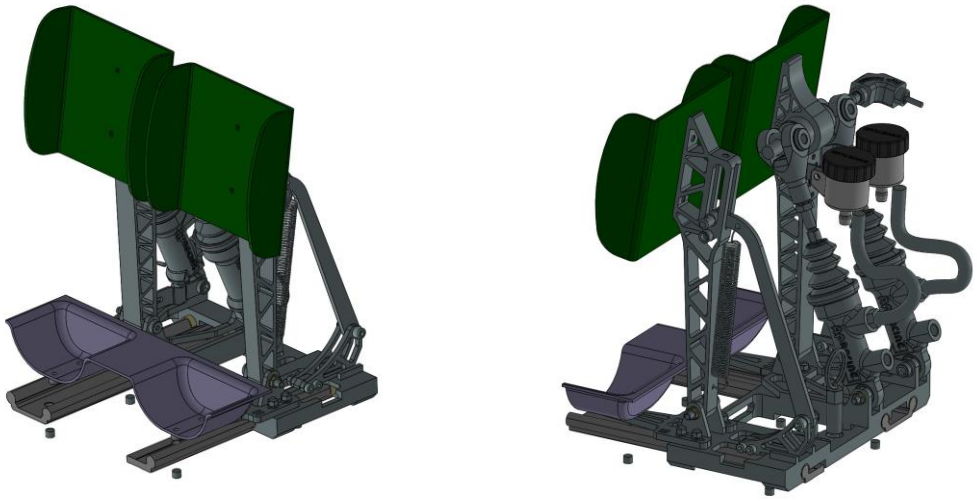
V případě použití digitálních přenosů (CAN, Flex Ray) musí být ve zprávě FMEA řádně zdokumentovány všechny možné poruchové stavy, způsoby jejich detekce a také testy, jakými byly ověřeny [1].

2 Návrh konstrukce snímače

Tato kapitola se zabývá mechanickou konstrukcí snímače a výběrem vhodného umístění, jak snímače jako celku vůči konstrukci pedálu, tak výběrem vhodného rozložení senzorů (magnetu a sond) tak, aby se docílilo co nejlepších vlastností.

2.1 Konstrukce pedálu

V minulosti byla na univerzitě v rámci fakulty strojní postavena studentská formule se spalovacím motorem. Poznatky a díly vyvinuté studenty na této formuli se jistě uplatní i na nové elektrické verzi. Plynový pedál, který je na spalovací formuli, dosud tahal za lanko vedoucí na škrtící klapku spalovacího motoru, kde byl namontovaný senzor natočení. Konstrukce pedálu bude velmi podobná, jen se bude jeho poloha snímat pomocí určitého přepákování přímo tzn. bez lanka. Model původního pedálu je možno vidět na obrázku níže (Obr. 2.1).



Obr. 2.1: Konstrukce pedálů spalovací formule

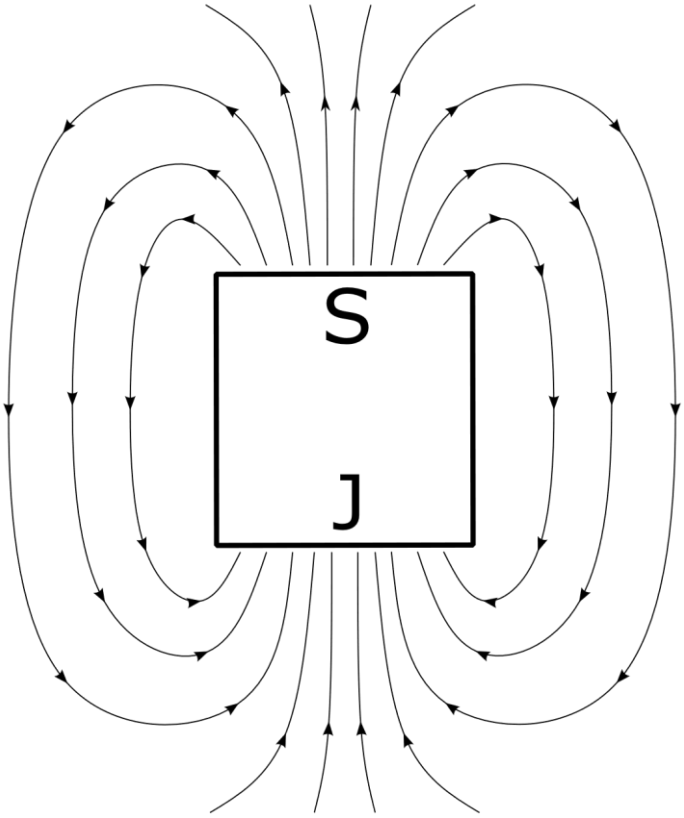
2.2 Výběr magnetického systému

Jak již bylo zmíněno, snímaní polohy bude řešeno pomocí magnetu a Hallovo sond. Při konstrukci jakéhokoli snímače je výhodné, aby odezvy snímacích prvků (senzorů) byly lineární. Odpadají tím následné problémy při zpracovávání signálů.

Existuje několik možností uspořádání měřící sestavy magnet-sonda, které se zásadně liší, nejen v linearitě, ale i citlivosti a délce snímatelné dráhy. Mluvíme zde samozřejmě o Hallovo sondách s určitou formou lineárního výstupu.

2.2.1 Magnetické pole v okolí magnetu

V okolí permanentního magnetu působí magnetické pole. To je znázorňováno pomocí tzv. siločar. Tyto siločáry znázorňují magnetický tok, který se uzavírá od severního pólu magnetu k jižnímu (Obr. 2.2) [4]. Tečna těchto siločar v kterémkoli bodě je pak směrnicí vektoru magnetické indukce v tomto bodě [4].



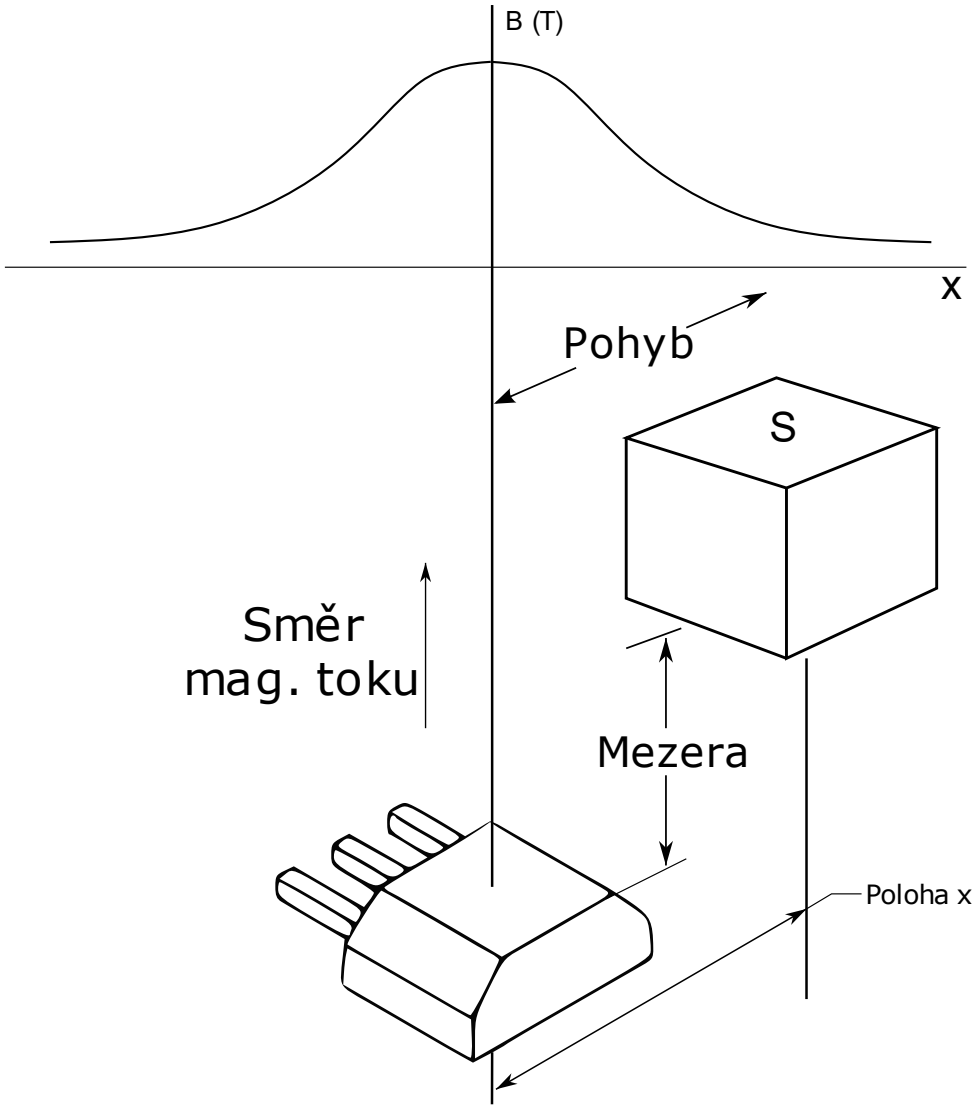
Obr. 2.2: Znázornění magnetických siločar v okolí permanentního magnetu [4]

2.2.2 Umístění Hallovo sondy vůči magnetu

Jak je známo z principu Hallovo sondy, napětí na jejím výstupu je úměrné vektorovému součinu vektoru elektrického proudu a magnetické indukce [4]. Hallovo sondu lze jedním magnetem budit dvěma způsoby [4]:

- unipolárně – magnet je čelem (pólem) k sondě (Obr. 2.3)
- bipolárně – magnet je bokem k sondě (Obr. 2.4).

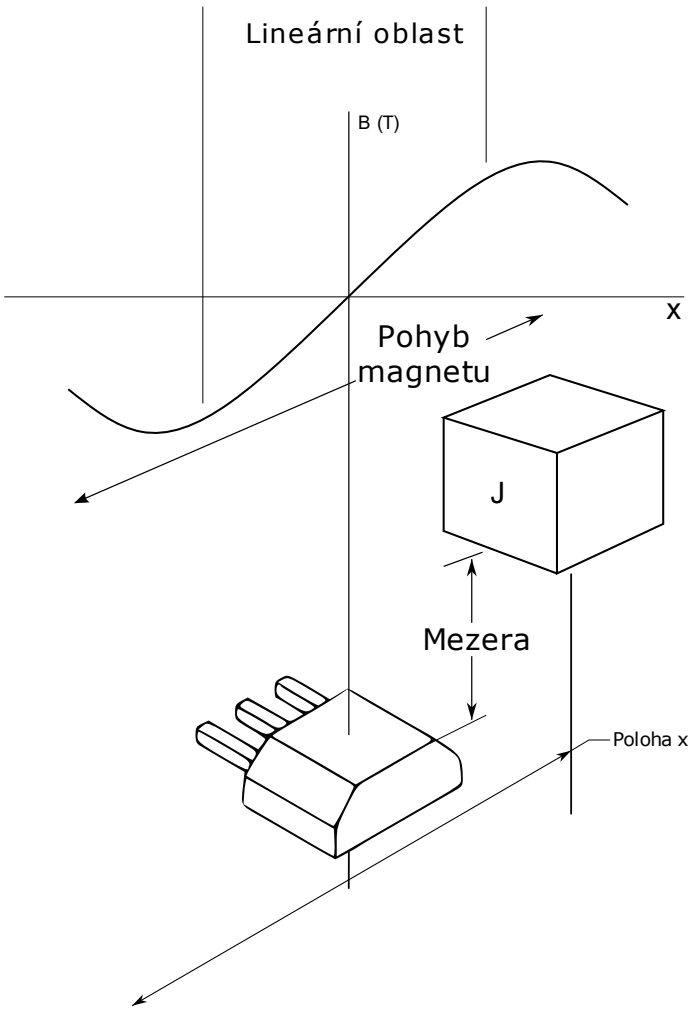
Umístění magnetu čelem k sondě má vzhledem k siločarám nelineární charakter, ať už magnet suneme směrem k sondě, či podél sondy (Obr. 2.3) [4].



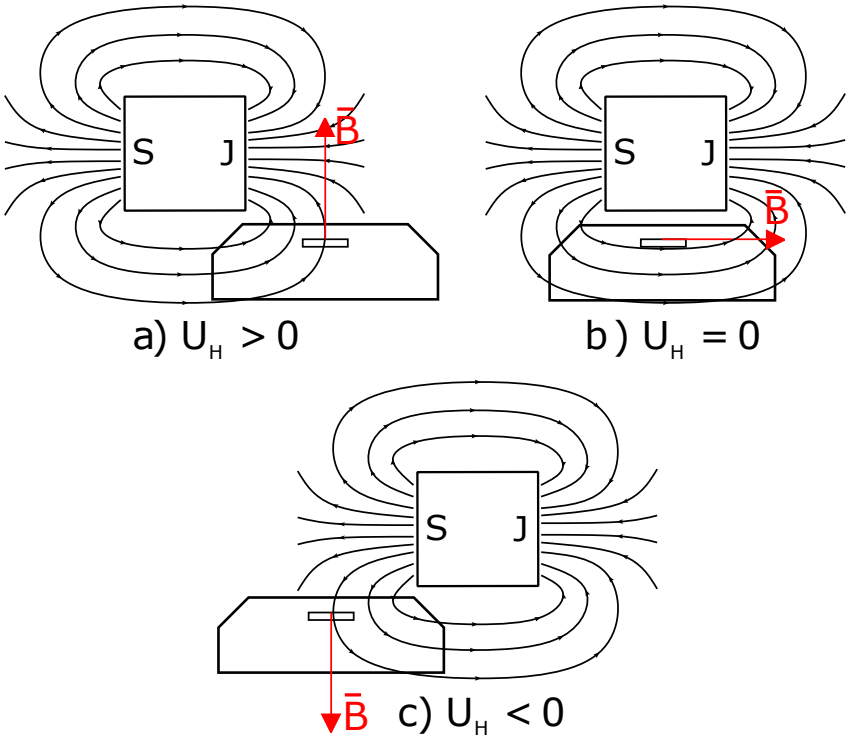
Obr. 2.3: Unipolární posuvné uspořádání [4]

Není tomu tak u buzení bipolárního (Obr. 2.4). Na začátku dráhy siločáry protínají Hallovo sondu kolmo a sonda je vybuzena v jedné polaritě (Obr. 2.5a). Při postupném sunutí magnetu se mění úhel protínajících siločar vůči sondě a vybuzení (Hallovo napětí) klesá k nule. To nastane právě, když je magnet přímo před sondou tj., když se středová osa magnetu kryje s osou středu sondy – siločáry protínají destičku rovnoběžně (Obr. 2.5b). S dalším postupným posunem opět vzrůstá vybuzení sondy vzhledem k úhlu protínajících siločar, ovšem v druhé polaritě (Obr. 2.5c). Nejvyššího vybuzení je opět dosaženo při protnutí sondy siločárami kolmo. Tento bod se hodí označit jako konec použitelné dráhy, jelikož dále by vybuzení, resp. Hallovo napětí sondy, opět klesalo a způsobilo nejednoznačnost polohy.

Průběh magnetické indukce v závislosti na výše popsaném posunu má v určité oblasti lineární charakter, což je důvod, proč bude využit při konstrukci.



Obr. 2.4: Bipolární posuvné uspořádání [4]



Obr. 2.5: Princip bipolárního buzení při posuvu

Jak je viditelné z obrázků výše, rozdíl je i v měřitelné dráze. U bipolárního uspořádání je dráha dvounásobná. Obě varianty jsou závislé na mezeře mezi sondou a magnetem, a proto je nutné zabezpečit, aby se během měření co nejméně měnila.

2.2.3 Výběr Hallovo sondy

Sondy musí mít určitou formu lineárního výstupu. Na trhu se nacházejí senzory s lineárními analogovými napěťovými výstupy, s výstupy PWM, sběrnicí IIC nebo SPI.

IIC a SPI není pro tento případ použití ve formuli dostatečně robustní, a proto budou z výběru rovnou vyřazeny. V pravidlech jsou zmíněny jen analogové výstupy nebo sběrnice CAN či FlexRay [1]. PWM zde tedy výslovně zakázána není, nicméně pro jistou shodu s pravidly bude vhodnější využít analogový výstup.

Jak bylo zmíněno v předchozí kapitole, výstupy senzorů by měly využívat jen určitý rozsah napětí. To bude jedním z hlavních kritérií výběru, spolu s citlivostí a rozsahem měřené magnetické indukce *B*.

Výběr padl na senzor od firmy Texas Instruments – DRV5055-Q1. Ten má již integrované pomocné obvody (zesilovač, kompenzace). Je přímo vyvinut pro automobilové aplikace a splňuje kvalifikaci AEC-Q100. [5]

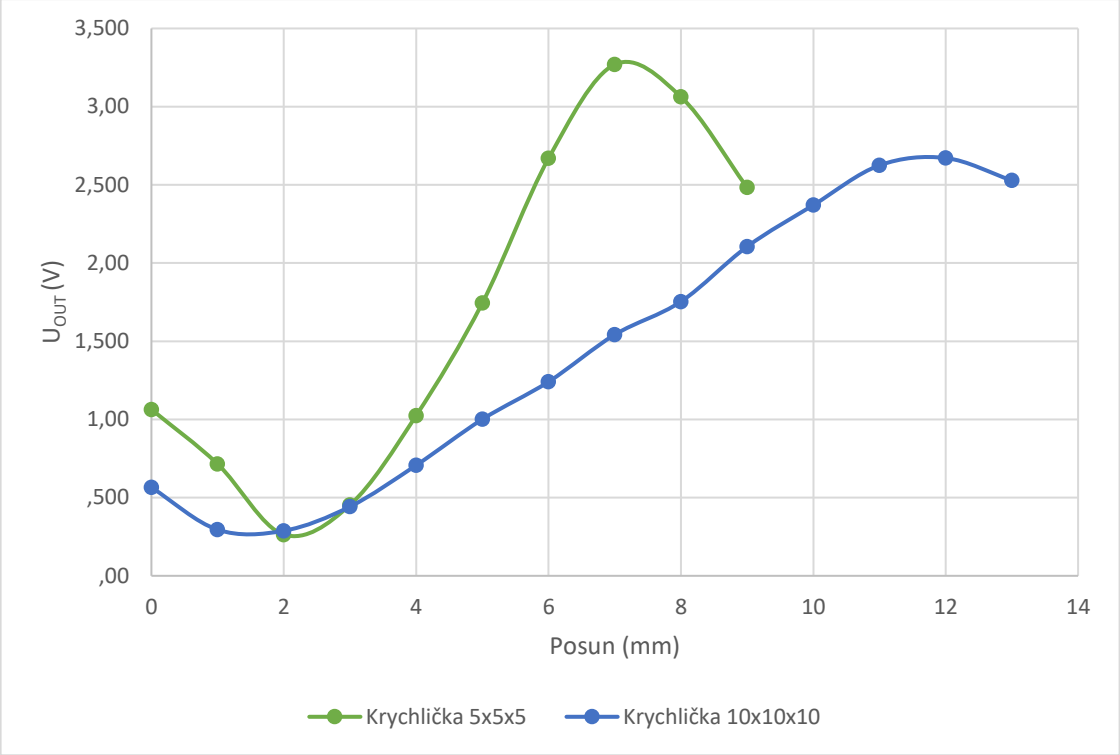
Senzor má lineární analogový výstup. Jeho výhodou je, že je možné napájet jej ve dvou napájecích úrovních, totiž 3,3 nebo 5 V. Od velikosti napájecího napětí se odvíjí i výstupní napěťové úrovně. [5] Toho se dá využít pro zajištění rozdílných sklonů a offsetů sond. Při nulové magnetické indukci je pak na výstupu polovina napájecího napětí [5].

Tento typ se vyrábí v několika variantách citlivostí a rozsahů měřitelné magnetické indukce. Obecně platí, že pro větší rozsahy magnetické indukce dostaneme menší citlivost senzoru. Pro variantu s rozsahem magnetické indukce $B = \pm 850$ Gauss (± 85 mT) je citlivost 1,5 mV/Gauss (15 mV/mT). Senzor je dostupný jak ve variantě pro povrchovou montáž, tak pro THT technologii. [5]

2.2.4 Měření vlastností magnetického systému

Bylo objednáno několik zástupců Hallovo sond DRV5055-Q1 s rozdílnou citlivostí. Senzor se umístil do nepájivého pole a neodymový magnet do přípravku z dřevěného prkna. Magnet byl posouván nad senzorem a posun měřen na pravítku. Toto měření bylo provedeno pro několik magnetů, několik druhů senzorů s různou citlivostí, a s různými mezerami mezi senzory a magnety. Měnilo se i napájení senzorů z 3,3 na 5 V.

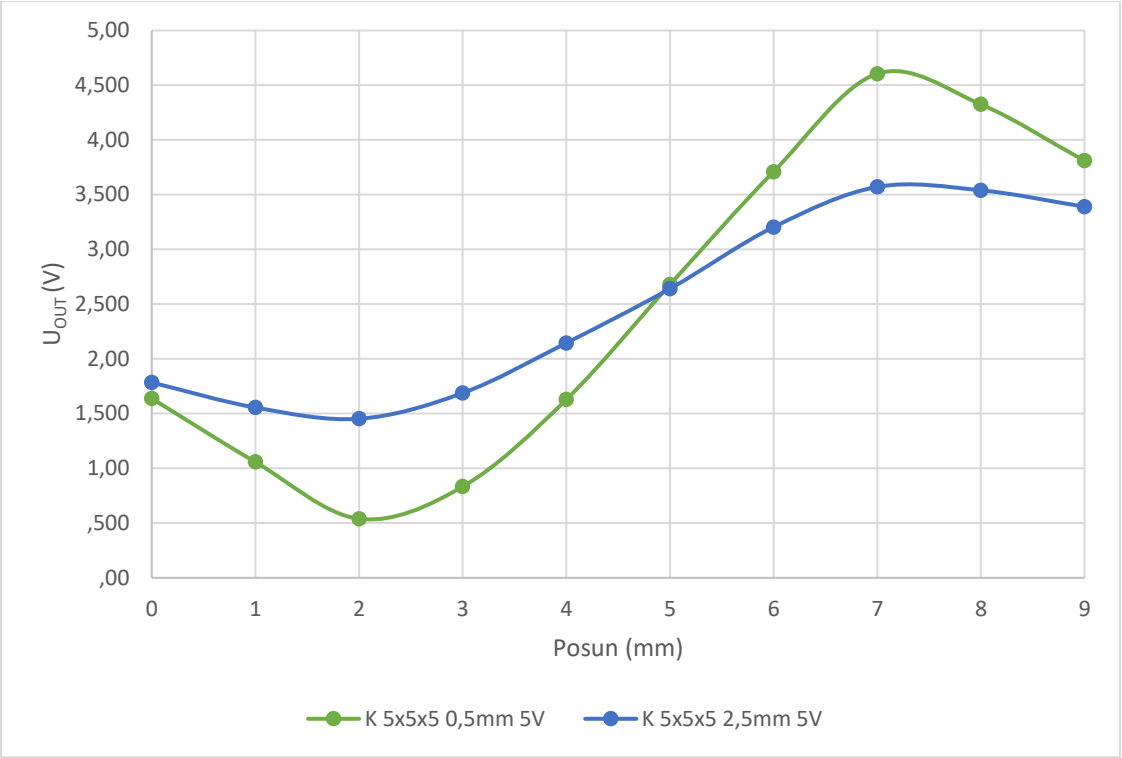
Nejlepší výsledky vycházejí pro neodymové magnety ve tvaru krychličky. Obrázek níže (Obr. 2.6) ilustruje rozdíl mezi krychličkou o straně 5 a 10 mm. Je patrné, že délka detekovatelné dráhy je úměrná straně magnetu.



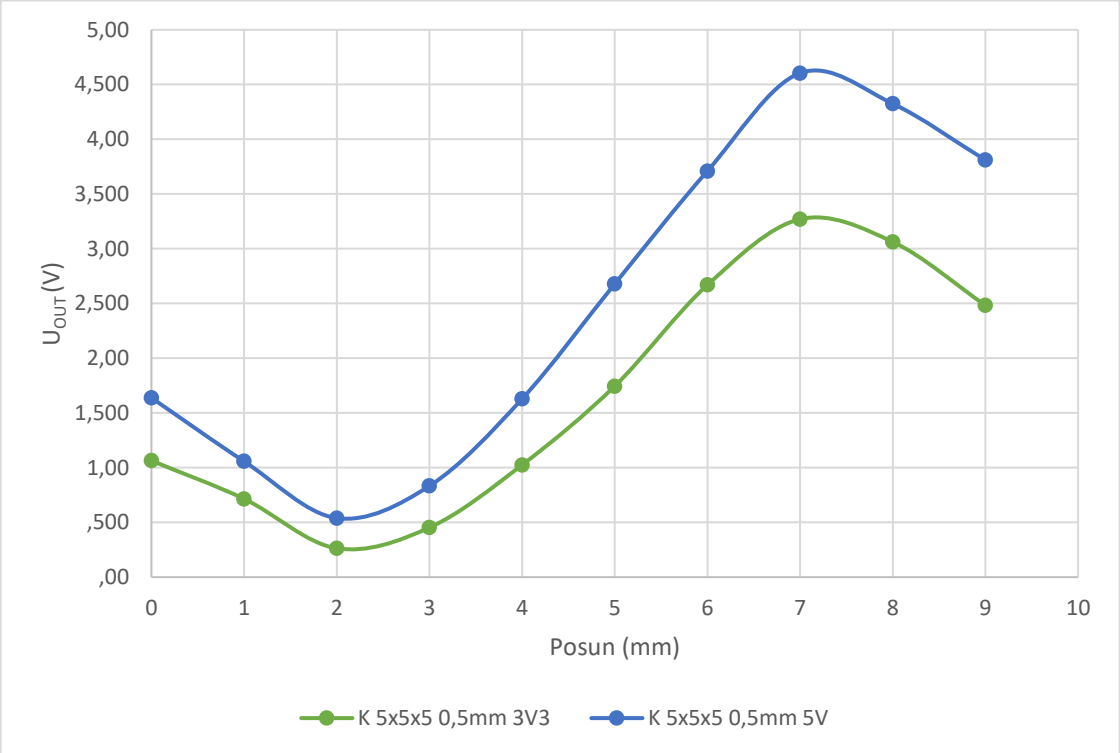
Obr. 2.6: Porovnání bipolárního buzení sondy různými magnety

Rozdíl ve velikosti vybuzení je způsoben lehce odlišnou vzdáleností mezi magnetem a Hallovo sondou. To, jak je tento vliv markantní, ilustruje obrázek dále (Obr. 2.7). Je zřejmé, že s větší vzdáleností magnetu od senzoru klesá velikost jeho maximálního vybuzení. Pokud by se tak dělo i v konstruovaném snímači, znamenalo by to vážný problém s bezpečností, poněvadž by i při uvolněném pedálu mohla být při velké mezeře vyhodnocena dráha sešlápnutí i několik desítek procent (v extrému až 50 %). Taktéž je velikostí mezery zmenšována lineární oblast grafu, což je také nežádoucím efektem.

Poslední graf (Obr. 2.8) znázorňuje změnu charakteristik Hallovo sond při napájení 3,3 a 5 V.



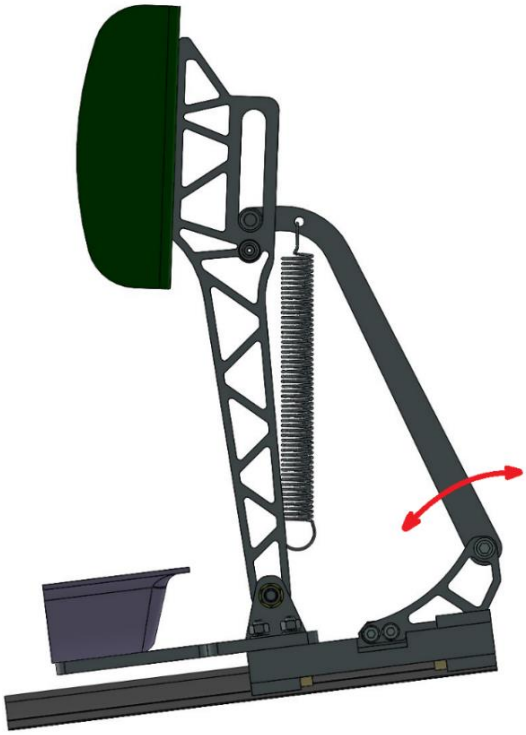
Obr. 2.7: Vliv mezery na výslednou charakteristiku (napájení 5 V) – magnet 5x5x5 mm



Obr. 2.8: Charakteristiky při napájení 3,3 a 5 V

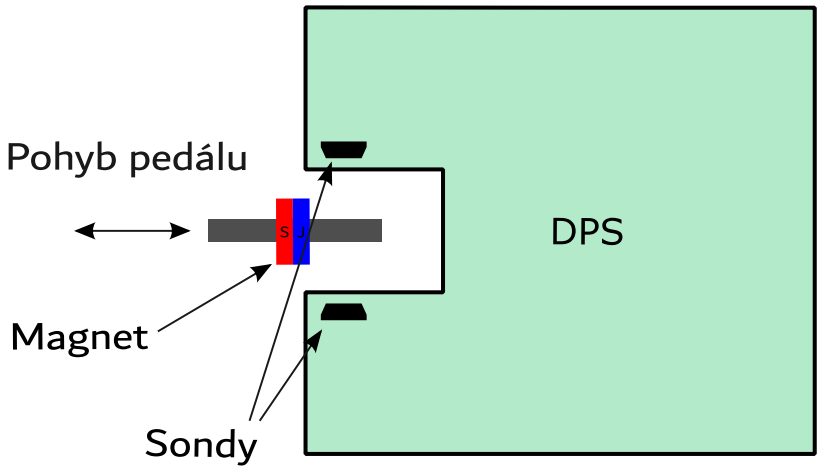
2.3 Konstrukce snímače

Při prvním prohlédnutí modelu pedálů formule (Obr. 2.1) se nabízela otázka, zdali by nebylo možné využít pohyb již existujícího přepákování (Obr. 2.9).



Obr. 2.9: Akcelerační pedál s vyznačeným místem původně zamýšleného snímání

Hlavní ideou bylo, že by se na sekundární rameno pedálu umístil magnet a na každou stranu přepákovaného ramena se umístila jedna Hallova sonda jako senzor. Celý snímač by pak mohl mít podobu jedné DPS, s výřezem uprostřed, přičemž rameno s magnety by se při sešlápnutí pedálu zasunovalo do tohoto výřezu – mezi senzory (Obr. 2.10).



Obr. 2.10: Znázornění původně zamýšlené konstrukce

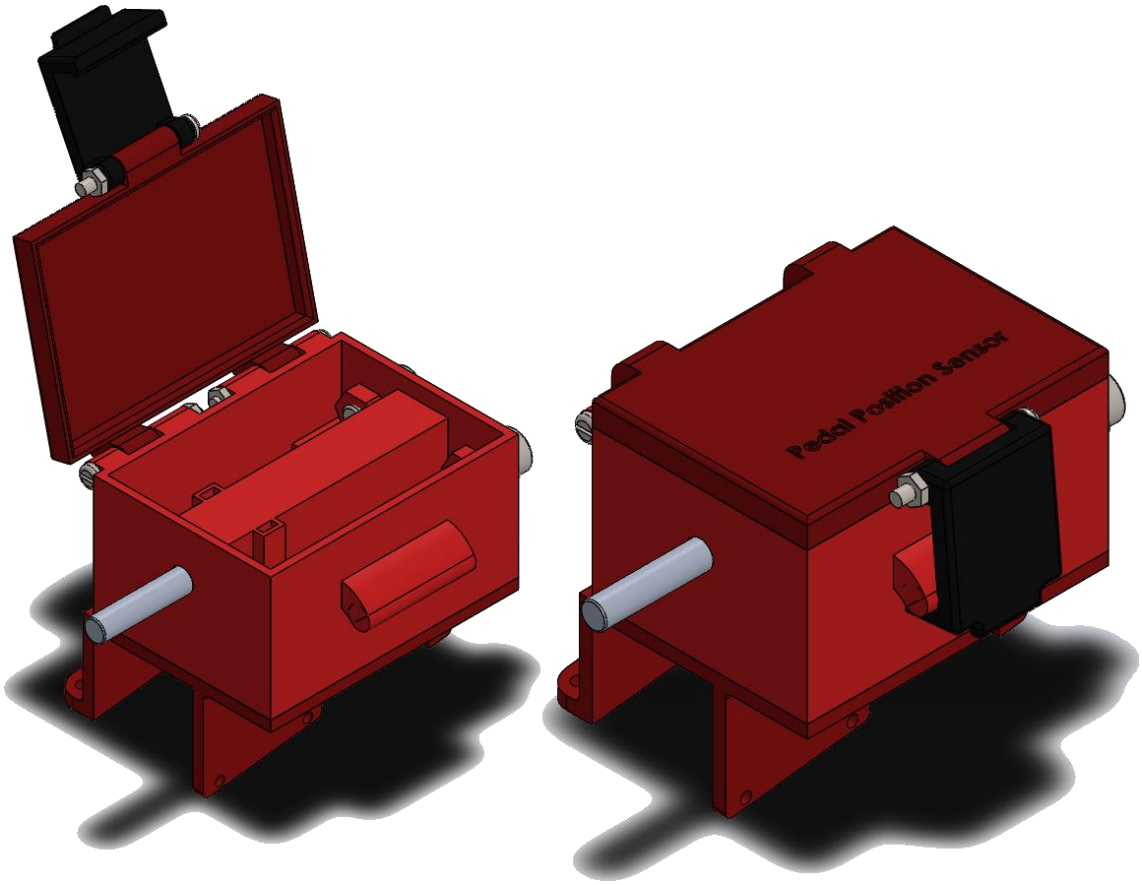
Toto řešení má však svá úskalí. Sekundární rameno, které by drželo magnety, má určitou vůli ve svém čepu. Ta mu dovoluje pohybovat se nejen žádoucím směrem, ale i do stran, a to o přibližně 2 mm. To znamená, že by se mohla měnit i mezera mezi sondou a magnetem o tyto 2 mm. To je v takovém měřítku nepřijatelné, jak ukazuje provedené měření.

Navíc má toto provedení i další nevýhodu v podobě absence ochrany magnetu před např. železným prachem či jinými magnetickými materiály, které by mohly na permanentním magnetu ulpívat.

Z výše uvedených důvodů byla myšlenka upravena. Magnet je nyní umístěn v krytu – v tunýlku, kde je chráněný před vlivy okolí, prachu či jiných nečistot. Zároveň tunýlek poskytuje dostatečně přesné vedení, a tedy vymezuje mezeru mezi magnetem a sondami. Spojení s pedálem zaštiťuje díl se závitem, ke kterému bude vyvinuto vhodné přepákování z pedálu.

2.3.1 Model snímače

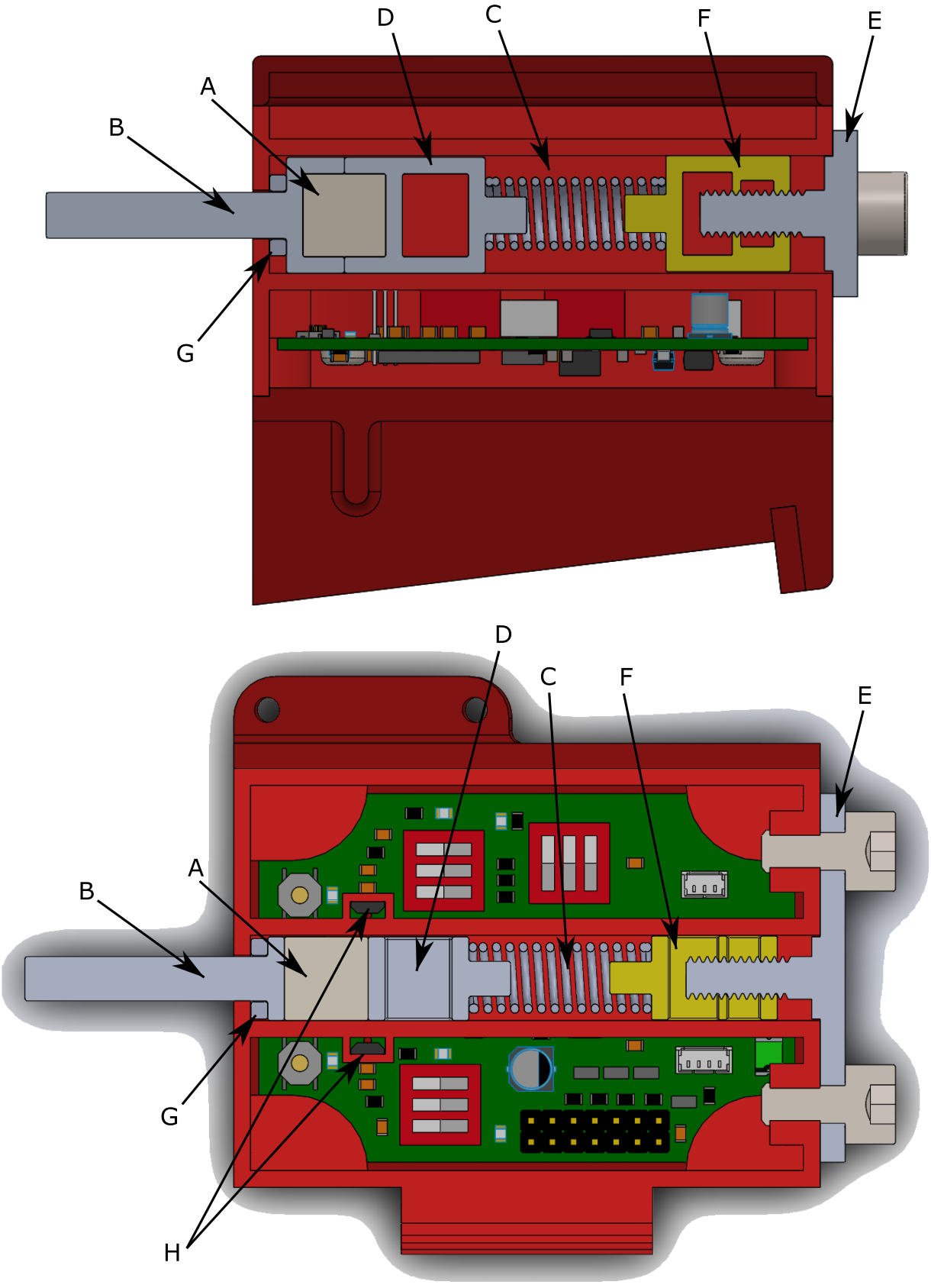
Zmíněné uspořádání bylo namodelováno v programu SolidWorks (Obr. 2.11). Hlavní pouzdro se skládá z těla a dvou vík. Spodní víko je k tělu šroubované a slouží k vložení DPS a uchycení celého snímače ke konstrukci pedálů. Plošný spoj je na svém místě zajištěn čtyřmi šrouby. Horní kryt má panty a jeho zajištění je realizováno zacvakávacím mechanismem. Je tomu tak kvůli nutnosti snadného přístupu k přepínačům sloužícím k technické inspekci či kalibraci nebo nastavení zařízení.



Obr. 2.11: Model pouzdra snímače, vlevo otevřený, vpravo zavřený

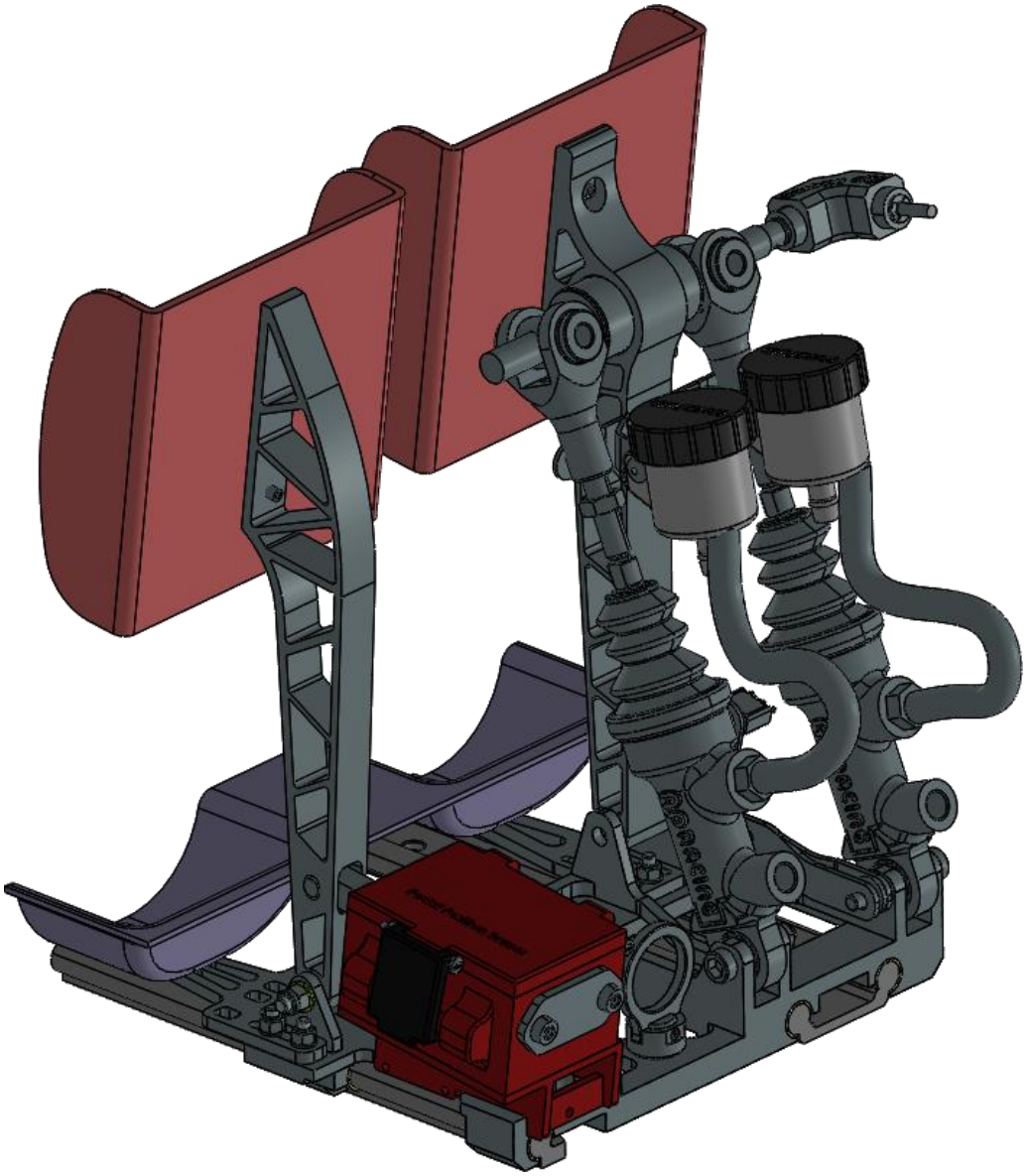
Součástí hlavního těla krabičky je i tunel na magnet, který je znázorněn v řezu na obrázku níže (Obr. 2.12). Magnet (*A*) se do tunelu vkládá otvorem z boku krabičky a je zalepen v nástavci (*B*) zajišťujícím správné vedení magnetu a venkovní spojení s pedálem. Tento díl musí být vyrobený z nemagnetického materiálu, aby nedocházelo k nežádoucím deformacím magnetického pole. Dále je v tunelu vratná pružina (*C*). Ta je ze stejného důvodu držena dále od magnetu druhým nemagnetickým nástavcem (*D*). Celý tunel je uzavřen krytem (*E*), jehož funkcí je i nastavení dorazu (*F*). Doraz na začátku dráhy tvoří podložka (*G*). Hallovo sondy (*H*) jsou drženy na svém místě pouzdry na stěně tunelu s magnetem.

Jak již prokázalo měření, délka dráhy je úměrná délce strany krychlového magnetu. Jelikož bylo zařízení projektované na dráhu (doraz – doraz) 1 cm, byl zvolen magnet o straně 1 cm.



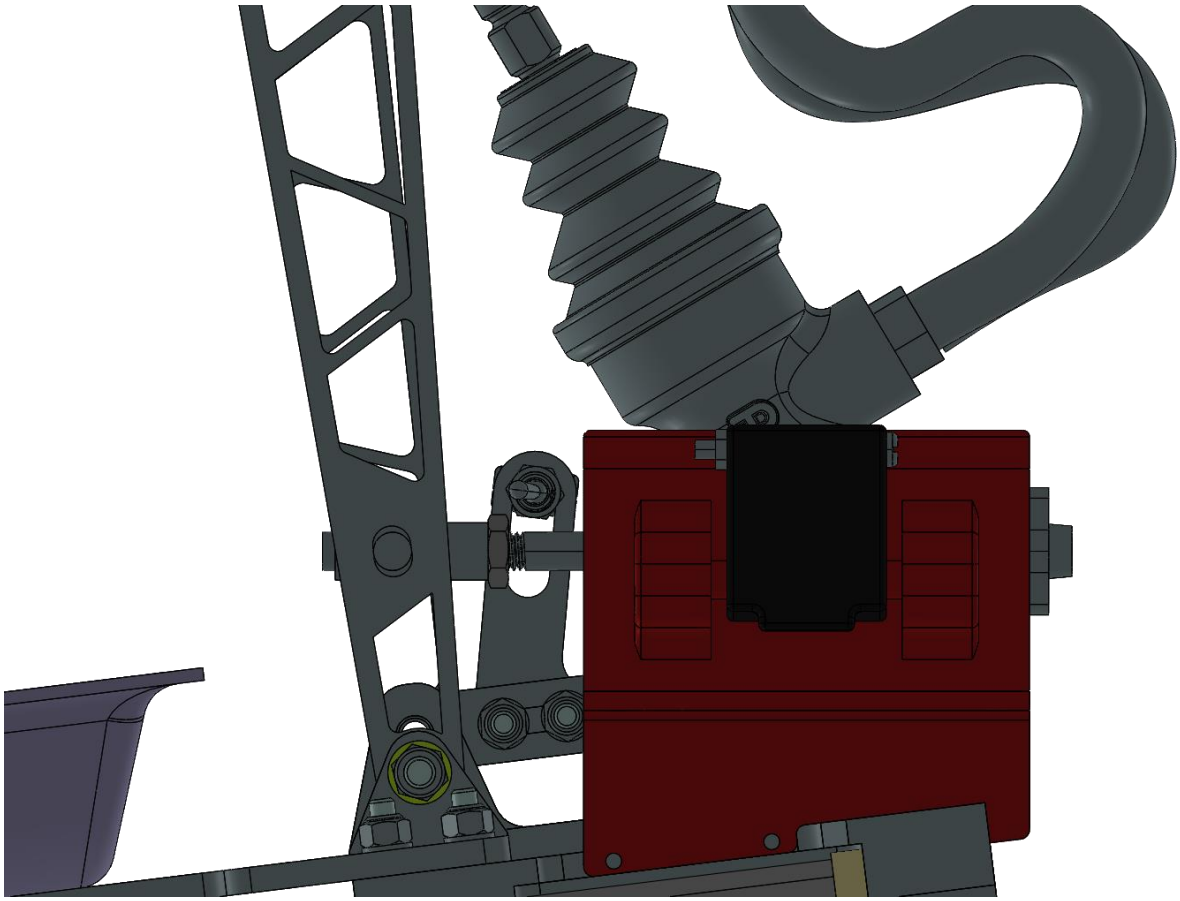
Obr. 2.12: Zobrazení snímače v řezu

Takovýto snímač pak může být napasován na původní pedály z obrázku výše (Obr. 2.1) například způsobem, který ukazuje následující obrázek níže (Obr. 2.13).



Obr. 2.13: Umístění snímače

Aby mohl být snímač pevně uchycen, musí být v pedálu vytvořena drážka s čepem dovolujícím pohyb nahoru, nikoli však vytvářející vůli ve směru sešlápnutí. To je viditelné na dalším obrázku (Obr. 2.14).



Obr. 2.14: Uchycení snímače na bloku pedálů, pohled z boku

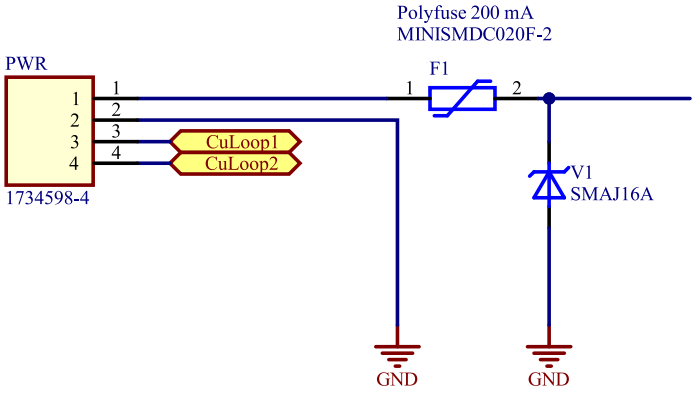
3 Vývoj elektrického schématu

Tato kapitola se zaměřuje na postup při návrhu elektrického schématu snímače polohy pedálu. Schéma zařízení lze rozdělit na tři části:

- napájecí část (Příloha I)
- analogová část (Příloha II)
- digitální část (Příloha III).

3.1 Napájecí část

V napájecí části se nachází vratná pojistka, ochrany proti přepětí, vstupní filtr zajišťující elektromagnetickou kompatibilitu a step-down měnič napětí s LDO stabilizátory na konečnou úpravu napětí. Z nich jsou napájeny veškeré komponenty. Obrázek (Obr. 3.1) ilustruje vstupní obvod.



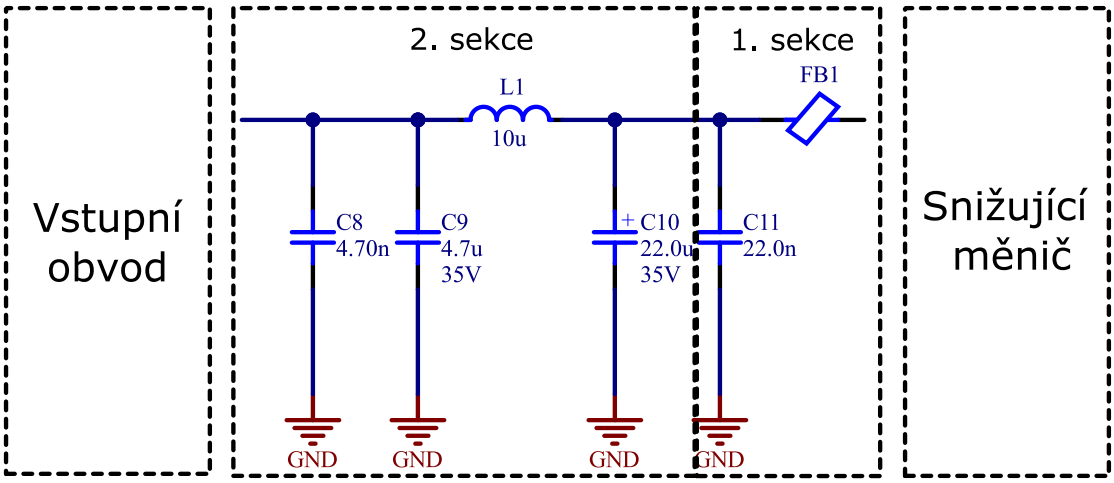
Obr. 3.1: Vstupní obvod

Zařízení je napájeno z nízkonapěťové sítě elektrovozidla o napětí 12 V stejnosměrných. Toto napětí je přivedeno na konektor (*PWR*) společně s kontrolní proudovou smyčkou (tzv. Interlock, ve schématu označeno *CuLoop*), která má za úkol detekci případného rozpojení důležitých konektorů napříč celým vozidlem (viz kapitola 3.3.5). Následuje vratná pojistka (*F1*) na 200 mA a ochranná TVS dioda (*V1*) s prahovým napětím 16 V.

3.1.1 Vstupní filtr

Vstupní filtr je na obrázku (Obr. 3.2). Jedná se o dvousekční filtr. Jako první sekce je označena vysokofrekvenční část blíže k snižujícímu měniči. Její hlavní částí je feritová perlička (*FB1*), jejíž umístění je nezbytně nutné volit co nejblíže měniči [6]. Tím je zajištěno okamžité zatlumení vysokofrekvenčního rušení dříve, než by se mohlo navázat do jiných míst [6]. Se vzdáleností se tedy účinnost feritové perličky snižuje [6].

Druhou sekcí je označen hlavní filtr. Skládá se z indukčnosti (*L1*) a kondenzátorů (*C8*, *C9* a *C10*).



Obr. 3.2: Vstupní filtr

Občas se volí filtry třísekční, kdy se ve třetí sekci nachází tlumivka pro potlačení souhlasného rušení. V tomto případě, kdy je napájení jen dvouvodičové a mínusový pól je spojen s kostrou, její použití nemá žádný přínos, poněvadž se souhlasný signál obvykle uzavírá přes kapacitu vůči kostře (ochrannému vodiči) [7]. Proto je nyní tato sekce úplně vypuštěna.

Hodnoty součástek se navrhují dle hodnot potřebného zatlumení na spínací frekvenci. To lze zjistit buď předešlým měřením, či určitou aproximační metodou [7]. Pro návrh byla využita metoda modelování vstupního proudu jako ideálního obdélníkového průběhu dle [7]. Potřebné zatlumení se pak určí analytickým vzorcem:

$$|A|\ (\mathrm{dB\mu V}) = 20\log\left(\frac{\frac{I}{\pi^2 f_S C_{IN}}\cdot\sin(\pi D)}{1\mu\mathrm{V}}\right) - L_{MAX}, \qquad (3.1)\ [7]$$

kde A je potřebné zatlumení, I výstupní proud, f_S frekvence spínání, D střída a C_{IN} kapacita vstupního kondenzátoru měniče. L_{MAX} je pak limitní úroveň šumu daná normou.

Frekvence spínání vychází z použitého step-down měniče a je 500 kHz. Kapacita před konvertorem je dána katalogovým listem [8] C_{IN} = 4,7 µF. Předpokládejme proud I = 200 mA a před LDO stabilizátory chtějme napětí 6,3 V. Při snižování na toto napětí ze vstupních 12 V lze vypočíst střídu spínání dle vzorce:

$$D\ (-) = \frac{U_O}{U_{IN}\cdot\eta} = \frac{6{,}3}{12\cdot 0{,}94} = 0{,}559, \qquad (3.2)$$

kde η je účinnost (předpokládaná 94 %), U_O je výstupní napětí a U_{IN} vstupní napětí konvertoru. Po dosazení předpokládaných hodnot s hodnotou střídy do vztahu (3.1), dostaneme výpočet (3.3) a z něj potřebné zatlumení 33,6 dBμV.

$$|A| = 20\log\left(\frac{\frac{0{,}2}{\pi^2 \cdot 500 \cdot 10^3 \cdot 4{,}7 \cdot 10^{-6}} \cdot \sin(\pi \cdot 0{,}559)}{1\mu\text{V}}\right) - 45 = 33{,}6\ \text{dB}\mu\text{V} \tag{3.3}$$

Indukčnost L_f (*L1*) určuje rezonanční kmitočet filtru [7] a je volena pro nízkopříkonové aplikace v rozmezích od 1 do 10 μH. Je nejlepší zvolit co největší indukčnost s přihlédnutím k rozměrům cívky [7]. Vzhledem k nízkému proudu není rozměr pro tuto aplikaci nijak drastický a je tedy zvolena indukčnost L_f = 10 μH.

Velikost filtračního kondenzátoru C_f (*C9*) lze určit z následujících vztahů (3.4, 3.5) [7]. Volí se hodnota z toho vztahu, která je vyšší.

$$C_{fA}\ (\text{F}) = \frac{C_{IN}}{C_{IN}L_f \cdot \left(\frac{2\pi f_S}{10}\right)^2 - 1} \tag{3.4 [7]}$$

$$C_{fB}\ (\text{F}) = \frac{1}{L_f} \cdot \left(\frac{10^{\frac{|A|}{40}}}{2\pi f_S}\right)^2 \tag{3.5 [7]}$$

Dosazením do vztahů dostáváme výpočty:

$$C_{fA} = \frac{4{,}7 \cdot 10^{-6}}{4{,}7 \cdot 10^{-6} \cdot 10 \cdot 10^{-6} \cdot \left(\frac{2\pi \cdot 500 \cdot 10^3}{10}\right)^2 - 1} = 1{,}3\ \mu\text{F} \tag{3.6}$$

$$C_{fB} = \frac{1}{10 \cdot 10^{-6}} \cdot \left(\frac{10^{\frac{33{,}6}{40}}}{2\pi \cdot 500 \cdot 10^3}\right)^2 = 485\ \text{nF} \tag{3.7}$$

Z výpočtů (3.6) a (3.7) je vyšší hodnota 1,3 μF. Použitý kondenzátor musí mít vyšší kapacitu, než je tato hodnota. Byl zvolen kondenzátor 4,7 μF.

Jako poslední se volí kapacita tzv. damping kondenzátoru C_d (*C10*). Jeho funkcí je snížení impedanční špičky filtru vzniklé parazitní rezonancí součástek. Kondenzátor působí jako blokace stejnosměrné složky napětí. K samotnému snížení impedance se využívá jeho ekvivalentního sériového odporu ESR_d [7]:

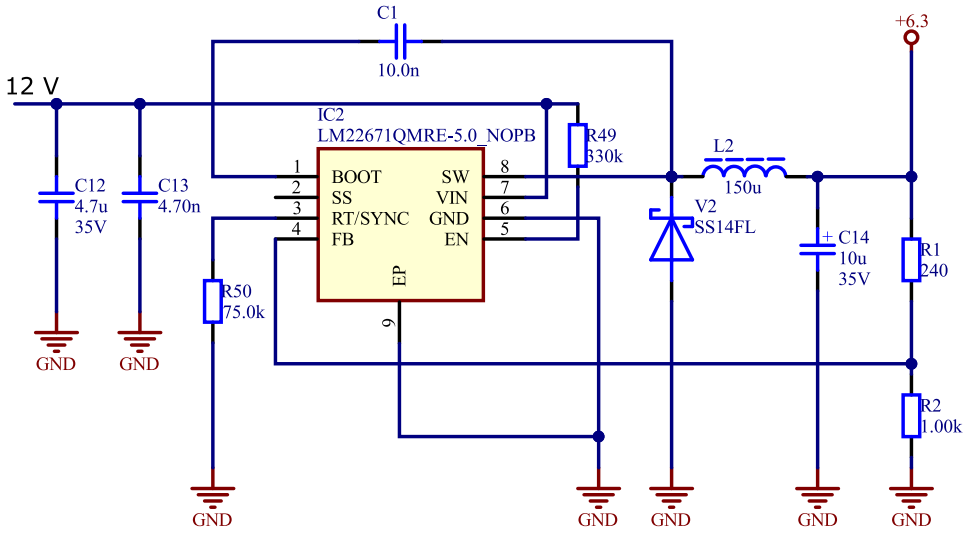
$$C_d \geq 4 \cdot C_{IN} \Rightarrow\ C_d \geq 4 \cdot 4{,}7 \cdot 10^{-6} \Rightarrow\ C_d \geq 18{,}8\ \mu\text{F} \tag{3.8 [7]}$$

$$ESR_d \leq \sqrt{\frac{L_f}{C_{IN}}} \Rightarrow ESR_d \leq \sqrt{\frac{10}{4{,}7}} \Rightarrow ESR_d \leq 1{,}46\ \Omega \tag{3.9 [7]}$$

S přihlédnutím k podmínkám (3.8) a (3.9) byl zvolen elektrolytický kondenzátor 22 µF/25 V s hodnotou *ESR* = 0,76 Ω. Pro další potlačení důsledků parazitních rezonancí jsou přidány další kondenzátory (*C8* a *C11*).

3.1.2 Snižující měnič

Bezprostředně za vstupním filtrem se nachází step-down konvertor s řídícím obvodem LM22671 (Obr. 3.3). Jedná se o spínaný regulátor s integrovaným tranzistorem řízeným polem s kanálem typu N [8].



Obr. 3.3: Snižující měnič s LM22671

Ten působí jako spínač mezi piny *7* (*VIN*) a *8* (*SW*) [8]. Pokud je tento tranzistor sepnutý, dochází k akumulaci energie v indukčnosti (*L2*) a proud indukčností narůstá. V tuto chvíli dioda (*V2*) nevede, jelikož je polarizována závěrně. Po vypnutí tranzistoru (rozepnutí spínače) nemůže proud indukčností okamžitě zaniknout, jelikož je stavovou veličinou. Aby proud nezanikl okamžitě, cívka (*L2*) otočí svou polaritu napětí. Začne se tak chovat jako zdroj. Dioda (*V2*) je díky tomu ve vodivém stavu a umožňuje průtok proudu z indukčnosti do výstupního kondenzátoru (*C14*) a zátěže. Dochází tak k přesunu energie z indukčnosti do kapacity a zátěže.

Dioda (*V2*) je typu Schottky. Je tak zvolena z důvodu její rychlosti, a také z důvodu jejího malého prahového napětí. Díky tomu je na ní menší výkonová ztráta a účinnost konvertoru je tak vyšší.

Na výstupu je zapojen napěťový dělič (*R1* a *R2*) tvořící zápornou zpětnou vazbu. Velikost výstupního napětí tak určuje dělící poměr tohoto děliče. Při návrhu děliče se má dle katalogového listu [8] zvolit jako první spodní rezistor R_{FBB} = 1 kΩ (R2). K této hodnotě R_{FBB} se dle vztahu (3.10) dopočte druhá hodnota rezistoru R_{FBT} (R1):

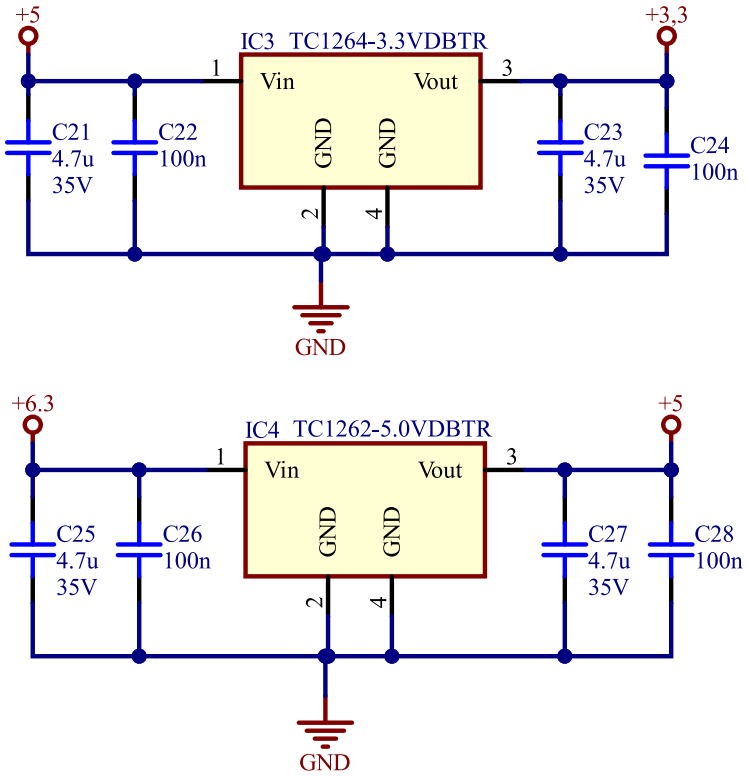
$$R_{FBT} = \frac{R_{FBB} \cdot (U_O - 5)}{5 + R_{FBB} \cdot 5 \cdot 10^{-4}} = \frac{1000 \cdot (6{,}3 - 5)}{5 + 1000 \cdot 5 \cdot 10^{-4}} = 236{,}4\ \Omega \qquad (3.10)\ [8]$$

Jako vrchní rezistor děliče (*R1*) byl zvolen nejbližší vyšší rezistor v řadě, tedy 240 Ω. Napětí vznikající na děliči je dále vedeno na pin *4* (*FB*). Frekvence PWM je u tohoto obvodu v základu f_S = 500 kHz. Lze ji ovlivnit externím rezistorem R_T (R50). Pokud má obvod běžet na své základní frekvenci, je vhodné nenechat pin nezapojený, ale použít rezistor o hodnotě 75 kΩ [8] zapojený vůči zemi.

Kondenzátor (*C1*) je tzv. Boost Capacitor [8]. Tento kondenzátor zajišťuje napětí potřebné k řízení hradla integrovaného tranzistoru.

3.1.3 Lineární stabilizátory

Pro napájení zbylých obvodů jsou třeba dvě napájecí úrovně, a totiž 5 a 3,3 V. Důvod využití dvou napájecích hladin byl již nastíněn ve druhé kapitole (sondy musí mít rozdílné sklony výstupních charakteristik). Napětí se získávají pomocí kaskády dvou lineárních stabilizátorů s nízkým úbytkem napětí (LDO). Prvním je TC1262-5.0V vyrobený firmou Microchip. Jeho vstup je napájen ze snižujícího měniče 6,3 V. Druhým je TC1264-3.3V od stejného výrobce. Je napájen z výstupu stabilizátoru na 5 V. Zapojení je na obrázku níže (Obr. 3.4).



Obr. 3.4: Zapojení lineárních stabilizátorů

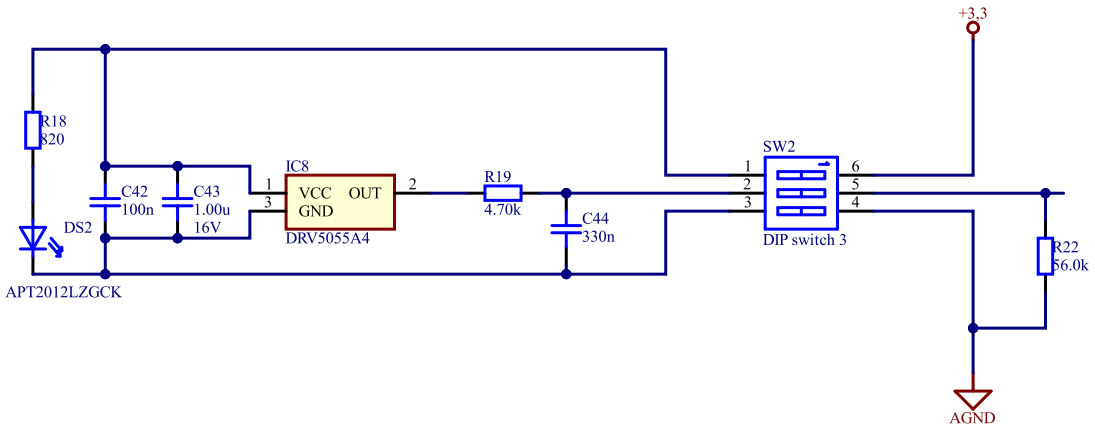
Zapojení obou regulátorů jsou identická. Kromě samotných obvodů jsou zde k vidění i filtrační a blokovací kondenzátory.

3.2 Analogová část

Jádrem této části jsou samotné senzory (Hallovo sondy), jejichž výstupní napětí se dále filtruje aktivní dolní propustí a přivádí přes anti-aliasingové filtry na analogově-digitální převodník v mikrokontroléru.

3.2.1 Snímací část

Obrázek (Obr. 3.5) ukazuje senzor (*IC8*) napájený napájecím napětím 3,3 V. Senzor je blokován dvojicí kondenzátorů (*C42* a *C43*). Přítomnost napájecího napětí je indikována svítivou diodou (*DS2*). Omezení proudu diodou realizuje rezistor (*R18*). Pro senzor napájený napětím 5 V je zapojení totožné, jen omezovací rezistor LED má jinou hodnotu.

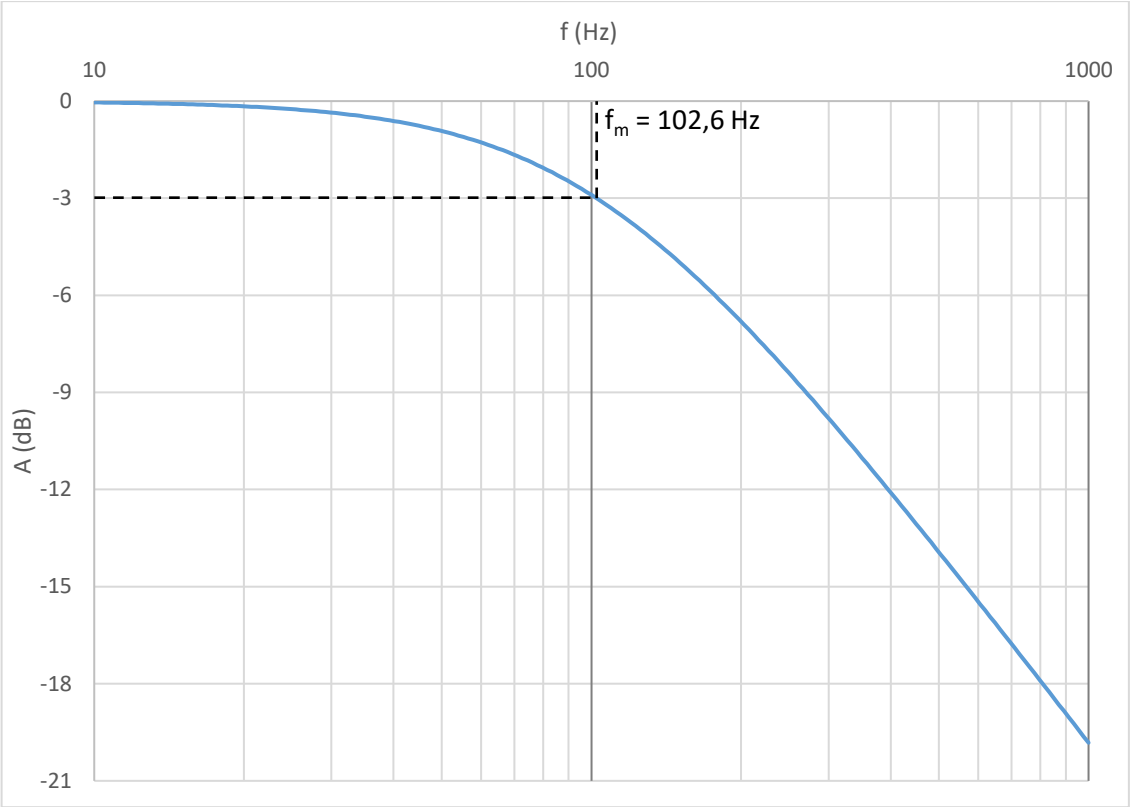


Obr. 3.5: Zapojení sondy napájené napětím 3,3 V

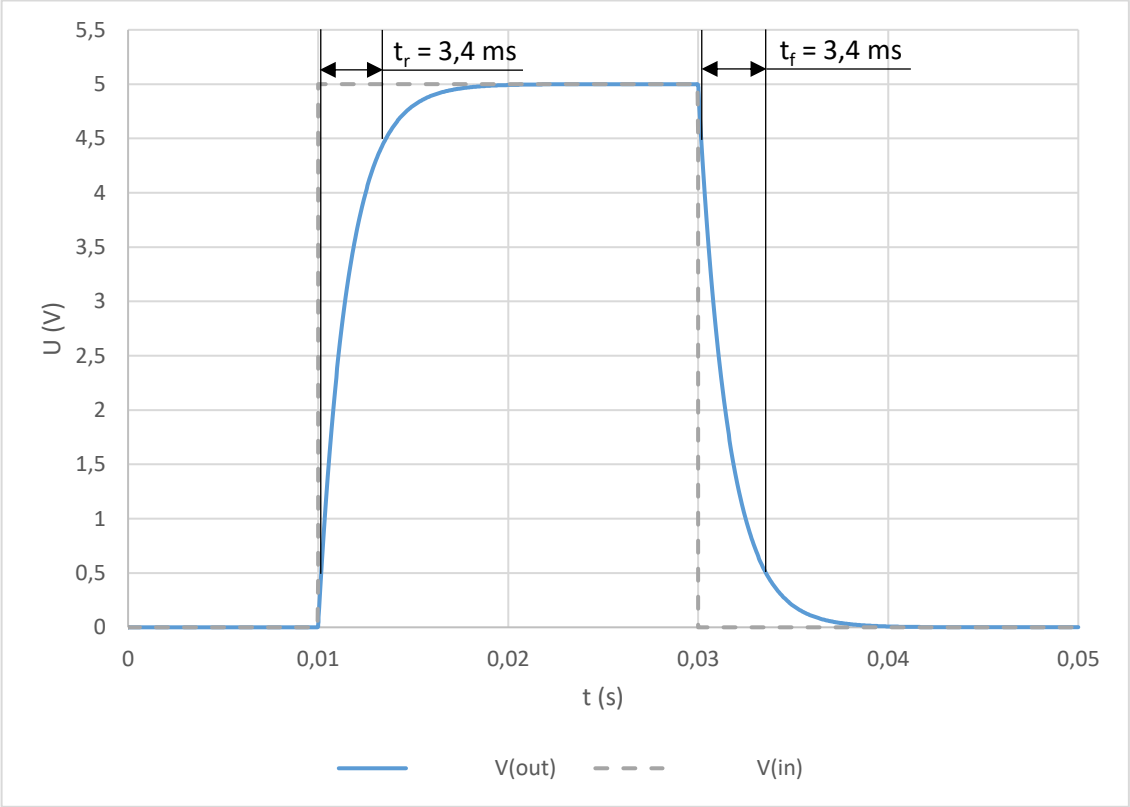
Výstup senzoru je okamžitě filtrován pasivní dolní propustí, jak doporučuje i katalogový list [5]. Důvodem oříznutí nepotřebného pásma co nejblíže výstupu. Zařízení musí reagovat na nesrovnalosti trvající více než 100 ms [1] [2]. Musí proto být schopno registrovat rychlejší změny. Z této úvahy byl zvolen i mezní kmitočet filtru, totiž f_m = 100 Hz. Při volbě C = 330 nF (*C44*) pak ze vztahu (3.11) získáváme hodnotu rezistoru R.

$$R = \frac{1}{2\pi \cdot C \cdot f_m} = \frac{1}{2\pi \cdot 330 \cdot 10^{-9} \cdot 100} = 4{,}823\ \text{k}\Omega \tag{3.11}$$

Nejbližší hodnota z řady rezistorů je 4,7 kΩ (*R19*). Správný návrh byl ověřen simulací v programu PSpice. Na obrázku (Obr. 3.6) je viditelná amplitudová charakteristika a na druhém obrázku (Obr. 3.7) tranzientní analýza s časy doběhu a náběhu.



Obr. 3.6: Amplitudová charakteristika pasivního filtru



Obr. 3.7: Tranzientní analýza pasivního filtru

Zvolením rezistoru z řady vznikl mírný posun mezního kmitočtu ze 100 na 102,6 Hz. To je naprosto zanedbatelné. Čas náběhu a doběhu (10 až 90 %) se dle očekávání rovná a je 3,4 ms. To je dostatečně rychlé.

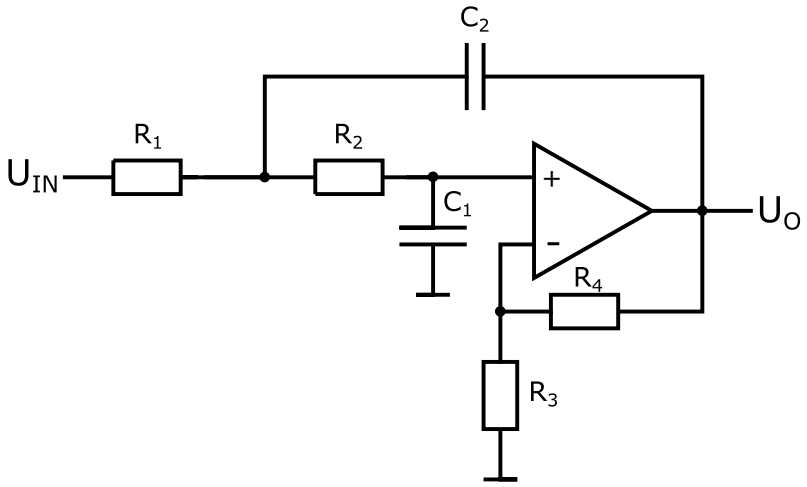
Dále v cestě všem signálům stojí DIP switch, který umožňuje rozpojovat cesty k senzoru, pro možnost případného ověření bezpečnostních mechanismů, které můžou vyžadovat inspektoři při předzávodní prohlídce vozidla. Kvůli rozpojení signálové cesty je zde zařazen pull-down rezistor (*R22*). Jeho velikost byla zvolena s ohledem na co nejmenší zatížení senzoru při normálním provozu.

3.2.2 Aktivní filtrace měřených signálů

Následuje aktivní dolní propust. Ta má za úkol účinně odstranit veškeré nepotřebné pásmo signálů. Jedná se o propust druhého řádu typu Sallen-Key (Obr. 3.8). Obecně lze přenos filtru druhého řádu *F* vyjádřit následovně:

$$F(p) = \frac{A_0}{1 + a_1 p + b_1 p^2}, \tag{3.12} [9]$$

kde A_0 je zesílení, a_1 a b_1 jsou koeficienty zvolené aproximace [9].



Obr. 3.8: Aktivní dolní propust typu Sallen-Key

Pro zjednodušení bylo zvoleno speciální zapojení Sallen-Key, kdy se rezistory (R_1, R_2) a kondenzátory (C_1, C_2) rovnají [9]. Tím se zjednoduší výběr součástek a také přenos (3.13). Ten je pak u takového filtru následovný:

$$F(p) = \frac{A_0}{1 + \omega_m RC(3 - A_0)p + (\omega_m RC)^2 p^2} \tag{3.13} [9]$$

kde ω_m je mezní úhlový kmitočet, *R* hodnota rezistorů (R_1, R_2) a *C* hodnota kondenzátorů (C_1, C_2). V porovnání se vztahem (3.12) jsou zřejmé následující rovnice [9].

$$a_1 = \omega_m RC(3 - A_0) \quad (3.14)\ [9]$$

$$b_1 = (\omega_m RC)^2 \quad (3.15)\ [9]$$

Jelikož je u této aplikace nežádoucí překmit v časové odezvě na jednotkový skok, je vhodné zvolit Besselovu aproximaci, která se tímto vyznačuje [10]. Při tomto zapojení filtru, se zesílením A_0, resp. poměrem rezistorů ve zpětné vazbě, volí i použitá aproximace [9] viz vztah (3.16).

$$A_0 = 3 - \frac{a_1}{\sqrt{b_1}} = 1 + \frac{R_4}{R_3} \quad (3.16)\ [9]$$

Po dosazení koeficientů Besselovy aproximace z tabulky v [9] (a_1 = 1,3617 a b_1 = 0,618) do (3.16) by vyšel poměr rezistorů 0,268 resp. zesílení A_0 = 1,268 viz (3.17).

$$A_0 = 3 - \frac{1{,}3617}{\sqrt{0{,}618}} = 1{,}268 \implies \frac{R_4}{R_3} = 1{,}268 - 1 = 0{,}268 \quad (3.17)$$

Vidíme, že poměr rezistorů je malý. Jelikož se obejdeme bez zesílení filtru, je možné zvolit poměr rezistorů nula, což v praxi znamená rezistory R_3 i R_4 vynechat. Tím je zvoleno A_0 = 1, a zároveň se tím ušetří součástky [9]. Samozřejmě se tím změní koeficienty aproximace, ale není se třeba obávat žádných překmitů či nestability, jelikož jsou pak póly ryze reálné a nestabilita nastává při kořenech ryze imaginárních, tj. při zesílení $A_0 \geq 3$ [10].

Jakost filtru Q se dá spočítat vztahem (3.18) níže [9].

$$Q = \frac{1}{3 - A_0} = \frac{1}{3 - 1} = \frac{1}{2} \quad (3.18)\ [9]$$

Při znalosti vztahu (3.19) lze dosazením výsledku z (3.18) snadno zjistit poměr koeficientů a_1 a b_1 (3.20).

$$Q = \frac{\sqrt{b_1}}{a_1} \quad (3.19)\ [9]$$

$$\frac{\sqrt{b_1}}{a_1} = \frac{1}{2} \quad (3.20)$$

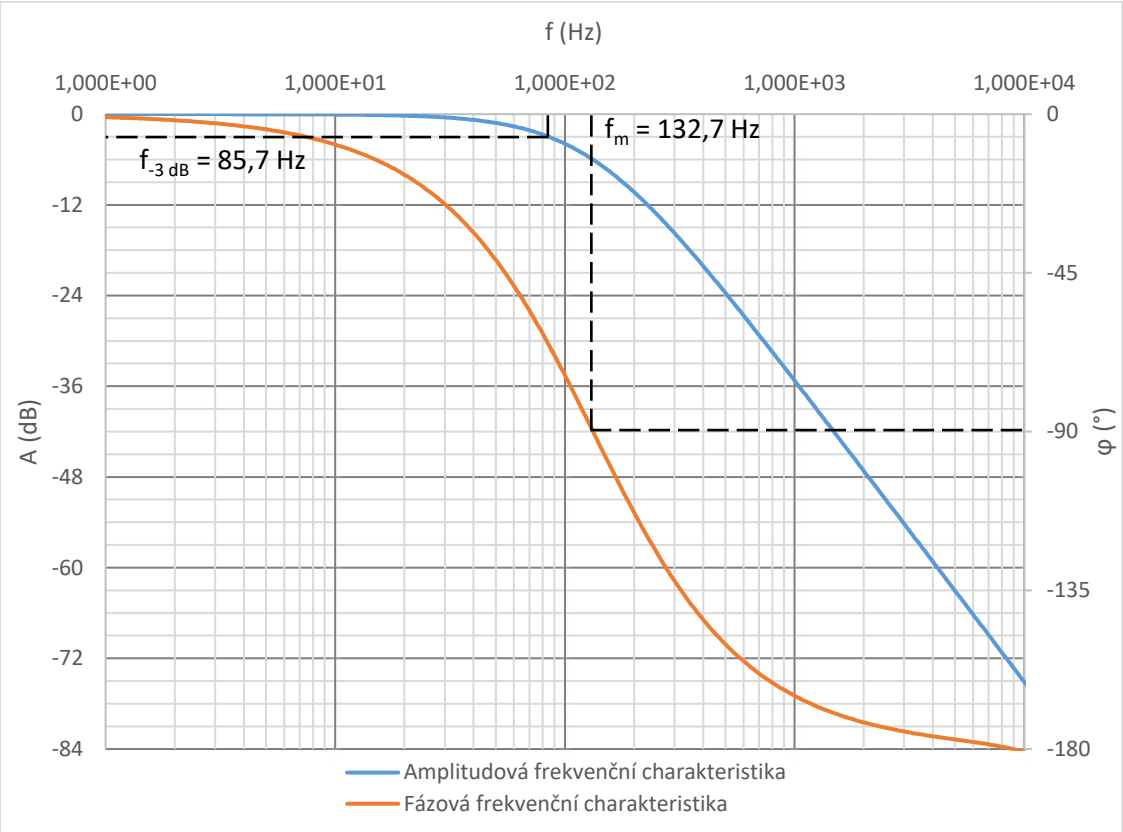
Pro jednoduchost můžeme b_1 zvolit 1. Z poměru (3.20) pak vychází koeficient a_1 = 2. Z rovnice (3.15) se dá vyjádřit vztah (3.21) pro hodnotu rezistorů:

$$R = \frac{\sqrt{b_1}}{2\pi f_m C}, \quad (3.21)\ [9]$$

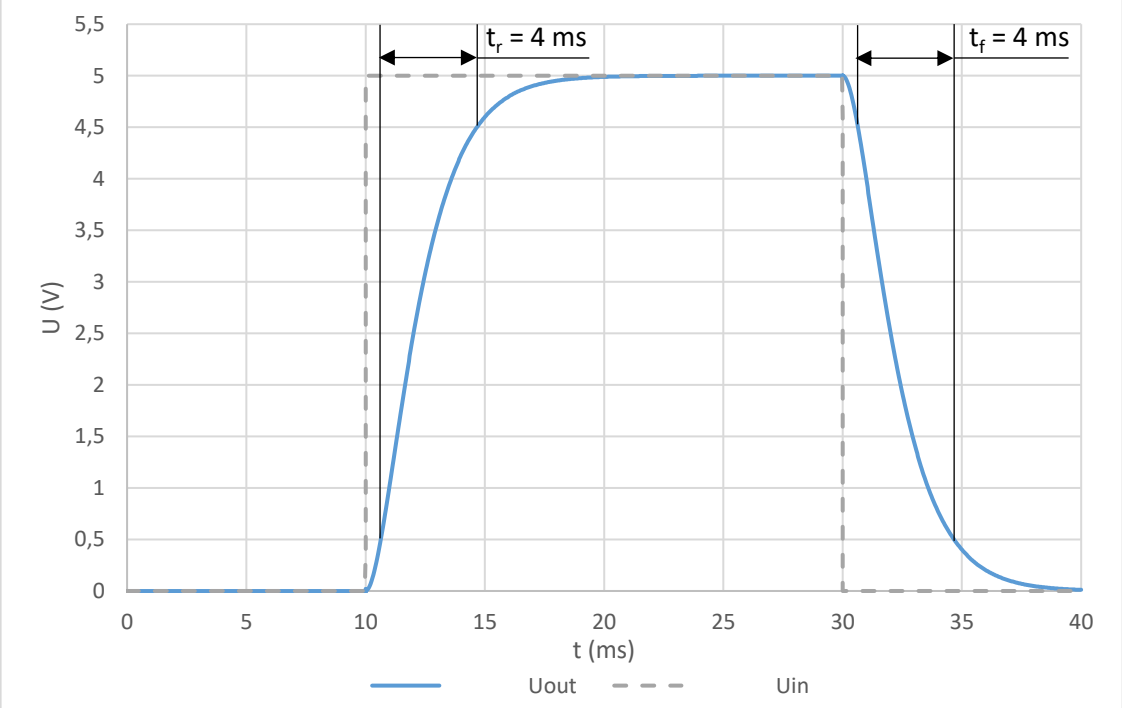
kde f_m je mezní kmitočet filtru, v tomto případě definovaný pro polovinu maximální fáze, tj. pro φ = 90°, nikoli pro pokles o 3 dB, jak bývá jinde zvykem [10]. Proto volíme kmitočet o něco vyšší, a to f_m = 130 Hz. Volbou hodnoty kondenzátorů C = 100 nF a dosazením do (3.21) dostáváme konečně hodnotu rezistorů $R = R_1 = R_2$ = 12 kΩ (3.22).

$$R = \frac{\sqrt{1}}{2\pi \cdot 130 \cdot 100 \cdot 10^{-9}} \doteq 12\ \text{k}\Omega \tag{3.22}$$

Návrh filtru byl ověřen v programu PSpice. Výsledky simulace znázorňují obrázky dále.

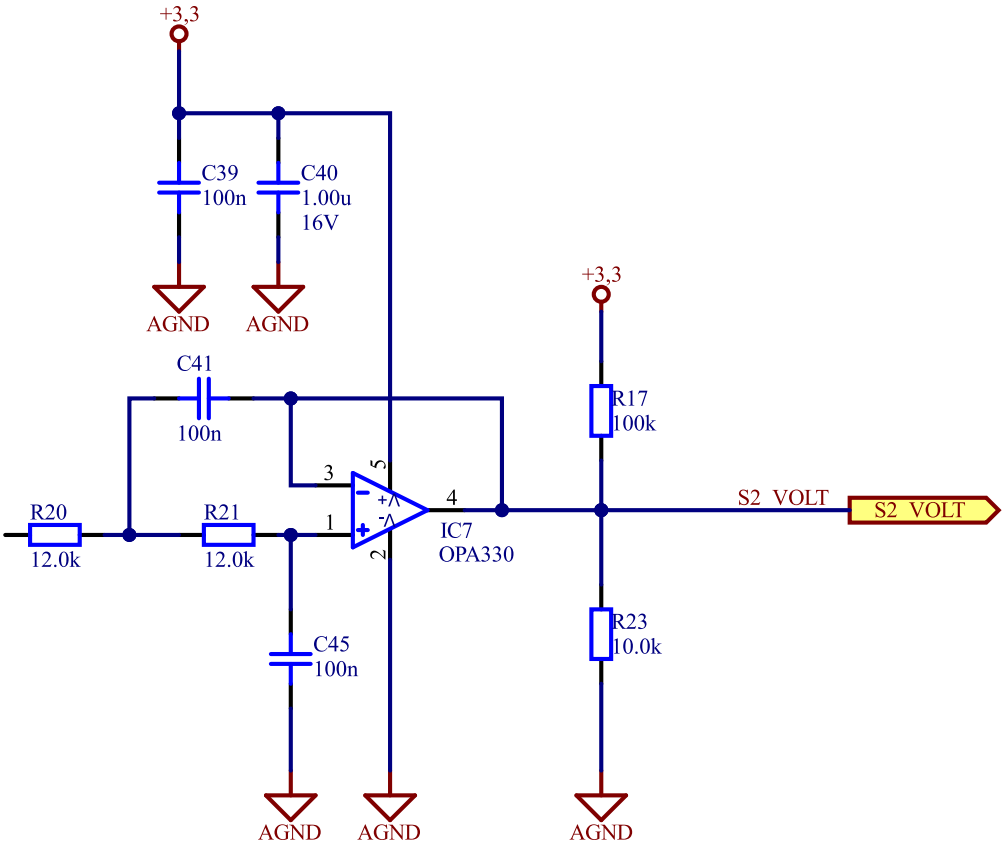


Obr. 3.9: Frekvenční charakteristiky aktivní dolní propusti



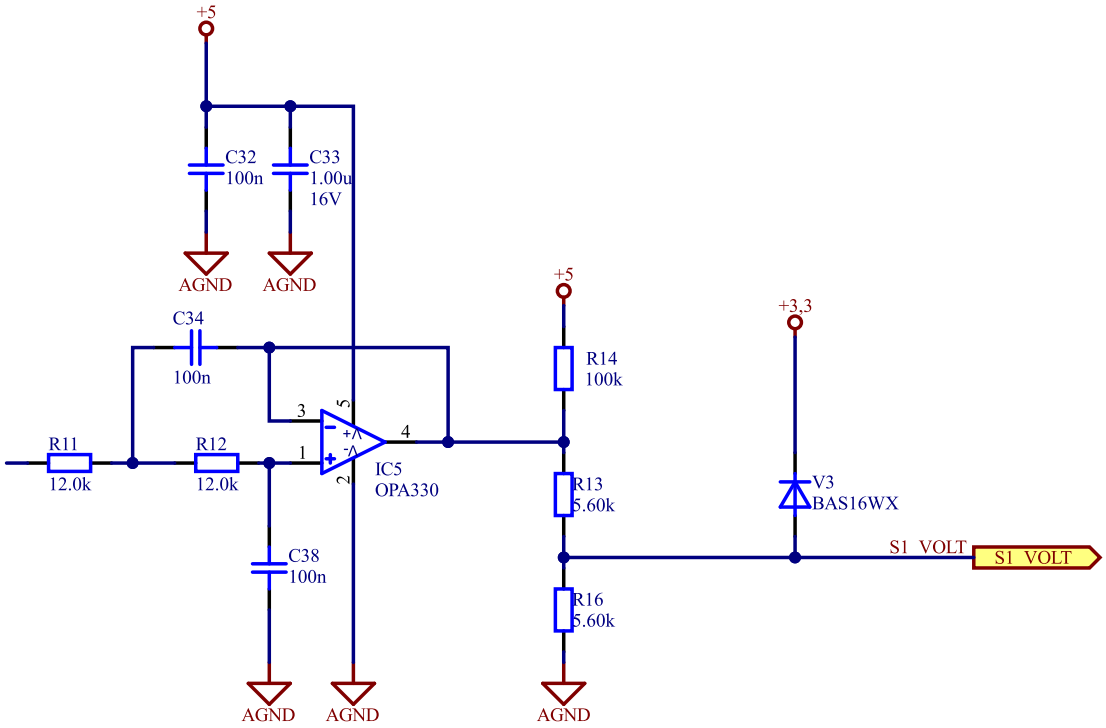
Obr. 3.10: Odezva aktivní dolní propusti na skok

Výsledný filtr vypadá následovně (Obr. 3.11). Kondenzátory (*C39, C40*) slouží k blokování napájecí větve. Napěťový dělič (*R17-R23*) definuje bezpečnou úroveň na vstupu ADC (za portem *S2_VOLT*) v případě poruchy např. analogové země, či výstupu zesilovače.



Obr. 3.11: Aktivní dolní propust pro senzor napájený napětím 3,3 V

Zapojení dolní propusti je shodné pro oba senzory, ovšem existují drobné rozdíly, a to díky rozdílnému napájecímu napětí. U senzoru napájeného z napětí 5 V je i jemu příslušející aktivní dolní propust napájena z této napěťové úrovně, viz obrázek níže (Obr. 3.12). Dále je zde navíc napěťový dělič (*R13-R16*), navržený tak, aby za normálního provozu na vstup analogově-digitálního převodníku nepřišla vyšší úroveň než 3,3 V (dělící poměr ½ - max. 2,5 V). Kdyby přeci jen došlo k překročení napěťové úrovně na vstupu převodníku (např. poruchou země), je zde ochranná omezovací dioda (*V3*), která signál ořízne. Dělič navíc definuje bezpečnou úroveň podobně jako u předchozího senzoru s pomocí pull-up rezistoru (*R14*), který má na dělící poměr děliče jen zanedbatelný vliv.



Obr. 3.12: Aktivní dolní propust pro senzor napájený napětím 5 V

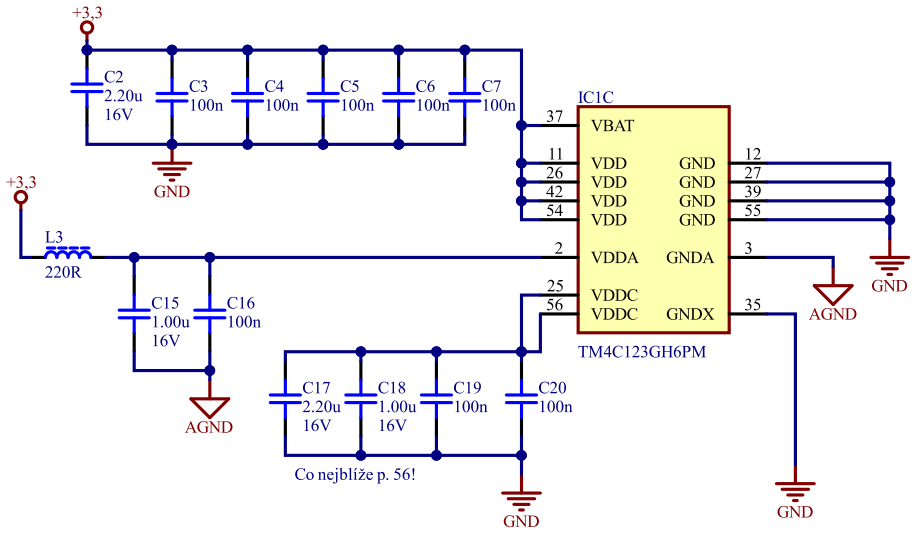
3.3 Číslicová část

Veškeré vyhodnocení signálů ze senzorů se odehrává v této části. Analogový signál, řádně vyfiltrovaný, jak bylo popsáno výše, je přiveden přes anti-aliasingové filtry na vstupy A/D převodníku mikrokontroléru. Byl zvolený mikrokontroler od firmy Texas Instruments z modelové řady Tiva, konkrétně TM4C123GH6PM. Dále se do číslicové části dá zařadit budič sběrnice CAN.

3.3.1 Mikrokontroler a jeho pomocné obvody

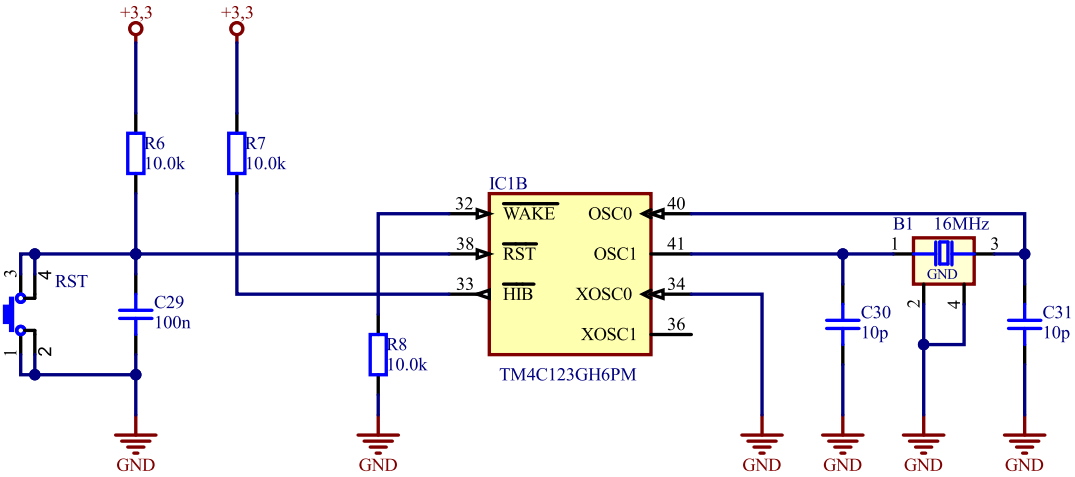
Mikroprocesor TM4C123GH6PM disponuje jádrem ARM Cortex M4F na pracovní frekvenci až 80 MHz. K dispozici má 256 KB flash paměti, 32 KB SRAM a 2 KB EEPROM. [11] Mikroprocesor dále obsahuje spoustu periférií. Pro funkci tohoto zařízení jsou stěžejní dvě periférie, a to kontrolér sběrnice CAN a 12bitové analogově-digitální převodníky. Vhod také přijde interní časovač proti zaseknutí (tzv. watchdog timer).

Napájení mikrokontroléru je zajištěno z 3,3V větve. Země kontroléru a obecně digitální části je od analogové oddělena. Je tomu tak kvůli omezení galvanické vazby číslicových a analogových součástek. Napájení procesoru je na obrázku (Obr. 3.13). Za povšimnutí stojí připojení analogové země a filtrace vstupu napájení analogové reference A/D převodníku (*L3, C15* a *C16*).



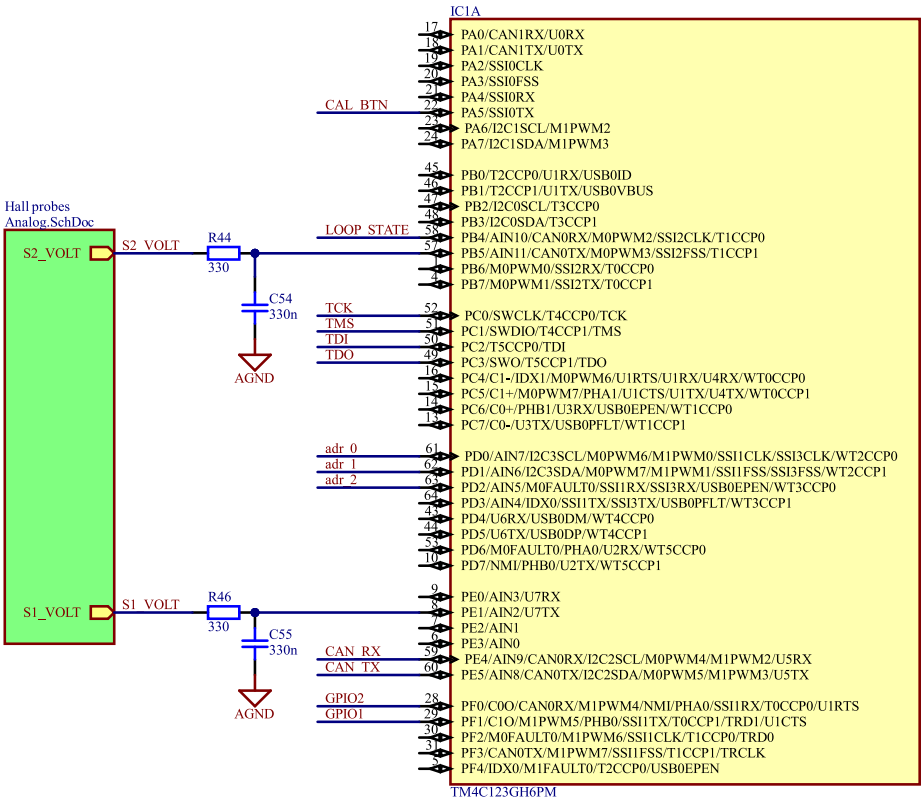
Obr. 3.13: Napájení mikrokontroléru Tiva

Na dalším schématu (Obr. 3.14) je viditelné zapojení krystalového oscilátoru 16 MHz a jeho zatěžovacích kondenzátorů (*C30* a *C31*). Funkce hibernace nebude využívána, a proto je natvrdo vyřazena přes rezistory (*R7* a *R8*). Taktéž vstup úsporného oscilátoru (*XOSC0*) je nepotřebný, a v případě nevyužití má být přizemněný [11]. Poslední věcí na obrázku (Obr. 3.14) je RC obvod (*R6* a *C29*) s resetovacím tlačítkem (*RST*). Tento článek má za úkol uvést mikroprocesor do reset stavu na určitou dobu po zapnutí napájení zařízení, než dojde k ustálení napájecích úrovní. Pro případ potřeby resetovat obvod ručně je zde resetovací tlačítko (*RST*), které zajistí vybití kondenzátoru (*C29*) a vyvolání reset stavu.



Obr. 3.14: Pomocné obvody mikroprocesoru Tiva

Na obrázku (Obr. 3.15) je vykresleno zapojení mikrokontroléru. Jednotlivé připojené signály budou rozebrány v dalším textu.



Obr. 3.15: Zapojení mikroprocesoru Tiva

3.3.2 Anti-aliasingové filtry

Při používání A/D převodníku se musí dodržovat tzv. Shannon-Kotělnikovův vzorkovací teorém. Dle tohoto teorému je nutné, aby se v digitalizovaném signálu nevyskytovaly žádné frekvenční složky vyšší než polovina vzorkovací frekvence převodníku. Při nedodržení by došlo k překryvu frekvenčních pásem, a tudíž k znehodnocení digitalizovaných dat. Tento efekt se nazývá aliasing. Proto musí vstupům A/D převodníků předcházet tzv. anti-aliasing filtr (*R44-C54* a *R46-C55*). Tento filtr typu dolní propust tedy omezuje šířku pásma digitalizovaného signálu před vzorkovačem. [10]

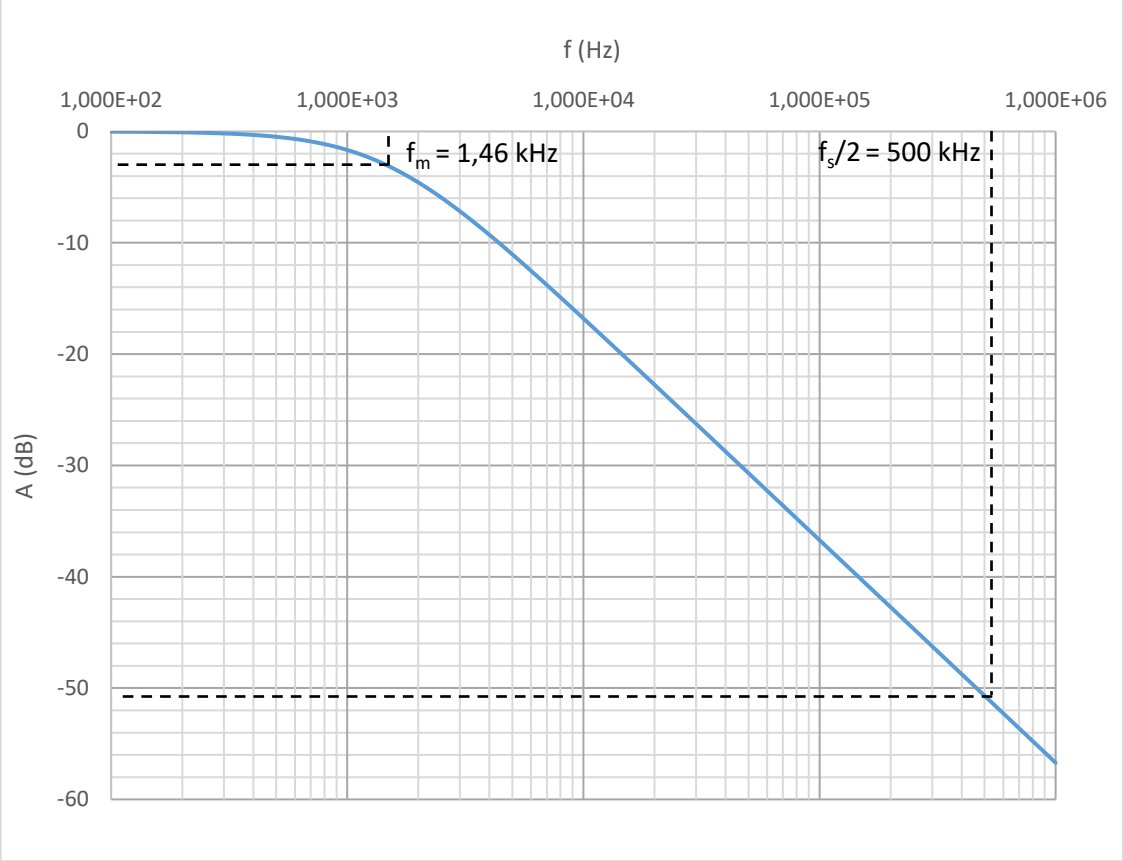
Signály z výše popsaných analogových obvodů zpracovávajících napětí ze senzorů jsou přivedeny na piny *8* a *57* (Obr. 3.15). Na těchto pinech se nachází kanály analogově-digitálních převodníků [11]. Tyto signály využívají frekvenční šířku pásma přibližně 100 Hz. Analogově-digitální převodníky v použitém mikrokontroléru zvládají rychlosti vzorkování až 1 MS/s [11]. Lze tedy signály ze sond vzorkovat přibližně 10 000krát větší frekvencí. Ačkoli by teoreticky stačilo vzorkovat frekvencí lehce přes 200 Hz, dá se těchto schopností vzorkovače využít ke snížení nároků na strmost útlumové charakteristiky anti-aliasingového filtru. Takové metodě se říká převzorkování [10].

Díky převzorkování lze použít jen obyčejný RC článek se sklonem útlumové charakteristiky 20 dB/dek. Při jeho použití dosáhneme útlumu na polovině vzorkovacího

kmitočtu přibližně 50 dB, jak dokazuje výstup ze simulace v programu PSpice na obrázku níže (Obr. 3.16). Filtr byl navržen dle vztahu (3.23). Při návrhu anti-aliasingových filtrů je vhodné dle [15] zvolit mezní kmitočet filtru přibližně 10krát vyšší, než je maximální kmitočet vyskytující se v digitalizovaném signálu, tzn. v tomto případě f_m = 1 kHz. Hodnota rezistoru byla zvolena jako R = 330 Ω.

$$C = \frac{1}{2\pi \cdot R \cdot f_m} = \frac{1}{2\pi \cdot 330 \cdot 1 \cdot 10^3} = 482{,}3 \text{ nF} \tag{3.23}$$

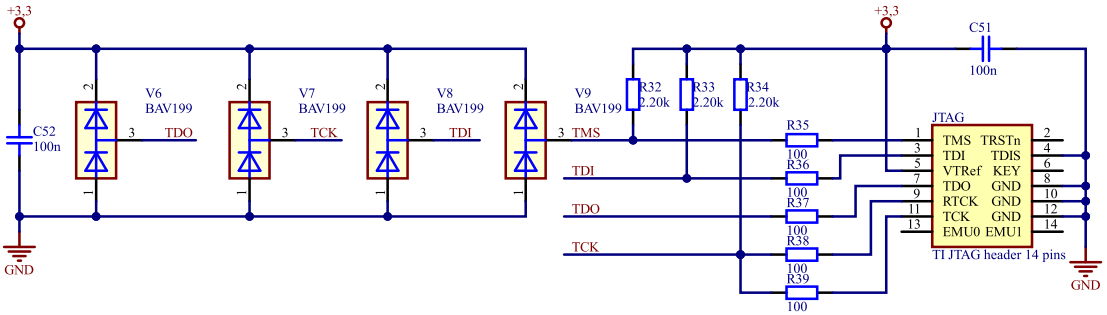
Hodnota kondenzátoru byla nakonec zvolena 330 nF, ačkoli se nejedná o úplně nejbližší hodnotu v řadě. Je tomu tak z důvodu, že tato kapacita je již ve schématu použita a v této aplikaci taková odchylka není kritická, přičemž se ušetří na množstevní slevě a práci při osazování. Mezní kmitočet se posune na f_m = 1,46 kHz, jak je patrné i z obrázku níže (Obr. 3.16).



Obr. 3.16: Simulace anti-aliasing filtru v programu PSpice

3.3.3 Rozhraní JTAG

Pro možnosti programování a dodatečného debuggování je na desce snímače umístěn i standardní 14 pinový konektor rozhraní JTAG (Obr. 3.17). Toto rozhraní má tři vstupy a jeden výstup. Vstupy jsou značeny *TCK* (Clock), *TDI* (Data Input), *TMS* (Test Mode Select), a jsou ošetřeny pull-up rezistory. Výstupní signál je značen *TDO* (Data Output) [11].

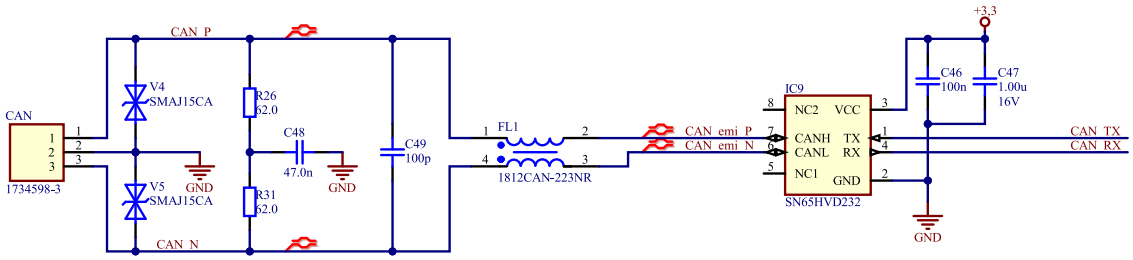


Obr. 3.17: Zapojení konektoru a ochran rozhraní JTAG

Na obrázku výše (Obr. 3.17) si lze všimnout čtveřice ochranných diod (*V6*, *V7*, *V8* a *V9*), chránících vstupy mikrokontroléru proti přepětí. Rezistory (*R35* až *R39*) slouží k omezení proudu při působení ochranných diod. Veškeré zmíněné signály jsou pak přivedeny na mikrokontroler – na obrázku (Obr. 3.15) piny *49* až *52*.

3.3.4 Obvody sběrnice CAN

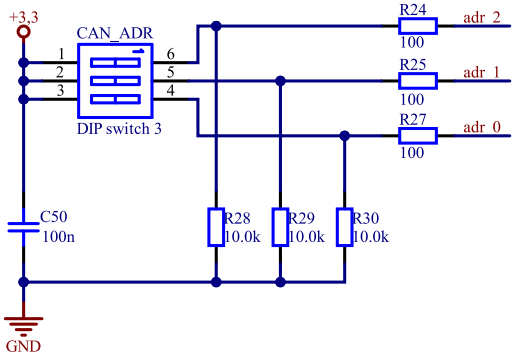
Mikrokontroler disponuje dvěma rozhraními sběrnice CAN [11]. V návrhu je počítáno s využitím CAN kontroléru *CAN0* na pinech mikroprocesoru *59* a *60* (Obr. 3.15). Z těchto pinů jsou signály (*CAN_RX* a *CAN_TX*) vedeny na budič sběrnice CAN, totiž na obvod SN65HVD232 od firmy Texas Instruments. Ten tvoří interface mezi diferenciálními linkami sběrnice a signály z CAN kontroléru v mikroprocesoru na 3,3V úrovni [12]. Obvod navíc obsahuje ochrany proti ESD, nebo například zajišťuje bezpečné stavy na RX výstupu při selhání sběrnice či jejím rozpojení [12].



Obr. 3.18: Zapojení sběrnice CAN

Popsané zapojení je na obrázku (Obr. 3.18). Dále se zde nacházejí další obvody pro ochranu proti ESD – transily (V4 a V5), nebo pro terminaci sběrnice (R26 a R31). Pro odstranění nesymetrických rušivých proudů je v diferenciální cestě umístěna proudově kompenzovaná tlumivka (FL1). Jedná se o tlumivku speciálně vyráběnou pro datové automobilové aplikace [13].

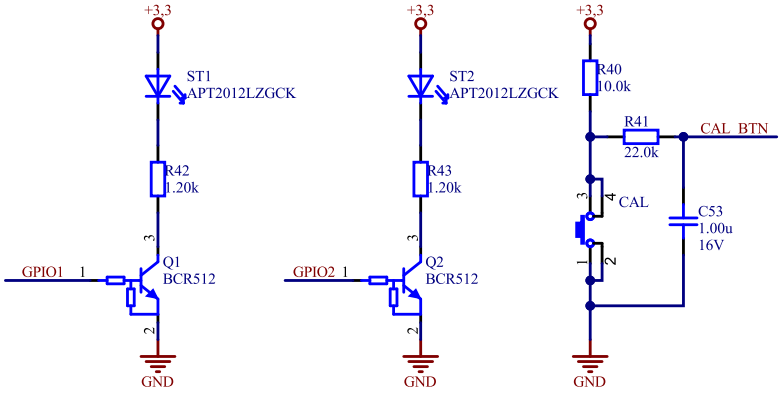
Zařízení bude na sběrnici CAN vystupovat pod určitou adresou. Aby bylo možné operativně měnit alespoň část adresy, jsou na piny (61 až 63) mikroprocesoru (Obr. 3.15) zapojeny kontakty z DIP switche (CAN_ADR) na obrázku níže (Obr. 3.19).



Obr. 3.19: DIP switch pro volbu adresy na sběrnici CAN

3.3.5 Doplňkové obvody

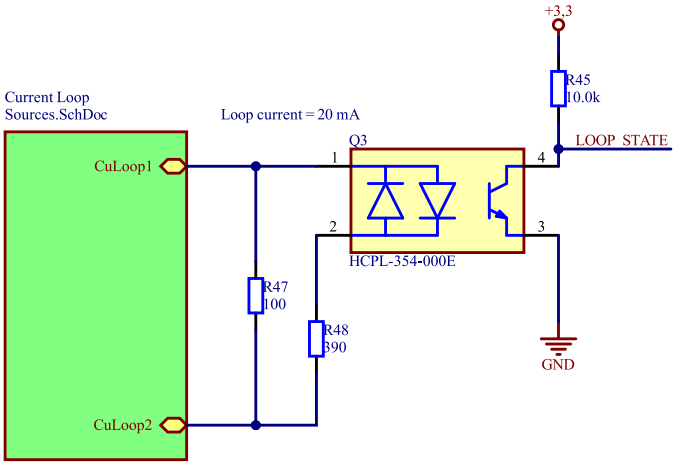
Ve schématu se dále vyskytují doplňkové obvody sloužící pro signalizaci a obsluhu snímače. Jsou tím myšleny například obvody stavových svítivých diod (*ST1* a *ST2*) na obrázku (Obr. 3.20).



Obr. 3.20: Stavové LED a kalibrační tlačítko

LED jsou ovládány přes tranzistory s integrovanými rezistory (*Q1* a *Q2*). Samotné tranzistory pak ovládá mikrokontroler pomocí signálů (*GPIO1* a *GPIO2*) z digitálních výstupů na pinech *28* a *29* (Obr. 3.15). Dále lze na obrázku výše (Obr. 3.20) spatřit tlačítko. Toto tlačítko má umožnit přechod zařízení do kalibračního módu (viz kapitola 5.2.3). Tlačítko je opatřeno pull-up rezistorem (*R40*) a RC článkem (*R41-C53*) pro omezení zákmitů. Tento výstup tlačítka je pak přiveden na pin *22* mikrokontroléru (Obr. 3.15).

Posledním nepopsaným obvodem je obvod pro detekci rozpojení proudové kontrolní smyčky (tzv. Interlock). Ta již byla zmíněna u popisu vstupních obvodů. Smyčka prochází veškerými důležitými komponentami vozidla. Pokud některá komponenta není připojena, smyčkou neprotéká žádný proud, což je vyhodnoceno jako chyba. Aby i samotný snímač pedálu mohl detekovat rozpojenou smyčku, je do schématu zahrnut proudový dělič (*R47-R48*) s optočlenem (*Q3*) na obrázku níže (Obr. 3.21). Optočlen byl zvolen obousměrný, aby nezáleželo na polaritě proudu v připojené proudové smyčce.



Obr. 3.21: Optočlen s proudovým děličem pro detekci rozpojení proudové smyčky

Při návrhu se počítalo s proudem uzavřenou smyčkou I = 20 mA. Proud optočlenem byl zvolen s ohledem na jeho mezní parametry v katalogovém listě [14] na I_{OPT} = 4 mA.

Tranzistor optočlenu pak přivádí na pin mikrokontroléru číslo *58* (Obr. 3.15) v případě uzavřené smyčky úroveň logické nuly. V případě rozpojené smyčky se tranzistor uzavře a přes pull-up rezistor (*R45*) je přivedena na pin mikrokontroléru úroveň logické jedničky.

4 Návrh desky plošných spojů

Velikost i tvar desky plošných spojů vychází z modelu popisovaného v kapitole 2.3.1. Umístění senzorů (Hallovo sond) je na desce pevně dané. Dále i samotná konstrukce krabičky zabraňuje umístění ovládacích či signalizačních prvků (tlačítka, spínače, LED) do středu desky, a to vzhledem k tunelu s magnetem, který by překážel.

Schéma obsahuje, resp. deska nese, celkem 147 součástek a její rozměry jsou přibližně 64 mm na délku a 44 mm na šířku. Z těchto důvodů byla již od začátku deska projektována jako čtyřvrstvá v konstrukční třídě 6. Prostřední dvě vrstvy slouží pro napájení, horní a spodní vrstva pak pro vedení signálových cest.

Návrh desky, stejně jako i schématu, byl proveden v programu Altium Designer. Motivy jednotlivých vrstev desky jsou přiloženy v přílohách.

4.1 Rozmisťování součástek

V první řadě bylo nutné správně umístit senzory. Jak již bylo zmíněno, jejich pozice je pevně dána přídržnými šachtami na straně tunelu s magnetem. Pro snadné ověření správného umístění se proto hodila funkce programu Altium Designer pro import DXF výkresu z externího návrhového systému. Do návrhu desky se tak snadno vložil výkres krabičky a definoval se tak i samotný tvar desky.

Následovalo pak přibližné rozmístění vysokých součástek a součástek, které musí být přístupné. Ty se umístily mimo oblast pod tunelem. Velké a těžké součástky (typicky indukčnosti a elektrolytické kondenzátory) byly umisťovány hlavně z vrchní strany, na spodní stranu desky byl umístěn procesor a jiné nízké a lehké součástky.

Dále bylo rozmisťování již standardní. První přišly na řadu blokovací kondenzátory a vstupní napájecí filtr. Následovaly méně kritické součástky jako rezistory atp.

4.2 Tvorba cest

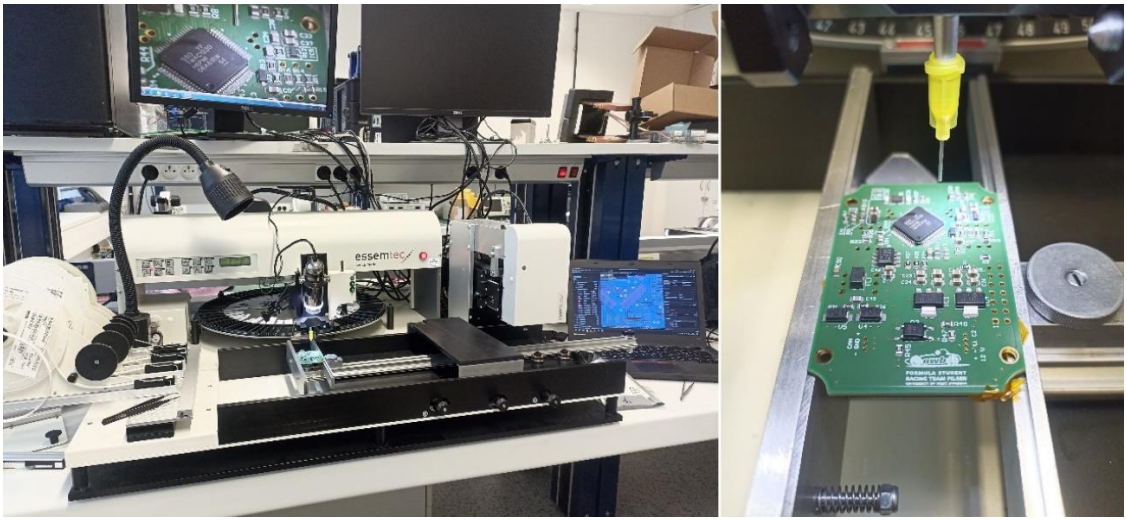
Vnitřní vrstvy jsou rozlity pomocí funkce „plane“. Signálové vrstvy (horní a dolní) mají rozlitý zemní polygon pro dosažení co nejnižší impedance země. Tyto zemní vrstvy jsou z téhož důvodu na mnoha místech propojeny pomocí prokovů. Analogová země je od digitální oddělena a s digitální je spojena pouze v jednom místě na horní vrstvě pomocí nulového rezistoru (*R3*).

Vnitřní zemní vrstva je taktéž rozdělena na oblast analogové a digitální země. Celá oblast analogové země kopíruje tvar oblasti, kde jsou umístěny analogové součástky.

Vnitřní vrstva napájení je rozdělena tvarově obdobně jako vrstva země. Na vrstvě jsou tři různá napájení, konkrétně 3,3 V pro digitální a 3,3 a 5 V pro analogové komponenty. Mezi jednotlivými plochami vnitřních vrstev se uplatňuje určitá kapacita, která je výhodná jako další přídavné blokování napájení.

4.3 Výroba, osazení a oživení

Deska plošného spoje byla vyrobena v Německu výrobcem Multi-cb. Vyrobeny byly i šablony pro nanášení pájecí pasty. Osazení proběhlo na ruční osazovačce (Obr. 4.1) v laboratoři elektroniky v RICE.



Obr. 4.1: Ruční osazovačka a pohled na DPS při osazování

Osazená deska z jedné strany byla protažena pecí. Těžké součástky na již zapájené straně byly zajištěny kaptonovou páskou a osazena druhá strana. Ta byla zapájena opět protažením pecí. Kaptonová páska zajistila, že součástky na druhé straně neupadly. Následně byly ručně zapájeny zbylé THT součástky.

Krabička byla vytisknuta na 3D sintrové tiskárně, dle modelu v kapitole 2.3.1. Poté byly do krabičky vyřezány závity a celý výrobek složen dohromady. Výsledný snímač je na obrázku níže (Obr. 4.2).

Obr. 4.2: Hotový snímač polohy

5 Program

Mikrokontroler byl naprogramován pomocí programu Code Composer Studio v programovacím jazyce C. Při programování byly využity knihovny TivaWare speciálně dodávané firmou Texas Instruments pro své řady mikrokontrolerů Tiva. Celý program je uvedený v příloze (Příloha VIII).

Na následujících řádcích bude následovat popis určitých segmentů programu s popisem chování zařízení v určitých situacích pro vysvětlení principu funkce.

5.1 Inicializace

V hlavní funkci „`main(void)`“ je nejprve inicializována floating-point jednotka. Následuje nastavení pracovního taktu na frekvenci 80 MHz. Této frekvence se dosáhne při použitém 16MHz krystalu pomocí fázového závěsu a děličce 2,5. Dále je aktivován tzv. Watchdog časovač a periferie EEPROM. Inicializace GPIO a ostatních periferií je řešena pomocí funkce „`PortFunctionInit(void)`“ vygenerované programem PinMux. Ten umožňuje velice snadnou volbu funkcí jednotlivých pinů pomocí grafického uživatelského rozhraní.

```
int main(void)
{
    /* ----- Initialization part ----- */
    FPULazyStackingEnable();
    FPUEnable();
    // Set the clocking to run from the PLL at 80MHz
    SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
    // Setup the watchdog timer
    InitializeWatchdog();
    // Enable the EEPROM
    SysCtlPeripheralEnable(SYSCTL_PERIPH_EEPROM0);
    // Set pins to their functions
    PortFunctionInit();
    // Initialize SysTick
    InitializeSysTick();
    // Initialize ADC for reading signals
    InitializeADCs();
    // Initialize the CAN Bus for sending information to the control unit
    InitializeCAN();
    // Clear the interrupt status flag.  This is done to make sure the
    // interrupt flag is cleared before we sample.
    ADCIntClear(ADC0_BASE, ADC_SEQ);
    ADCIntClear(ADC1_BASE, ADC_SEQ);
    // Trigger first conversion
    ADCProcessorTrigger(ADC0_BASE, ADC_SEQ | ADC_TRIGGER_WAIT);
    ADCProcessorTrigger(ADC1_BASE, ADC_SEQ | ADC_TRIGGER_SIGNAL);
    /* ----- End of Initialization part ----- */

…
```

Ukázka kódu 5.1: Inicializace

Dále je zavolána funkce „`InitializeSysTick(void)`“, která nastaví takt a zaregistruje obsluhu přerušení časovače SysTick. Časovač je nastaven tak, aby přetekl, a tudíž vyvolal

přerušení každou milisekundu. V obsluze přerušení dochází k inkrementaci proměnné „`uint64_t _ticks`“, která je dále v programu využívána pro potřeby časování.

```
void InitializeSysTick(void)
{
    // Setting period of SysTicks
    SysTickPeriodSet(SysCtlClockGet() / SYSTICKS_PER_SECOND);

    // Register the handler for SysTick
    SysTickIntRegister(SysTickHandler);

    // Enable SysTick interrupts
    SysTickIntEnable();

    // Enable SysTick
    SysTickEnable();
}
```

Ukázka kódu 5.2: Inicializace časovače SysTick

Funkce „`InitializeADCs(void)`“ (Ukázka kódu 5.3) následně provede nastavení analogově-digitálních převodníků. Mikrokontrolér obsahuje dva analogově digitální převodníky (*ADC0* a *ADC1*). Oběma je na začátku nastavena nejvyšší rychlost vzorkování. Dále jsou oběma nakonfigurovány sekvencery. Periferie ADC obsahuje 4 sekvencery [11], každý s jinou hloubkou FIFO paměti. Vzhledem k tomu, že chceme vzorkovat jen ze dvou kanálů a máme dva převodníky, postačí sekvencery číslo 3 s hloubkou FIFO rovno jedné. Každý převodník tedy bude vzorkovat jen jeden z kanálů. Taktéž je nastaveno spouštění ADC pomocí procesoru, nastaveno přerušení a zaregistrována jeho obsluha. Aplikace využívá hardwarového průměrování hodnot z ADC, konkrétně 8 po sobě jdoucích vzorků. Toto nastavení zajišťuje funkce „`ADCHardwareOversampleConfigure(…)`“.

```
void InitializeADCs(void)
{
    // Set full speed on ADC
    ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PIOSC | ADC_CLOCK_RATE_FULL, 1);
    // Set Sequencers on ADC
    ADCSequenceConfigure(ADC0_BASE, ADC_SEQ, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceConfigure(ADC1_BASE, ADC_SEQ, ADC_TRIGGER_PROCESSOR, 0);
    // Configure sequencers steps
    ADCSequenceStepConfigure(ADC0_BASE, ADC_SEQ, 0,
                             ADC_CTL_CH2 | ADC_CTL_IE | ADC_CTL_END); // ADC0 read S1
    ADCSequenceStepConfigure(ADC1_BASE, ADC_SEQ, 0,
                             ADC_CTL_CH11 | ADC_CTL_IE | ADC_CTL_END); // ADC2 read S2
    // Configure hardware averaging of 8 values
    ADCHardwareOversampleConfigure(ADC0_BASE, 8);
    ADCHardwareOversampleConfigure(ADC1_BASE, 8);
    // Since sample sequence is now configured, it must be enabled.
    ADCSequenceEnable(ADC0_BASE, ADC_SEQ);
    ADCSequenceEnable(ADC1_BASE, ADC_SEQ);
    // Setting the interrupts handler for ADCs
    ADCIntRegister(ADC0_BASE, ADC_SEQ, ADCIntHandler);
    ADCIntRegister(ADC1_BASE, ADC_SEQ, ADCIntHandler);
    // Enable interrupts
    ADCIntEnable(ADC0_BASE, ADC_SEQ);
    ADCIntEnable(ADC1_BASE, ADC_SEQ);
}
```

Ukázka kódu 5.3: Inicializace analogově-digitálních převodníků

V neposlední řádě je inicializován kontrolér sběrnice CAN. Funkce „`InitializeCAN(void)`“ (Ukázka kódu 5.4) nejprve uloží ukazatel na buffer „`uint32_t CANMsgBuffer`“. Ten slouží k dočasnému uložení dat zprávy před odesláním. Následuje samotná inicializace, nastavení komunikační rychlosti, zaregistrování obsluhy přerušení a zapnutí periferie (*CAN0*). Následuje příprava objektu používaného k odesílání zpráv „`tCANMsgObject CANMessage`“. Tento objekt (struktura) obsahuje kompletní informace o odesílané zprávě, tzn. adresu, DLC, ukazatel na buffer, masku a různé doplňkové flagy. Adresa je dána čtením z pinů z dip-switche „CAN_ADDR“. Jelikož se čte jen v tomto místě programu, po proběhnutí této části již nedojde ke změně adresy po přepnutí DIP-switche. Pro projevení změny je tedy nutný reset MCU. Délka dat (DLC) je nastavena na pevné 3 bajty.

```
void InitializeCAN(void)
{
    // Set the message buffer pointer
    CANMsgDataP = (uint8_t *) &CANMsgData;
    // Initialize the CAN controller
    CANInit(CAN0_BASE);
    // Set up the bit rate for the CAN bus: 500k
    CANBitRateSet(CAN0_BASE, SysCtlClockGet(), 500000);
    // Setting the interrupt handler for CAN bus controller
    CANIntRegister(CAN0_BASE, CANIntHandler);
    // Enable interrupts on the CAN peripheral
    CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR | CAN_INT_STATUS);
    // Enable the CAN for operation.
    CANEnable(CAN0_BASE);
    // Initialize the message object that will be used for sending CAN messages.
    CANMsgData = 0;
    // CAN ID - reading address from dip-switch
    CANMessage.ui32MsgID = GPIOPinRead(CAN_ADDR_BASE, CAN_ADDR_MASK);
    CANMessage.ui32MsgIDMask = 0;
    CANMessage.ui32Flags = MSG_OBJ_TX_INT_ENABLE;
    CANMessage.ui32MsgLen = 3; // DLC = 3
    CANMessage.pui8MsgData = CANMsgDataP;
}
```

Ukázka kódu 5.4: Inicializace kontroléru sběrnice CAN

Pro dokončení inicializace (Ukázka kódu 5.1) je ještě zapotřebí vymazat flagy přerušení ADC a spustit první převod. Dále následují již jen definice proměnných, a pak nekonečná smyčka „`while(1)`“.

5.2 Nekonečná smyčka

V této části programu se řeší periodicky opakované děje, což je v podstatě veškerá činnost, kterou má toto zařízení vykonávat. Z nekonečné smyčky se program dostane jen pro obsluhu přerušení, poté se vrací nazpět.

5.2.1 Čtení hodnot z ADC

V nekonečné smyčce přichází první na řadu vyčtení dat z registrů analogově-digitálních převodníků, jsou-li k dispozici (Ukázka kódu 5.5). To je indikováno flagem „`volatile bool ADCValueReady`“, který je nastaven při dokončení převodu obou ADC v obsluze přerušení „`ADCIntHandler(void)`“ (Ukázka kódu 5.6).

```
...
   // Firstly read the sensors values from ADCs
   if (ADCValueReady)
   {
        // Copy the measured data to ADCxValue variables
        ADCSequenceDataGet(ADC0_BASE, ADC_SEQ, &mainStruct.ADCValues[0]);
        ADCSequenceDataGet(ADC1_BASE, ADC_SEQ, &mainStruct.ADCValues[1]);

        // Set ADC flag down
        ADCValueReady = false;

        // Trigger next ADC conversion
        ADCProcessorTrigger(ADC0_BASE, ADC_SEQ | ADC_TRIGGER_WAIT);
        ADCProcessorTrigger(ADC1_BASE, ADC_SEQ | ADC_TRIGGER_SIGNAL);

        // Set the time of the last update of values
        ADCUpdateTicks = _ticks;
   }
...
```

Ukázka kódu 5.5: Část pro vyčítání hodnot z ADC

Po vyčtení je flag vynulován a spuštěna další konverze. Převody jsou spouštěny synchronně. Nakonec je uložen čas vyčtení hodnot v milisekundách (`_ticks`) do proměnné „`uint64_t ADCUpdateTicks`“.

```
...
void ADCIntHandler(void)
{
    if ((ADCIntStatus(ADC0_BASE, ADC_SEQ, 1) == ADC_INT_SS3)
            && (ADCIntStatus(ADC1_BASE, ADC_SEQ, 1) == ADC_INT_SS3))
    {
        // Set global flag
        ADCValueReady = 1;

        // Clear the ADC interrupt flag.
        ADCIntClear(ADC0_BASE, ADC_SEQ);
        ADCIntClear(ADC1_BASE, ADC_SEQ);
    }
}

...
```

Ukázka kódu 5.6: Obsluha přerušení ADC

5.2.2 Obsluha signalizačních LED

Dále jsou obsluhovány dvě signalizační svítivé diody (Ukázka kódu 5.7). To, jestli diody blikají, svítí či nesvítí, je řešeno pomocí pole „`uint16_t LEDtimePeriod[2]`“. V poli index 0 odpovídá první a index 1 druhé diodě. Nastavením libovolného čísla do proměnné se volí čas, po který dioda svítí a nesvítí (půlperioda). Pokud je nastavena krajní hodnota (0 nebo

0xFFFF), dioda buď svítí, nebo nikoli. Pro usnadnění obsluhy jsou nadefinována makra, namátkou např. `LED_ON`, `LED_BLINK_250ms` nebo `LED_OFF`.

```
...
// State indicators LEDs controlling cycle
uint8_t i;
for (i = 0; i < 2; i++)
{
        switch (LEDtimePeriod[i])
        {
            case LED_OFF:
                // LED should be OFF
                LEDState[i] = 0x00;
                break;
            case LED_ON:
                // LED should be ON
                LEDState[i] = 0xff;
                break;
            default:
                // If the value is in the range between 0 and 0xFFFF,
                // it should be blinking periodically
                if (_ticks >= blinkTicks[i])
                {
                    // Toggle state of LED
                    LEDState[i] = ~LEDState[i];
                    // Set the next time for toggling the LED
                    blinkTicks[i] = _ticks + LEDtimePeriod[i];
                }
        }
}
GPIOPinWrite(ST_LED1, LEDState[0]);
GPIOPinWrite(ST_LED2, LEDState[1]);
...
```

Ukázka kódu 5.7: Část kódu ovládající stavové LED

5.2.3 Stavový automat

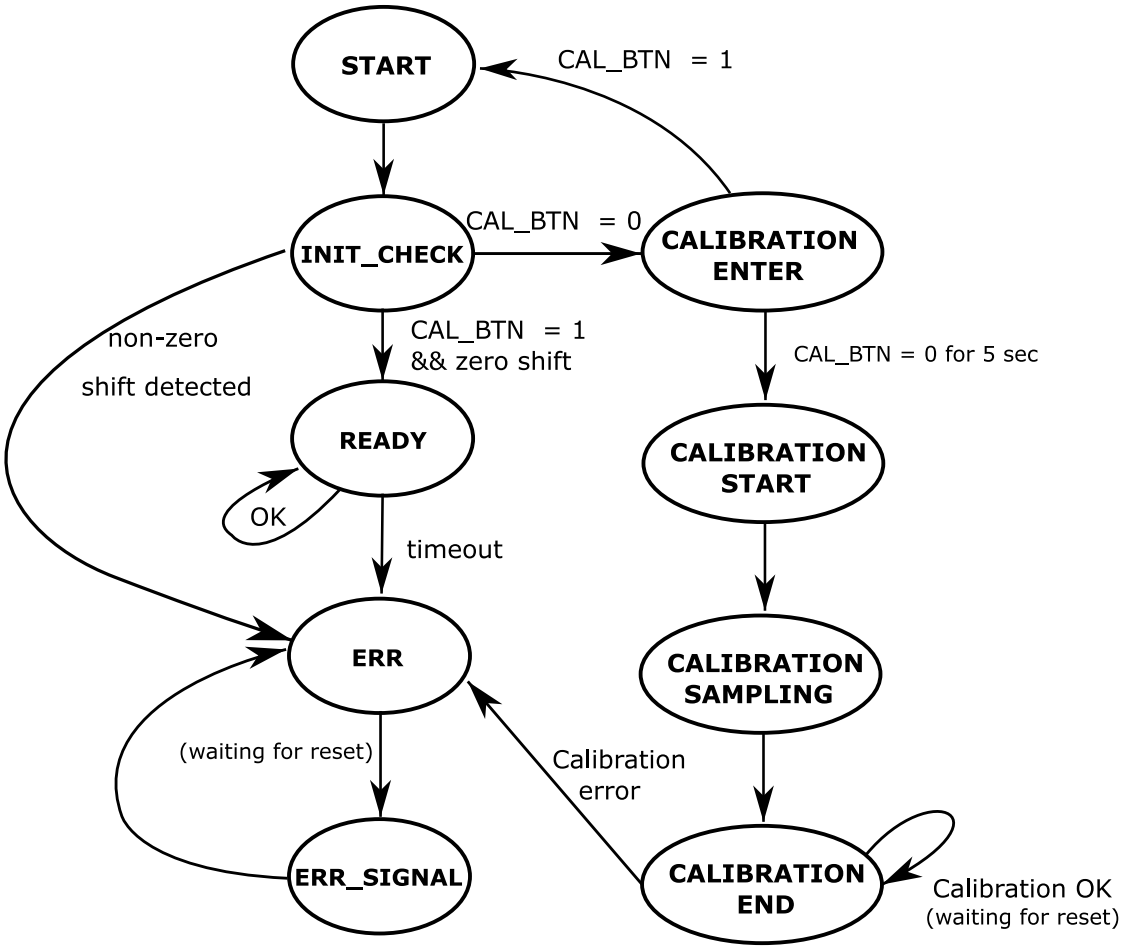
Následuje blok stavového automatu. Ten je řešený pomocí příkazu „`switch(state)`“, kdy proměnná „`states state`“ vyjadřuje aktuální stav automatu a je řešena jako výčtový typ (enumerated type), což ukazuje ukázka kódu dále (Ukázka kódu 5.8).

```
...
 typedef enum
 {
     START,
     INIT_CHECK,
     READY,
     CALIBRATION_ENTER,
     CALIBRATION_START,
     CALIBRATION_SAMPLING,
     CALIBRATION_END,
     ERR,
     ERR_SIGNAL
 } states;

...
```

Ukázka kódu 5.8: Definice výčtového typu „states“

Jak je patrné, automat má celkem 9 stavů a je spolu s možnými přechody naznačen na diagramu níže (Obr. 5.1).



Obr. 5.1: Diagram stavového automatu

Automat startuje stavem „START“. V tomto stavu je z paměti EEPROM vyčteno 8 bajtů s hodnotami maxim a minim jednotlivých senzorů. Tyto hodnoty jsou využívány v dalších stavech k vypočítávání posunu magnetu díky proložení těchto hodnot přímkou. Hodnoty byly do EEPROM uloženy v módu kalibrace, o kterém bude zmínka v dalším textu.

Po načtení hodnot ve stavu „START“ automat ihned přechází do stavu „INIT_CHECK“, ve kterém automat setrvá po dobu 5 vteřin. V tomto čase je neustále kontrolováno, zdali je posun magnetu vyhodnocován jako nulový. Pokud by během intervalu kontroly vyšel nenulový posun, mohlo by jít o chybu kalibrace a následujícím stavem proto bude stav chybový (ERR). Proto je při zapnutí napájení nutné nešlapat na akcelerační pedál po dobu alespoň 5 vteřin. Tento interval značí i LED signalizace, kdy pomalu bliká „ST2“. Pokud je někdy během těchto pěti vteřin stisknuto tlačítko kalibrace (tlačítko je zapojené proti zemi a při stisku je na GPIO logická nula), přechází okamžitě automat do potvrzovacího stavu „CALIBRATION_ENTER“. To je signalizováno rychlým blikáním LED „ST2“. V tomto stavu musí být tlačítko kalibrace pro přechod do kalibračního režimu drženo bez přerušení po dobu 5 vteřin (dokud nezačne „ST2“ trvale svítit, což značí stav kalibrace). Tím je zajištěno, že

nedojde k náhodnému překalibrování zařízení při jízdě např. vlivem otřesů. Pokud je tlačítko během potvrzovacího stavu uvolněno, automat se ihned vrací na start.

Pokud je ve stavu „INIT_CHECK“ kontrola úspěšná, a není během ní stisknuto tlačítko kalibrace, přechází automat do hlavního režimu „READY“ (Ukázka kódu 5.9), ve kterém jsou neustále vyhodnocovány hodnoty ze senzorů, ověřována jejich věrohodnost a dle toho upravovány výstupní proměnné.

```
...
case READY:
                // State READY - everything is fine - MCU is happy
                // Setting the LED indication
                LEDtimePeriod[1] = LED_OFF;

                if(ADCUpdateTicks > LastProcessTime)
                {
                    EvaluatePlausibility(&mainStruct, &outputStruct);
                    // Are the values plausible? If yes then we can set the new time limit
                    if(outputStruct.plausible)
                    {
                        // Setting of LED blinking period for indication of non-zero shift
                        if(outputStruct.shiftPercentage == 0)
                            LEDtimePeriod[0] = LED_BLINK_500ms;
                        else
                            LEDtimePeriod[0] = LED_BLINK_100ms;

                        SMTicks = _ticks + READY_TIME;
                    }
                    LastProcessTime = ADCUpdateTicks;
                }
                // If the time limit is gone - Signal is implausible and go to error mode
                if(_ticks >= SMTicks)
                {
                    nextState = ERR;
                }
                break;
...
```

Ukázka kódu 5.9: Stav „READY“

Hodnoty z ADC jsou zpracovávány funkcí „EvaluatePlausibility(…)“ (Ukázka kódu 5.10). Ta, jako výstup, vyplňuje strukturu, která obsahuje informaci o posunu, rozdílu signálů senzorů a flag značící, zdali jsou hodnoty věrohodné. Při nevěrohodnosti funkce nastaví globální chybový flag s kódovým označením chyby.

```
results EvaluatePlausibility(input *inputStruct, results *resultStruct)
{
    uint16_t shiftPercentage0, shiftPercentage1, difference;
    // If the values are in ranges, then...
    if ((inputStruct->ADCValues[0] >= MIN_RANGE_S1) && (inputStruct->ADCValues[0] <= MAX_RANGE_S1)
            && (inputStruct->ADCValues[1] >= MIN_RANGE_S2) && (inputStruct->ADCValues[1] <=
MAX_RANGE_S2))
    {
        // Calculate the percentages of shifting form both probes
        shiftPercentage0 = 1000
                * (inputStruct->ADCValues[0] - inputStruct->probeMin[0])
                / inputStruct->calibratedConst[0];
        shiftPercentage1 = 1000
                * (inputStruct->ADCValues[1] - inputStruct->probeMin[1])
                / inputStruct->calibratedConst[1];
        // Underflow protection
        if(inputStruct->ADCValues[0] <= inputStruct->probeMin[0])
            shiftPercentage0 = 0;

        if(inputStruct->ADCValues[1] <= inputStruct->probeMin[1])
            shiftPercentage1 = 0;

        // Calculate the absolute difference between the probes values
        if (shiftPercentage0 >= shiftPercentage1)
            difference = shiftPercentage0 - shiftPercentage1;
        else
            difference = shiftPercentage1 - shiftPercentage0;

        // Are they in tolerance? If yes then we can return the value, if no then sets the
errCode...
        if (difference <= DIFF_TOL)
...
```

Ukázka kódu 5.10: Část funkce „EvaluatePlausibility(…)“

Ve funkci (Ukázka kódu 5.10) se nejdříve kontroluje, zdali jsou hodnoty napětí jednotlivých senzorů v jejich pracovním rozsahu.

Pokud ano, funkce dále využívá linearity uspořádání měřící sestavy sonda-magnet, a vypočítává posun v procentech pomocí aproximace charakteristiky jako přímky. Rovnice přímky (5.1) je definována jako:

$$y(x) = kx + q \tag{5.1}$$

, kde *y(x)* je lineární funkce závislá na proměnné *x*, *q* je posun na ose y a *k* je tzv. směrnice, pro kterou platí:

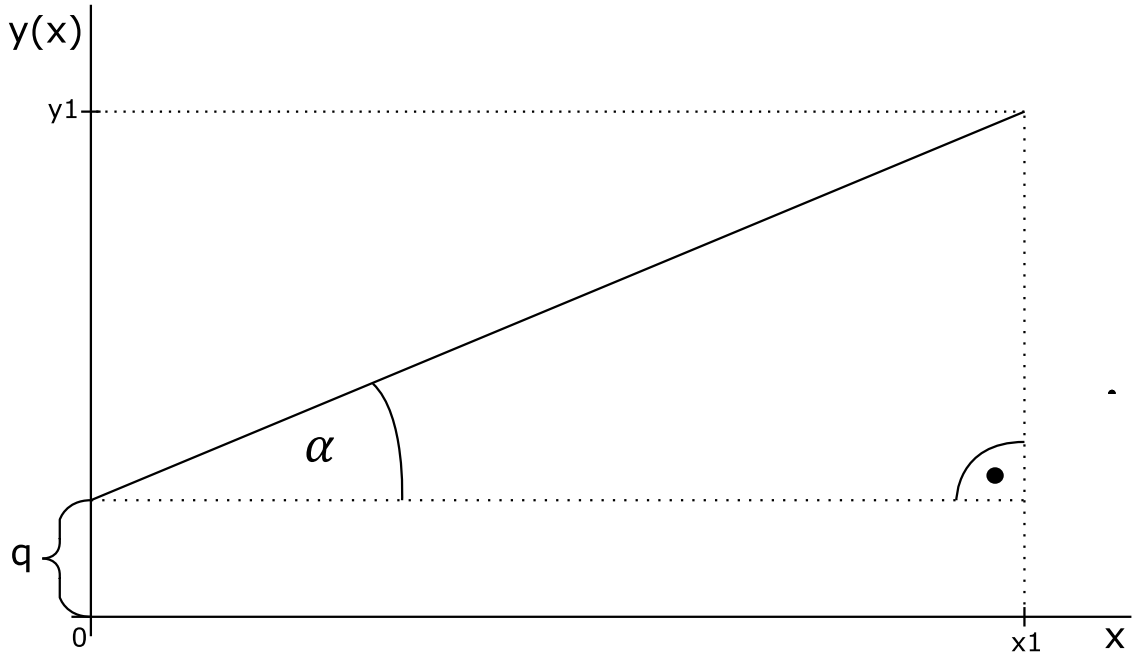
$$k = tg(\alpha) \tag{5.2}$$

, kde úhel α je úhel mezi přeponou a odvěsnou v pomyslném pravoúhlém trojúhelníku znázorněném na obrázku dále (Obr. 5.2). Z předchozích vztahů s využitím notoricky známé rovnosti:

$$\mathrm{tg}(\alpha) = \frac{\text{protilehlá odvěsna}}{\text{přilehlá odvěsna}} \tag{5.3}$$

lze napsat:

$$x(y) = \frac{\text{přilehlá odvěsna}}{\text{protilehlá odvěsna}} \cdot (y - q) \tag{5.4}$$



Obr. 5.2: Přímka

Řekneme-li, že hodnoty na ose x znázorňují dráhu magnetu od 0 do 1000 (tzn. na obrázku *x1* = 1000). Hodnoty na ose y, pak znázorňují hodnoty z analogově-digitálních převodníků. Tyto hodnoty mají své nenulové minimum (q) a také maximum ($y1$). Takto se získal vztah (5.5), který je využit ve funkci „EvaluatePlausibility(…)“ (Ukázka kódu 5.10), s tím rozdílem, že pořadí operací je uzpůsobeno tak, aby bylo možno využít celočíselného dělení a dělitel (*MAX-MIN*) je již před-vypočítaný v proměnné „calibratedConst[…]“.

$$posun(ADCVal) = \frac{1000}{MAX - MIN} \cdot (ADCVal - MIN) \tag{5.5}$$

Pokud je rozdíl drah vycházejících z naměřených hodnot jednoho a druhého senzoru větší, než dovolují pravidla (10 %), pak jsou hodnoty označeny, jako nevěrohodné, což značí flag „`bool outputStruct.plausible`“. Pokud hodnoty věrohodné jsou, je ve stavu „`READY`“ (Ukázka kódu 5.9) prodloužen čas setrvání automatu v tomto stavu o hodnotu „`READY_TIME`“. Pokud by hodnoty ze senzorů vycházely nevěrohodné po dobu delší, než je tato doba, dojde k přechodu do bezpečného chybového režimu (`ERR`).

V režimu „`ERR`“ dojde k rozsvícení stavové LED „ST1“ a po určitém intervalu dojde k přechodu do stavu „`ERR_SIGNAL`“, ve kterém je druhou stavovou LED signalizován kód

chyby jejím blikáním. Poté se znovu přechází do stavu „ERR“ a vzniká tak uzavřená smyčka. Tabulka níže (Tab. 1) obsahuje významy jednotlivých chybových kódů, jejich možné důvody a opravy.

Tab. 1: Kódy chyb, jejich důvody a možná řešení

Kód	Chyba	Možný důvod	Řešení
0	CAN	*Nefunkční sběrnice CAN*	*Kontrola konektorů, stavu sběrnice*
1	Signál senzorů	*Zkrat či rozpojení signálů z/do některého senzoru, nesoulad signálů ze senzorů (Hallovo sond)*	*Kontrola rozpojovacích DIP-switchů, kontrola senzorů*
2	Kalibrace	*Sešlápnutý pedál při spuštění, chyba při kalibraci, chybná kalibrace*	*Restart, nová kalibrace*
3	EEPROM	*Chybný zápis do EEPROM, chyba čtení*	*Restart, nová kalibrace*
4	Interlock	*Kontrolní proudovou smyčkou neprotéká proud*	*Kontrola zapojení všech zařízení ve smyčce*

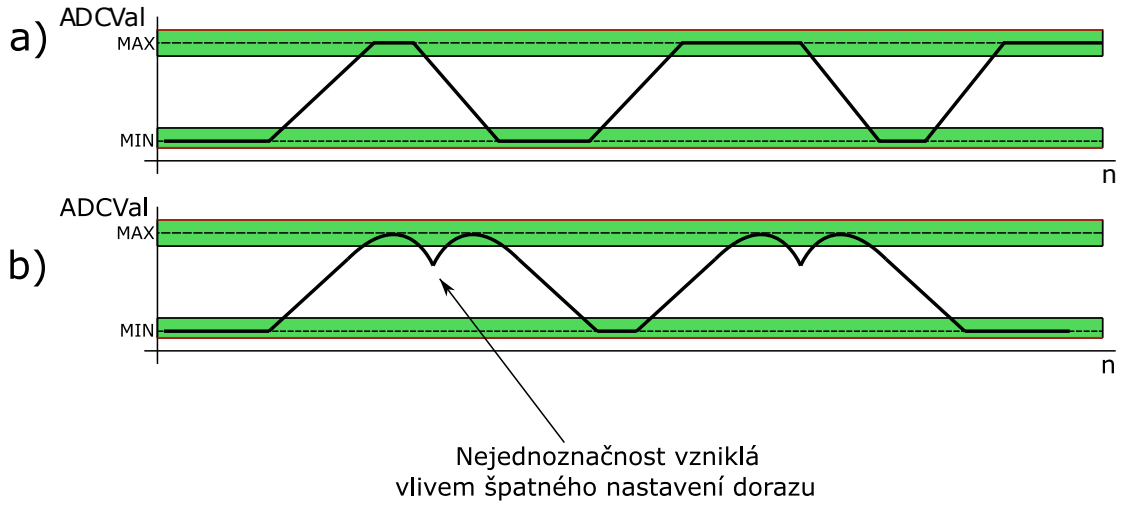
Proces kalibrace, jak již bylo naznačeno, začíná stavem „CALIBRATION_START“. Zde jsou uloženy aktuální vzorky ze senzorů jako minima, neboť je předpokládáno, a tedy nutné, aby byl při začátku kalibrace pedál uvolněn. Dále nastává stav „CALIBRATION_SAMPLING“. V tomto stavu je ukládáno do kalibračního bufferu 1000 hodnot z obou senzorů (celkem 2000 hodnot) s časovými rozestupy jednotlivých vzorků 10 ms (tento stav proto trvá 10 vteřin). Během něj je nutné opakovaně šlapat na pedál rychlostí asi jedno šlápnutí za jednu vteřinu. Rychlé šlapání by mohlo způsobit chybu kalibrace z důvodu dalšího zpracování (viz dále).

Po navzorkování těchto dat, přechází automat do stavu „CALIBRATION_END“, ve kterém jsou data z bufferu vyhodnocena tím způsobem, že se najde maximum, které se uloží a následně se hledají zlomy trendu charakteristik (změny znaménka směrnice tečny). Tyto zlomy musí nastat pouze v blízkosti maxim a minim. To se hlídá z důvodu kontroly správného nastavení dorazů. Pokud by byl doraz magnetu špatně nastaven, charakteristika by měla další zlomy, jak ukazuje spodní obrázek níže (Obr. 5.3b), ale již ne v blízkosti maxima nebo minima (tj. mimo zelenou toleranci). Správný průběh charakteristiky je naznačen na horním obrázku níže (Obr. 5.3a).

Nastane-li parazitní zlom ještě v zelené toleranci, nebude to mít žádný negativní vliv, jelikož při vyhodnocování posunu v již zmíněné funkci „EvaluatePlausibility(…)“ ve stavu „READY“, se vše, co je nad hranicí zelené tolerance, bere jako plně sešlápnuto. Vzniklo by tak jen lehké prodloužení „hluchého“ místa na konci dráhy pedálu. Výjimkou je

samozřejmě překročení červeně naznačené vrchní (resp. i spodní) hranice, která by znamenala poruchu senzoru.

Pokud je vše v pořádku, jsou hodnoty maxim a minim uloženy do EEPROM a automat se uzavírá v tomto stavu do nekonečné smyčky, kde čeká na manuální reset kontroléru. Při chybné kalibraci přechází do chybového stavu „ERR“.



Obr. 5.3: Ukázka průběhu signálu ze sond
a) Při správném nastavení
b) Při špatně nastaveném dorazu

Následující tabulka (Tab. 2) pro přehlednost shrnuje již v textu zmíněné LED signalizace jednotlivých stavů. Neuvedené stavy nejsou signalizovány z důvodu jejich krátkého trvání.

Tab. 2: Tabulka svitu stavových LED dle stavů stavového automatu

Stav	Poznámka	Stavová LED	
		ST1	ST2
INIT_CHECK	-	vypnuto	blikání 250ms
READY	nesešlápnuto	blikání 500ms	vypnuto
	sešlápnuto	blikání 100ms	vypnuto
CLIBRATION_ENTER	-	vypnuto	blikání 100ms
CALIBRATION_START	-	vypnuto	trvalý svit
CALIBRATION_END	po úspěšném nakalibrování	vypnuto	vypnuto
ERR	resp. ERR_SIGNAL	trvalý svit	blikání kódu chyby

5.2.4 Kontrola proudové smyčky

Následně se po bloku stavového automatu v nekonečné smyčce kontroluje, zdali protéká proud proudovou smyčkou (Interlock). To je indikováno přes optočlen na GPIO pin mikrokontroléru (více v kapitole 3.3.5). Pokud je pin na úrovni logické nuly, smyčkou protéká proud a nic se v programu nestane. Pokud je na pinu logická jednička, zařízení

přechází do chybového stavu tím, že je přepsána proměnná následujícího stavu stavového automatu (Ukázka kódu 5.11).

```
…
        // Interlock loop check
        if(GPIOPinRead(LOOP_STATE) && errCode != ERR_INTERLOCK)
        {
            errCode = ERR_INTERLOCK;
            nextState = ERR;
        }
…
```

Ukázka kódu 5.11: Kontrola proudové smyčky (Interlock)

5.2.5 Posílání zpráv po sběrnici CAN

Poslední částí nekonečné smyčky je podmínka zajišťující pravidelné odesílání zpráv na sběrnici CAN. K časování slouží extra proměnná „`uint64_t CANTicks`“. Interval posílání zpráv je nastavený na 70 ms. Pravidla deklarují, že po vyhodnocení nevěrohodnosti delší než 100 ms musí být ihned vypnuty motory (viz kapitola 1.4). Pro jisté splnění je tedy třeba posílat zprávy po sběrnici CAN rychleji než každých 100 ms, aby měl nadřazený kontrolér ještě určitou rezervu vypnout motory.

Jedna zpráva obsahuje 3 bajty. První bajt je kód chyby. Ten je při bezchybném stavu nulový a v případě chyby značí jeho obsah kód chyby dle již uvedené tabulky (Tab. 1) (kromě chyby sběrnice CAN). Druhý a třetí bajt obsahuje informaci o stisknutí pedálu v rozsahu od 0 do 1000.

Pokud nastane na sběrnici chyba, a zpráva se nepodaří odeslat, je tato chyba signalizována pouze svitem LED „ST1“ (kód chyby nula, tudíž „ST2“ nebliká). Do chodu stavového automatu se nijak nezasahuje, a po obnovení funkce sběrnice zařízení automaticky pracuje dále.

```
bool SendData2CAN(results *dataStruct)
{
    if (!CANBusy)
    {
        CANMsgData = 0 | (errCode) | (dataStruct->shiftPercentage << 8);

        // Send the CAN message using object number 1
        CANMessageSet(CAN0_BASE, 1, &CANMessage, MSG_OBJ_TYPE_TX);

        //Set the busy flag
        CANBusy = true;

        return true;
    }
    else
    {
        return false;
    }
}
```

Ukázka kódu 5.12: Funkce pro odesílání dat po sběrnici CAN

Zhodnocení a závěr

V této diplomové práci bylo provedeno shrnutí platných pravidel a doporučení pro konstrukci snímače polohy akceleračního pedálu studentských formulí, účastnících se v soutěžích Formula Student.

Na základě této rešerše a zkušeností z mé předcházející bakalářské práce byla zvolena koncepce snímání polohy pedálů. Bylo navrženo obvodové schéma snímače, a také deska plošných spojů s ohledem na geometrii navrhovaných pedálů, včetně krabičky.

Takto navržený snímač byl vyroben, oživen a naprogramován. Výsledkem je tedy plně funkční snímač polohy pedálu, který by měl bez problémů projít inspekční předzávodní prohlídkou vozidla pro kontrolu shody s pravidly FSAE a FSG a může být použit na studentské formuli závodního týmu Západočeské univerzity. Alternativně může být využit i na studentské motokáře, jejíž stavba v současné době také probíhá.

Snímač využívá jako senzory dvojici Hallovo sond. Pokud by byla v budoucnu vyžadována vyšší spolehlivost snímače, jednou z možností, jak tomu jít naproti, je použití třetího senzoru (Hallovo sondy) umístěné např. ze spodní strany magnetu. Pak by (jak dovolují pravidla – viz kapitola 1.3), při poruše jednoho senzoru a shodě signálů ze zbylých dvou, mohlo vozidlo pokračovat v jízdě a dokončit závod i s touto poruchou.

Další miniaturizace tohoto snímače je možná např. použitím nižšího magnetu, či věnováním dalšího času na přeuspořádání součástek pro docílení ještě větší hustoty součástek na desce plošných spojů.

Pouzdro snímače na sobě zatím nemá namontovaný konektor. Je tomu tak z důvodu stále probíhajícího návrhu kabelových svazků ve formuli. Konektor panelového typu bude v budoucnu umístěn v pouzdře na místě, kde to bude pro svazek nejvýhodnější.

Literatura

[1] FORMULA SAE: Rules 2021 [online]. Version 1.0. SAE International, 2020, 136 s. [cit. 2021-10-18]. Dostupné z: https://www.fsaeonline.com/cdsweb/gen/DownloadDocument.aspx?DocumentID=6d9f4b51-a642-425c-bfdf-5f95b4e5e10b

[2] FORMULA STUDENT GERMANY: Rules for FSG 2022 v. 0.9 [online]. 2021 [cit. 2021-11-15]. Dostupné z: https://www.formulastudent.de/fsg/rules/

[3] TROJAN, Adam. Systém snímání a vyhodnocení polohy pedálu elektrovozidla. Plzeň, 2020, 65 s. Dostupné také z: https://dspace5.zcu.cz/handle/11025/41857. Bakalářská práce. Západočeská univerzita v Plzni. Vedoucí práce Ing. Martin Zavřel.

[4] Sensing and Control: Hall effect sensing and application [online]. Freeport (Ilinois): Honeywell, [1998], 126 s. [cit. 2022-03-23]. 005715-2-EN IL50. Dostupné z: http://denethor.wlu.ca/pc300/projects/sensors/hallbook.pdf

[5] DRV5055-Q1 Automotive Ratiometric Linear Hall Effect Sensor [online]. Dallas: Texas Instruments, 2018, 31 s. [cit. 2021-10-18]. Dostupné z: https://www.ti.com/lit/ds/symlink/drv5055-q1.pdf. Datasheet.

[6] ZHANG, Neal, Daniel LI, Vincent ZHANG a Andy CHEN. Reduce Conducted EMI in Automotive Buck Converter Applications: Application Report [online]. Dallas: Texas Instruments, 2019, 21 s. [cit. 2022-03-23]. SNVA886. Dostupné z: www.ti.com/lit/an/snva886/snva886.pdf

[7] MARTIN, Alan. AN-2162 Simple Success With Conducted EMI From DCDC Converters: Application Report [online]. Rev. 2013. Dallas: Texas Instruments, 2011, 13 s. [cit. 2022-03-23]. SNVA489C. Dostupné z: https://www.ti.com/lit/an/snva489c/snva489c.pdf

[8] LM22671/-Q1 42 V, 500 mA SIMPLE SWITCHER® Step-Down Voltage Regulator with Features [online]. Rev. 2014. Dallas: Texas Instruments, 2008, 32 s. [cit. 2022-03-23]. Dostupné z: https://www.ti.com/lit/ds/symlink/lm22671.pdf. Datasheet.

[9] MANCINI, Ron. Op Amps For Everyone: Design Reference [online]. Dallas: Texas Instruments, 2002, 464 s. [cit. 2022-03-23]. Dostupné z: https://web.mit.edu/6.101/www/reference/op_amps_everyone.pdf

[10] PINKER, Jiří a Václav KOUCKÝ. Analogové elektronické systémy. 2. vyd. Plzeň: Západočeská univerzita, 1999. ISBN 80-7082-506-5.

[11] Tiva™ TM4C123GH6PM Microcontroller [online]. (Rev. E). Dallas: Texas Instruments, ©2022, 1409 s. [cit. 2022-04-02]. DS-TM4C123GH6PM-15842.2741 SPMS376E. Dostupné z: http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf. Datasheet.

[12] 3.3-V CAN TRANSCEIVERS: SN65HVD230-232 [online]. Dallas: Texas Instruments, © 2011, 33 s. [cit. 2022-04-02]. SLOS346K. Dostupné z: https://riceproject.fel.zcu.cz/Altium/PDF/Semiconductor/Logic/Driver/sn65hvd230-232.pdf. Datasheet.

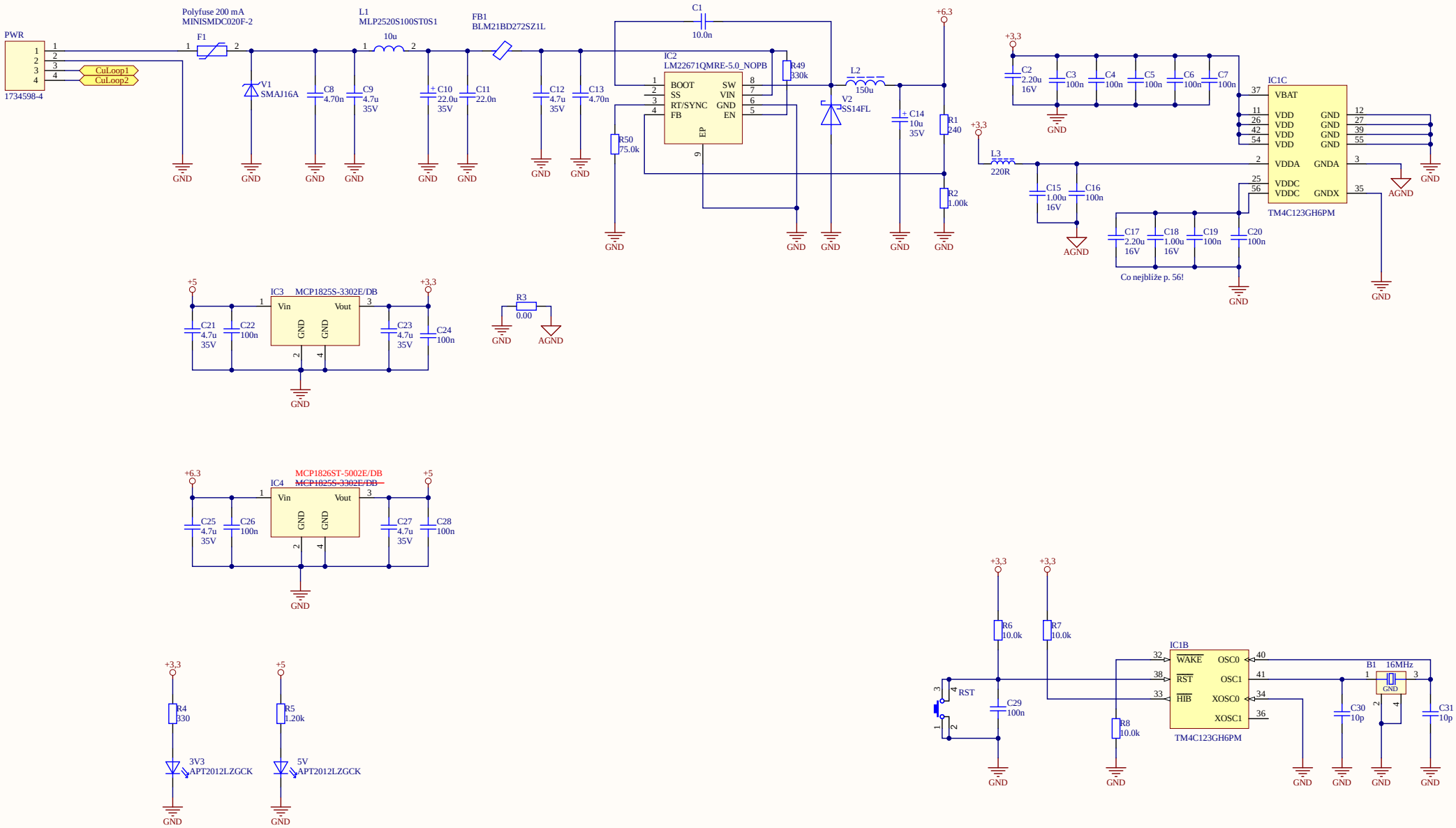
[13] Common Mode Choke: 1812CANbus [online]. Rev. 10/09/18. Cary (Illinois): Coilcraft, 2018, 2 s. [cit. 2022-05-09]. 1201-2. Dostupné z: https://www.farnell.com/datasheets/2876985.pdf. Datasheet.

[14] HCPL-354: AC Input Phototransistor Optocoupler [online]. San Jose (Kalifornie): Avago (Broadcom), 2007 [cit. 2022-05-09]. AV02-0775EN. Dostupné z: https://cz.mouser.com/datasheet/2/678/av02-0775en-1827504.pdf. Datasheet.

[15] BAKER, Bonnie. Designing an anti-aliasing filter for ADCs in the frequency domain [online]. Dallas: Texas Instruments, 2015, 5 s. [cit. 2022-05-09]. AAJ 2Q 2015. Dostupné z: https://www.ti.com/lit/an/slyt626/slyt626.pdf

Příloha I

Schéma napájecí části snímače polohy plynového pedálu.

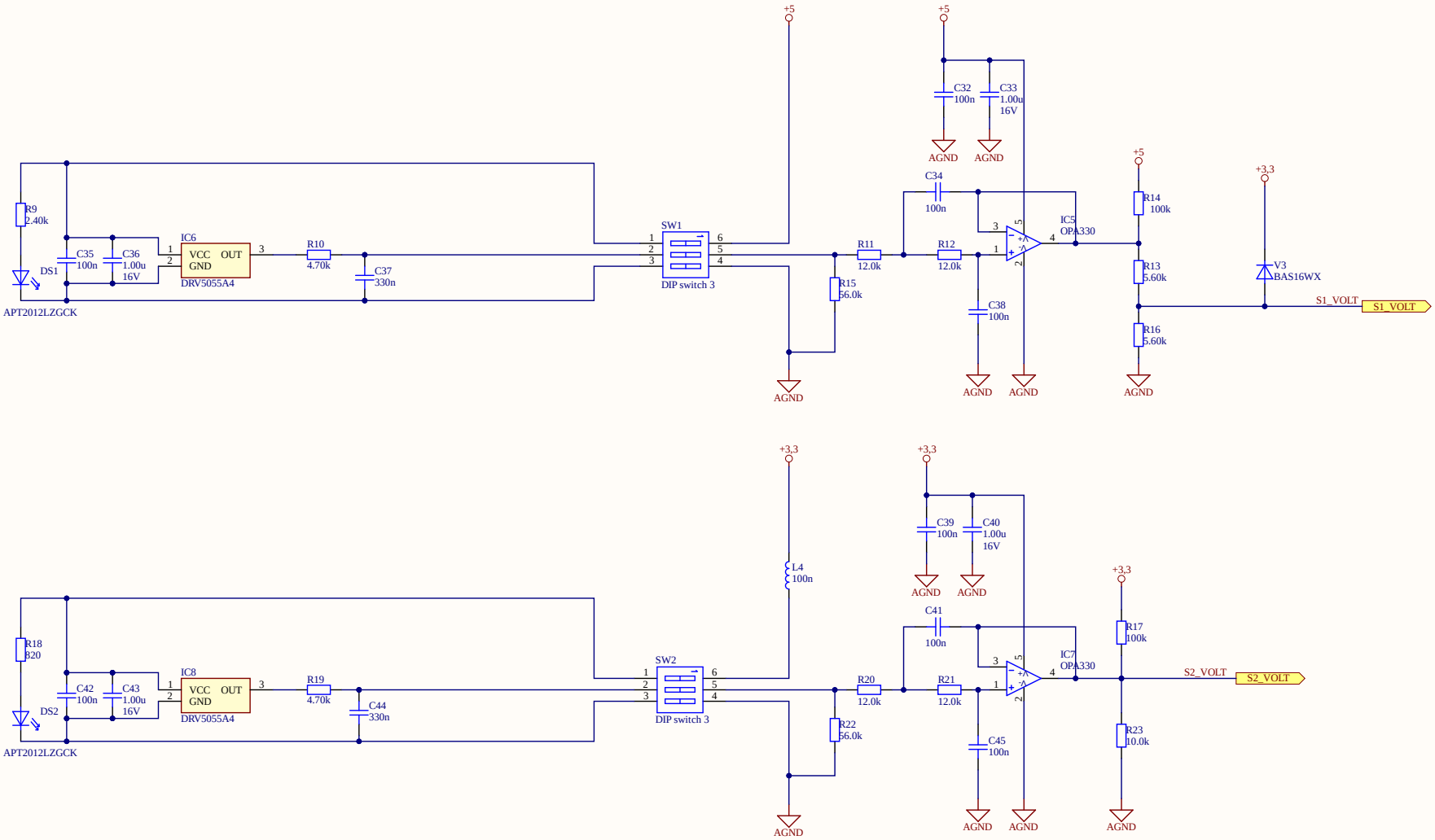


RICE	Size: A3	Date: 24.05.2022	Title: **Formula Student Pedals APPS**
Regional Innovation Centre for Electrical Engineering Univerzitní 26 306 14 Plzen Czech Republic	Author: Adam Trojan		Subtitle: **[SUBTITLE]** Variant: **[No Variations]**
	Checked by: Martin Zavrel		Nr.: Revision: **A1**
	Approved by: Lubos Streit		Type: Sheet 1 of 1
			File: Sources.SchDoc

Note: Data in fields enter via project/document parameters.

Příloha II

Schéma analogové části snímače polohy plynového pedálu.

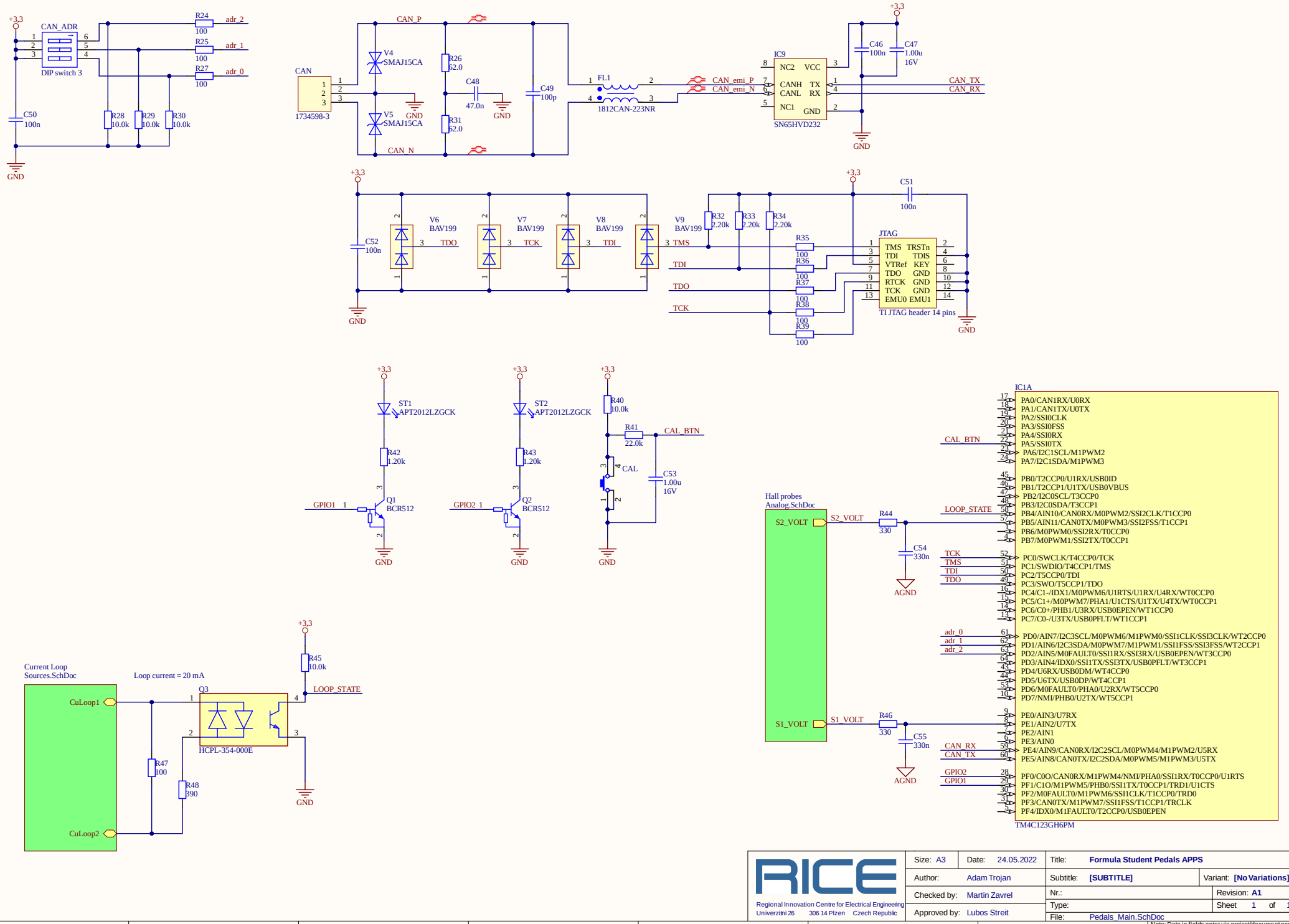


RICE	Size: A3	Date: 24.05.2022	Title: **Formula Student Pedals APPS**
Regional Innovation Centre for Electrical Engineering	Author: Adam Trojan		Subtitle: **[SUBTITLE]** Variant: **[No Variations]**
Univerzitní 26 306 14 Plzen Czech Republic	Checked by: Martin Zavrel		Nr.: Revision: **A1**
	Approved by: Lubos Streit		Type: Sheet 1 of 1
			File: Analog.SchDoc

Note: Data in fields enter via project/document parameters.

Příloha III

Schéma číslicové části snímače polohy plynového pedálu.


CAN_ADR
DIP switch 3
R24 100 adr_2
R25 100 adr_1
R27 100 adr_0
C50 100n
R28 10.0k
R29 10.0k
R30 10.0k
+3,3
GND
CAN
1734598-3
CAN_P
CAN_N
V4 SMAJ15CA
V5 SMAJ15CA
R26 62.0
R31 62.0
C48 47.0n
C49 100p
FL1
1812CAN-223NR
CAN_emi_P
CAN_emi_N
IC9
NC2 VCC
CANH TX
CANL RX
NC1 GND
SN65HVD232
C46 100n
C47 1.00u 16V
CAN_TX
CAN_RX
V6 BAV199
V7 BAV199
V8 BAV199
V9 BAV199
C52 100n
TDO
TCK
TDI
TMS
R32 2.20k
R33 2.20k
R34 2.20k
R35 100
R36 100
R37 100
R38 100
R39 100
C51 100n
JTAG
TMS TRSTn
TDI TDIS
VTRef KEY
TDO GND
RTCK GND
TCK GND
EMU0 EMU1
TI JTAG header 14 pins
ST1 APT2012LZGCK
ST2 APT2012LZGCK
R42 1.20k
R43 1.20k
R40 10.0k
R41 22.0k
CAL_BTN
CAL
C53 1.00u 16V
GPIO1
GPIO2
Q1 BCR512
Q2 BCR512
IC1A
TM4C123GH6PM
PA0/CAN1RX/U0RX
PA1/CAN1TX/U0TX
PA2/SSI0CLK
PA3/SSI0FSS
PA4/SSI0RX
PA5/SSI0TX
PA6/I2C1SCL/M1PWM2
PA7/I2C1SDA/M1PWM3
PB0/T2CCP0/U1RX/USB0ID
PB1/T2CCP1/U1TX/USB0VBUS
PB2/I2C0SCL/T3CCP0
PB3/I2C0SDA/T3CCP1
PB4/AIN10/CAN0RX/M0PWM2/SSI2CLK/T1CCP0
PB5/AIN11/CAN0TX/M0PWM3/SSI2FSS/T1CCP1
PB6/M0PWM0/SSI2RX/T0CCP0
PB7/M0PWM1/SSI2TX/T0CCP1
PC0/SWCLK/T4CCP0/TCK
PC1/SWDIO/T4CCP1/TMS
PC2/T5CCP0/TDI
PC3/SWO/T5CCP1/TDO
PC4/C1-/IDX1/M0PWM6/U1RTS/U1RX/U4RX/WT0CCP0
PC5/C1+/M0PWM7/PHA1/U1CTS/U1TX/U4TX/WT0CCP1
PC6/C0+/PHB1/U3RX/USB0EPEN/WT1CCP0
PC7/C0-/U3TX/USB0PFLT/WT1CCP1
PD0/AIN7/I2C3SCL/M0PWM6/M1PWM0/SSI1CLK/SSI3CLK/WT2CCP0
PD1/AIN6/I2C3SDA/M0PWM7/M1PWM1/SSI1FSS/SSI3FSS/WT2CCP1
PD2/AIN5/M0FAULT0/SSI1RX/SSI3RX/USB0EPEN/WT3CCP0
PD3/AIN4/IDX0/SSI1TX/SSI3TX/USB0PFLT/WT3CCP1
PD4/U6RX/USB0DM/WT4CCP0
PD5/U6TX/USB0DP/WT4CCP1
PD6/M0FAULT0/PHA0/U2RX/WT5CCP0
PD7/NMI/PHB0/U2TX/WT5CCP1
PE0/AIN3/U7RX
PE1/AIN2/U7TX
PE2/AIN1
PE3/AIN0
PE4/AIN9/CAN0RX/I2C2SCL/M0PWM4/M1PWM2/U5RX
PE5/AIN8/CAN0TX/I2C2SDA/M0PWM5/M1PWM3/U5TX
PF0/C0O/CAN0RX/M1PWM4/NMI/PHA0/SSI1RX/T0CCP0/U1RTS
PF1/C1O/M1PWM5/PHB0/SSI1TX/T0CCP1/TRD1/U1CTS
PF2/M0FAULT0/M1PWM6/SSI1CLK/T1CCP0/TRD0
PF3/CAN0TX/M1PWM7/SSI1FSS/T1CCP1/TRCLK
PF4/IDX0/M1FAULT0/T2CCP0/USB0EPEN
CAL_BTN
LOOP_STATE
TCK
TMS
TDI
TDO
adr_0
adr_1
adr_2
CAN_RX
CAN_TX
GPIO2
GPIO1
Hall probes
Analog.SchDoc
S2_VOLT
S1_VOLT
R44 330
R46 330
C54 330n
C55 330n
AGND
Current Loop
Sources.SchDoc
Loop current = 20 mA
CuLoop1
CuLoop2
Q3
HCPL-354-000E
R45 10.0k
R47 100
R48 390
RICE
Regional Innovation Centre for Electrical Engineering
Univerzitní 26 306 14 Plzen Czech Republic
Size: A3
Date: 24.05.2022
Author: Adam Trojan
Checked by: Martin Zavrel
Approved by: Lubos Streit
Title: Formula Student Pedals APPS
Subtitle: [SUBTITLE]
Variant: [No Variations]
Nr.:
Revision: A1
Type:
Sheet 1 of 1
File: Pedals_Main.SchDoc
Note: Data in fields enter via project/document parameters.

Příloha IV

Motiv desky plošných spojů – horní vrstva (TOP)

SOLDER MASK	☒ Top side	☒ Bot side
REMOVABLE MASK	☐ Top side	☐ Bot side
SILKSCREEN	☒ Top side	☒ Bot side

DRILL

☒ Finished diameter ☐ Tool diameter

Nr. of drills:

OUTLINE

☐ Cutted per unit ☐ Milled by bridges

☒ Milled per unit ☐ 'U' grooved

☐ No machining

MILLING

☒ Axis of line is border of milling ☐ Axis of line is trace of milling

MATERIAL

☒ FR4 ☐ Other _____

TRACK FINISH

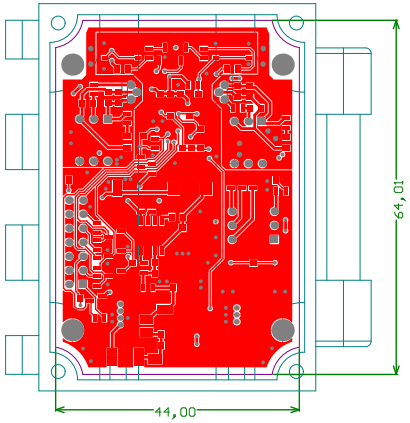
☐ HAL ☐ Immersion gold

☐ Immersion tin ☒ Galvanic gold

☐ Organic coating

Layer	Name	Material	Thickness	Constant	Board Layer Stack
	TopOverlay				
	TopSolder	Solder Resist	0,040mm	3.5	
1	TopLayer		0,035mm		
	Dielectric1	FR-4	0,113mm	4.28	
2	+Vcc		0,018mm		
	Dielectric2	FR-4	1,000mm	4.2	
3	GND		0,018mm		
	Dielectric3	FR-4	0,127mm	4.2	
4	BottomLayer		0,035mm		
	BottomSolder	Solder Resist	0,040mm	3.5	
	BottomOverlay				

TOTAL PCB thicknes = XX mm



	Layer Name	Symbol	File
TOP	VARNISH MASK	GM31 _ A1.	GM31
	SILKSCREEN	GTO _ A1.	GTO
	SOLDER PASTE	GTP _ A1.	GTP
	SOLDER MASK	GTS _ A1.	GTS
	TOP LAYER	GTL _ A1.	GTL
INNER	INNER LAYER	GP1 _ A1.	G1
	INNER LAYER	G2 _ A1.	G2
	INNER LAYER	G3 _ A1.	G3
	INNER LAYER	G4 _ A1.	G4
	INNER LAYER	G5 _ A1.	G5
	INNER LAYER	G6 _ A1.	G6
	INNER LAYER	G7 _ A1.	G7
	INNER LAYER	G8 _ A1.	G8
BOT	BOTTOM LAYER	GBL _ A1.	GBL
	SOLDER MASK	GBS _ A1.	GBS
	SOLDER PASTE	GBP _ A1.	GBP
	SILKSCREEN	GBO _ A1.	GBO
	VARNISH MASK	GM32 _ A1.	GM32
TECHNOLOGY	BOARD OUTLINE	GM1 _ A1.	GM1
	DRILL POSITION	TXT _ A1.	DRL
	DRILL TOOL	DRR _ A1.	DRR
	MILLING	GM9 _ A1.	GM9
	APERTURE DEF	APR _ A1.	GM9

RICE

Regional Innovation Centre for Electrical Engineering
Univerzitni 8 306 14 Plzen Czech Republic

Size: A3	Date: 24.05.2022	Title: **Formula Student Pedals APPS**	
Author: Adam Trojan		Subtitle **PCB Design**	
Checked by: Martin Zavrel		Nr.:	Revision: **A1**
Aprooved by: Lubos Streit		Type:	Sheet 1 of 1
		File: Pedals_APPS.PcbDoc	

Příloha V

Motiv desky plošných spojů – 1. vnitřní vrstva (+Vcc)

SOLDER MASK	[x] Top side	[x] Bot side
REMOVABLE MASK	[ ] Top side	[ ] Bot side
SILKSCREEN	[x] Top side	[x] Bot side

DRILL

[x] Finished diameter [ ] Tool diameter

Nr. of drills:

OUTLINE

[ ] Cutted per unit [ ] Milled by bridges

[x] Milled per unit [ ] 'U' grooved

[ ] No machining

MILLING

[x] Axis of line is border of milling [ ] Axis of line is trace of milling

MATERIAL

[x] FR4 [ ] Other _____

TRACK FINISH

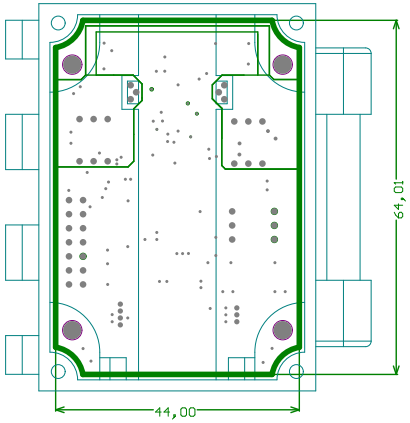
[ ] HAL [ ] Immersion gold

[ ] Immersion tin [x] Galvanic gold

[ ] Organic coating

Layer	Name	Material	Thickness	Constant	Board Layer Stack
	TopOverlay				
	TopSolder	Solder Resist	0,040mm	3.5	
1	TopLayer		0,035mm		
	Dielectric1	FR-4	0,113mm	4.28	
2	+Vcc		0,018mm		
	Dielectric2	FR-4	1,000mm	4.2	
3	GND		0,018mm		
	Dielectric3	FR-4	0,127mm	4.2	
4	BottomLayer		0,035mm		
	BottomSolder	Solder Resist	0,040mm	3.5	
	BottomOverlay				

TOTAL PCB thicknes = XX mm



	Layer Name	Symbol	File
TOP	VARNISH MASK	GM31 _ A1.	GM31
	SILKSCREEN	GTO _ A1.	GTO
	SOLDER PASTE	GTP _ A1.	GTP
	SOLDER MASK	GTS _ A1.	GTS
	TOP LAYER	GTL _ A1.	GTL
INNER	INNER LAYER	GP1 _ A1.	G1
	INNER LAYER	G2 _ A1.	G2
	INNER LAYER	G3 _ A1.	G3
	INNER LAYER	G4 _ A1.	G4
	INNER LAYER	G5 _ A1.	G5
	INNER LAYER	G6 _ A1.	G6
	INNER LAYER	G7 _ A1.	G7
	INNER LAYER	G8 _ A1.	G8
BOT	BOTTOM LAYER	GBL _ A1.	GBL
	SOLDER MASK	GBS _ A1.	GBS
	SOLDER PASTE	GBP _ A1.	GBP
	SILKSCREEN	GBO _ A1.	GBO
	VARNISH MASK	GM32 _ A1.	GM32
TECHNOLOGY	BOARD OUTLINE	GM1 _ A1.	GM1
	DRILL POSITION	TXT _ A1.	DRL
	DRILL TOOL	DRR _ A1.	DRR
	MILLING	GM9 _ A1.	GM9
	APERTURE DEF	APR _ A1.	GM9

RICE

Regional Innovation Centre for Electrical Engineering
Univerzitni 8 306 14 Plzen Czech Republic

Size: A3	Date: 24.05.2022	Title: **Formula Student Pedals APPS**	
Author: Adam Trojan		Subtitle **PCB Design**	
Checked by: Martin Zavrel		Nr.:	Revision: **A1**
Aprooved by: Lubos Streit		Type:	Sheet 1 of 1
		File: Pedals_APPS.PcbDoc	

Příloha VI

Motiv desky plošných spojů – 2. vnitřní vrstva (GND)

SOLDER MASK	[x] Top side	[x] Bot side
REMOVABLE MASK	[ ] Top side	[ ] Bot side
SILKSCREEN	[x] Top side	[x] Bot side

DRILL

[x] Finished diameter [ ] Tool diameter

Nr. of drills:

OUTLINE

[ ] Cutted per unit [ ] Milled by bridges

[x] Milled per unit [ ] 'U' grooved

[ ] No machining

MILLING

[x] Axis of line is border of milling [ ] Axis of line is trace of milling

MATERIAL

[x] FR4 [ ] Other _____

TRACK FINISH

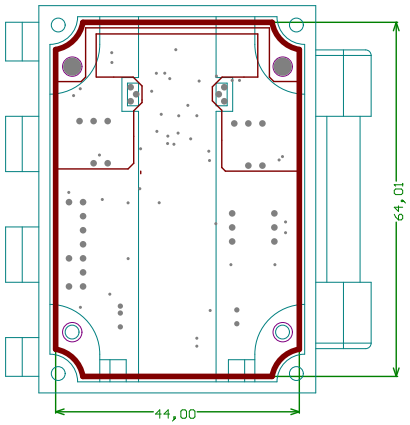
[ ] HAL [ ] Immersion gold

[ ] Immersion tin [x] Galvanic gold

[ ] Organic coating

Layer	Name	Material	Thickness	Constant	Board Layer Stack
	TopOverlay				
	TopSolder	Solder Resist	0,040mm	3.5	
1	TopLayer		0,035mm		
	Dielectric1	FR-4	0,113mm	4.28	
2	+Vcc		0,018mm		
	Dielectric2	FR-4	1,000mm	4.2	
3	GND		0,018mm		
	Dielectric3	FR-4	0,127mm	4.2	
4	BottomLayer		0,035mm		
	BottomSolder	Solder Resist	0,040mm	3.5	
	BottomOverlay				

TOTAL PCB thicknes = XX mm



	Layer Name	Symbol	File
TOP	VARNISH MASK	GM31 _ A1.	GM31
	SILKSCREEN	GTO _ A1.	GTO
	SOLDER PASTE	GTP _ A1.	GTP
	SOLDER MASK	GTS _ A1.	GTS
	TOP LAYER	GTL _ A1.	GTL
INNER	INNER LAYER	GP1 _ A1.	G1
	INNER LAYER	G2 _ A1.	G2
	INNER LAYER	G3 _ A1.	G3
	INNER LAYER	G4 _ A1.	G4
	INNER LAYER	G5 _ A1.	G5
	INNER LAYER	G6 _ A1.	G6
	INNER LAYER	G7 _ A1.	G7
	INNER LAYER	G8 _ A1.	G8
BOT	BOTTOM LAYER	GBL _ A1.	GBL
	SOLDER MASK	GBS _ A1.	GBS
	SOLDER PASTE	GBP _ A1.	GBP
	SILKSCREEN	GBO _ A1.	GBO
	VARNISH MASK	GM32 _ A1.	GM32
TECHNOLOGY	BOARD OUTLINE	GM1 _ A1.	GM1
	DRILL POSITION	TXT _ A1.	DRL
	DRILL TOOL	DRR _ A1.	DRR
	MILLING	GM9 _ A1.	GM9
	APERTURE DEF	APR _ A1.	GM9

RICE

Regional Innovation Centre for Electrical Engineering
Univerzitni 8 306 14 Plzen Czech Republic

Size: A3	Date: 24.05.2022	Title: **Formula Student Pedals APPS**	
Author: Adam Trojan		Subtitle **PCB Design**	
Checked by: Martin Zavrel		Nr.:	Revision: **A1**
Aprooved by: Lubos Streit		Type:	Sheet 1 of 1
		File: Pedals_APPS.PcbDoc	

Příloha VII

Motiv desky plošných spojů – spodní vrstva (BOTTOM)

SOLDER MASK	[x] Top side	[x] Bot side	
REMOVABLE MASK	[ ] Top side	[ ] Bot side	
SILKSCREEN	[x] Top side	[x] Bot side	

DRILL

[x] Finished diameter [ ] Tool diameter

Nr. of drills:

OUTLINE

[ ] Cutted per unit [ ] Milled by bridges

[x] Milled per unit [ ] 'U' grooved

[ ] No machining

MILLING

[x] Axis of line is border of milling [ ] Axis of line is trace of milling

MATERIAL

[x] FR4 [ ] Other _____

TRACK FINISH

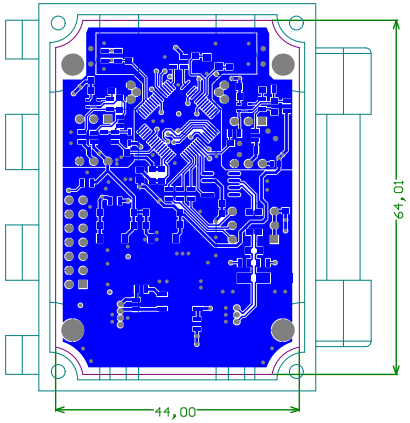
[ ] HAL [ ] Immersion gold

[ ] Immersion tin [x] Galvanic gold

[ ] Organic coating

Layer	Name	Material	Thickness	Constant	Board Layer Stack
	TopOverlay				
	TopSolder	Solder Resist	0,040mm	3.5	
1	TopLayer		0,035mm		
	Dielectric1	FR-4	0,113mm	4.28	
2	+Vcc		0,018mm		
	Dielectric2	FR-4	1,000mm	4.2	
3	GND		0,018mm		
	Dielectric3	FR-4	0,127mm	4.2	
4	BottomLayer		0,035mm		
	BottomSolder	Solder Resist	0,040mm	3.5	
	BottomOverlay				

TOTAL PCB thicknes = XX mm



	Layer Name	Symbol	File
TOP	VARNISH MASK	GM31 _ A1.	GM31
	SILKSCREEN	GTO _ A1.	GTO
	SOLDER PASTE	GTP _ A1.	GTP
	SOLDER MASK	GTS _ A1.	GTS
	TOP LAYER	GTL _ A1.	GTL
INNER	INNER LAYER	GP1 _ A1.	G1
	INNER LAYER	G2 _ A1.	G2
	INNER LAYER	G3 _ A1.	G3
	INNER LAYER	G4 _ A1.	G4
	INNER LAYER	G5 _ A1.	G5
	INNER LAYER	G6 _ A1.	G6
	INNER LAYER	G7 _ A1.	G7
	INNER LAYER	G8 _ A1.	G8
BOT	BOTTOM LAYER	GBL _ A1.	GBL
	SOLDER MASK	GBS _ A1.	GBS
	SOLDER PASTE	GBP _ A1.	GBP
	SILKSCREEN	GBO _ A1.	GBO
	VARNISH MASK	GM32 _ A1.	GM32
TECHNOLOGY	BOARD OUTLINE	GM1 _ A1.	GM1
	DRILL POSITION	TXT _ A1.	DRL
	DRILL TOOL	DRR _ A1.	DRR
	MILLING	GM9 _ A1.	GM9
	APERTURE DEF	APR _ A1.	GM9

RICE

Regional Innovation Centre for Electrical Engineering
Univerzitni 8 306 14 Plzen Czech Republic

Size: A3	Date: 24.05.2022	Title: **Formula Student Pedals APPS**	
Author: Adam Trojan		Subtitle **PCB Design**	
Checked by: Martin Zavrel		Nr.:	Revision: **A1**
Aprooved by: Lubos Streit		Type:	Sheet 1 of 1
		File: Pedals_APPS.PcbDoc	

Příloha VIII

Program mikrokontroléru

```c
/*
 * APPS Controller program
 *
 * University of West Bohemia in Pilsen
 * Czech Republic
 *
 * @author Adam Trojan
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include "pinout.h"
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/gpio.h"
#include "driverlib/can.h"
#include "driverlib/adc.h"
#include "driverlib/fpu.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/interrupt.h"
#include "driverlib/eeprom.h"
#include "driverlib/watchdog.h"

// The error routine that is called if the driver library encounters an error.
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
    while(1);
}
#endif

// SysTicks per seconds constant -> overflow every 1 ms
#define SYSTICKS_PER_SECOND 1000

// Definitions of pins
#define ST_LED1 GPIO_PORTF_BASE, GPIO_PIN_1
#define ST_LED2 GPIO_PORTF_BASE, GPIO_PIN_0
#define CAN_TX GPIO_PORTE_BASE, GPIO_PIN_5
#define CAN_RX GPIO_PORTE_BASE, GPIO_PIN_4
#define S1 GPIO_PORTE_BASE, GPIO_PIN_1
#define S2 GPIO_PORTB_BASE, GPIO_PIN_5
#define CAN_ADDR_BASE GPIO_PORTD_BASE
#define CAN_ADDR_MASK 0x00000007 // => (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2)
#define LOOP_STATE GPIO_PORTB_BASE, GPIO_PIN_4
#define CAL_BTN GPIO_PORTA_BASE, GPIO_PIN_5

/* Definitions of times in state machine */
#define INIT_CHECK_TIME 5000
#define CALIBRATION_ENTER_TIME 5000
#define CALIBRATION_END_TIME 5000
#define ERR_SIGNAL_INT 1500
// This time is important! It's the limit time for the implausibility check.
// Formula must react in 100ms when the signal is implausible, this time must have some
```

```
  reserve
63 #define READY_TIME 30
64 // This tolerance is important! It's the maximum deviation in percentage between the probes
   values given by rules! (max. 10% = 100)
65 #define DIFF_TOL 100
66
67 // Tolerance bands at the ends of the track in percent
68 #define MIN_MAX_TOL 10
69
70 // Size of buffer used to calibration in calibration mode
71 #define CAL_BUFF_SIZE 1000
72 // Time interval between samples in buffer when device is calibrating
73 #define CALIBRATION_TIME_INT 10
74
75 // The difference number of samples used to derive the signal trend when calibrating the
   device
76 #define DELTA_S 30
77
78 // Constants of minimum and maximum values of sensors (Hall probes),
79 // values from ADC must not go below (above) these levels if the sensors work properly
80 #define MIN_RANGE_S1 200
81 #define MIN_RANGE_S2 600
82 #define MAX_RANGE_S1 3000
83 #define MAX_RANGE_S2 3700
84
85 // Interval between sending the messages over CAN bus
86 #define CAN_MSG_INTERVAL 70
87 // Maximal interval between sending messages, after this time device will signalize error
88 #define CAN_MAX_MSG_INTERVAL 80 // MAX 100 ms!!
89
90
91 // Error codes definitions
92 #define NO_ERROR 0
93 #define ERR_SENSOR_SIGNAL 0x01
94 #define ERR_BAD_CALIBRATION 0x02
95 #define ERR_EEPROM 0x03
96 #define ERR_INTERLOCK 0x04
97
98 // LED macros
99 #define LED_ON 0x0000
100 #define LED_OFF 0xFFFF
101 #define LED_BLINK_500ms 500
102 #define LED_BLINK_100ms 100
103 #define LED_BLINK_250ms 250
104
105 //*****************************************************************************
106 // CAN definitions
107
108 // A flag to indicate that some transmission error occurred.
109 volatile bool CANErrFlag = 0;
110
111 // Can error status
112 uint32_t CANErrSts = 0;
113
114 volatile bool CANBusy = 0;
115
116 tCANMsgObject CANMessage;
117
118 uint32_t CANMsgData;
119 uint8_t *CANMsgDataP;
120
121 //*****************************************************************************
```

```
122 // ADC definitions
123
124 // ADC sequencer number
125 #define ADC_SEQ 3
126
127 // A flag to indicate that ADC value is ready to read
128 volatile bool ADCValueReady = 0;
129
130 //*****************************************************************************
131 // Other definitions
132
133 // Error code (no error = 0)
134 uint8_t errCode = 0x00;
135
136 // SysTick timer
137 // SysTick time counter variable:
138 uint64_t _ticks = 0;
139
140 void SysTickHandler(void)
141 {
142     _ticks++;
143 }
144
145 void WatchdogIntHandler(void)
146 {
147     // Clear the watchdog interrupt.
148     WatchdogIntClear(WATCHDOG0_BASE);
149 }
150
151 // This function is the interrupt handler for the CAN peripheral.  It checks
152 // for the cause of the interrupt.
153 void CANIntHandler(void)
154 {
155     uint32_t Status;
156
157     // Read the CAN interrupt status to find the cause of the interrupt
158     Status = CANIntStatus(CAN0_BASE, CAN_INT_STS_CAUSE);
159
160     // If the cause is a controller status interrupt, then get the status
161     if (Status == CAN_INT_INTID_STATUS)
162     {
163         // Read the controller status. This will return a field of status error bits that
    can indicate various errors.
164         // This function also clear interrupt
165         CANErrSts = CANStatusGet(CAN0_BASE, CAN_STS_CONTROL);
166
167         // If the interrupt status is not TX OK, then set the err flag for indication and
    later diagnosis
168         if((CANErrSts && ~CAN_STATUS_TXOK) != CAN_STATUS_TXOK)
169             CANErrFlag = 1;
170         else
171             CANErrFlag = 0;
172     }
173     // Check if the cause is message object 1, which what we are using for sending
    messages.
174     else if (Status == 1)
175     {
176         // Getting to this point means that the TX interrupt occurred on
177         // message object 1, and the message TX is complete.  Clear the
178         // message object interrupt.
179         CANIntClear(CAN0_BASE, 1);
180
```

```
181         // Since the message was sent, clear any error flags.
182         // ErrFlag = 0;
183 
184         // Clear busy flag
185         CANBusy = false;
186     }
187     // Otherwise, something unexpected caused the interrupt.  This should
188     // never happen.
189     else
190     {
191         // Spurious interrupt handling can go here.
192     }
193 }
194 
195 //Interrupt handler for both ADCs
196 void ADCIntHandler(void)
197 {
198     if ((ADCIntStatus(ADC0_BASE, ADC_SEQ, 1) == ADC_INT_SS3)
199             && (ADCIntStatus(ADC1_BASE, ADC_SEQ, 1) == ADC_INT_SS3))
200     {
201         // Set global flag
202         ADCValueReady = 1;
203 
204         // Clear the ADC interrupt flag.
205         ADCIntClear(ADC0_BASE, ADC_SEQ);
206         ADCIntClear(ADC1_BASE, ADC_SEQ);
207     }
208 }
209 
210 void InitializeSysTick(void)
211 {
212     // Setting period of SysTicks
213     SysTickPeriodSet(SysCtlClockGet() / SYSTICKS_PER_SECOND);
214 
215     // Register the handler for SysTick
216     SysTickIntRegister(SysTickHandler);
217 
218     // Enable SysTick interrupts
219     SysTickIntEnable();
220 
221     // Enable SysTick
222     SysTickEnable();
223 }
224 
225 void InitializeWatchdog(void)
226 {
227     // Enable the watchdog peripheral
228     SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);
229 
230     // Wait for the Watchdog 0 module to be ready.
231     while(!SysCtlPeripheralReady(SYSCTL_PERIPH_WDOG0)) {}
232 
233     // Enable the watchdog interrupt
234     IntEnable(INT_WATCHDOG);
235 
236     // Check to see if the registers are locked, and if so, unlock them
237     if(WatchdogLockState(WATCHDOG0_BASE))
238     {
239         WatchdogUnlock(WATCHDOG0_BASE);
240     }
241 
242     // Register the handler for interrupt
```

```
243     WatchdogIntRegister(WATCHDOG0_BASE, WatchdogIntHandler);
244
245     // Enable the watchdog interrupt
246     WatchdogIntEnable(WATCHDOG0_BASE);
247
248     // Setting the reload register value to sys clock, this means that the interrupt will
    be every second
249     // After the second interrupt, when the interrupt flag is not cleared, the MCU resets
    (after 2 sec)
250     WatchdogReloadSet(WATCHDOG0_BASE, SysCtlClockGet());
251
252     // Enable the processor reset if watchdog is not fed (feeding is clearing interrupt)
253     WatchdogResetEnable(WATCHDOG0_BASE);
254
255     // Prevent changes to the setup values
256     WatchdogLock(WATCHDOG0_BASE);
257
258     // Enable Watchdog
259     WatchdogEnable(WATCHDOG0_BASE);
260 }
261
262 void InitializeADCs(void)
263 {
264     // Set full speed on ADC
265     ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PIOSC | ADC_CLOCK_RATE_FULL, 1);
266
267     // Set Sequencers on ADC
268     ADCSequenceConfigure(ADC0_BASE, ADC_SEQ, ADC_TRIGGER_PROCESSOR, 0);
269     ADCSequenceConfigure(ADC1_BASE, ADC_SEQ, ADC_TRIGGER_PROCESSOR, 0);
270
271     // Configure sequencers steps
272     ADCSequenceStepConfigure(ADC0_BASE, ADC_SEQ, 0,
273                              ADC_CTL_CH2 | ADC_CTL_IE | ADC_CTL_END); // ADC0 read S1
274     ADCSequenceStepConfigure(ADC1_BASE, ADC_SEQ, 0,
275                              ADC_CTL_CH11 | ADC_CTL_IE | ADC_CTL_END); // ADC1 read S2
276
277     // Configure hardware averaging of 8 values
278     ADCHardwareOversampleConfigure(ADC0_BASE, 8);
279     ADCHardwareOversampleConfigure(ADC1_BASE, 8);
280
281     // Since sample sequence is now configured, it must be enabled.
282     ADCSequenceEnable(ADC0_BASE, ADC_SEQ);
283     ADCSequenceEnable(ADC1_BASE, ADC_SEQ);
284
285     // Setting the interrupts handler for ADCs
286     ADCIntRegister(ADC0_BASE, ADC_SEQ, ADCIntHandler);
287     ADCIntRegister(ADC1_BASE, ADC_SEQ, ADCIntHandler);
288     // Enable interrupts
289     ADCIntEnable(ADC0_BASE, ADC_SEQ);
290     ADCIntEnable(ADC1_BASE, ADC_SEQ);
291 }
292
293 void InitializeCAN(void)
294 {
295     // Set the message buffer pointer
296     CANMsgDataP = (uint8_t *) &CANMsgData;
297
298     // Initialize the CAN controller
299     CANInit(CAN0_BASE);
300
301     /*
302     // Enable test mode
```

```
303     CAN0_CTL_R |= CAN_CTL_TEST;
304
305     // Enable loopback mode for testing
306     CAN0_TST_R |= CAN_TST_LBACK;
307     */
308
309     // Set up the bit rate for the CAN bus: 500k
310     CANBitRateSet(CAN0_BASE, SysCtlClockGet(), 500000);
311
312     // Setting the interrupt handler for CAN bus controller
313     CANIntRegister(CAN0_BASE, CANIntHandler);
314
315     // Enable interrupts on the CAN peripheral
316     CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR | CAN_INT_STATUS);
317
318     // Enable the CAN for operation.
319     CANEnable(CAN0_BASE);
320
321     // Initialize the message object that will be used for sending CAN messages.
322     CANMsgData = 0;
323
324     // CAN Id - reading address from dip-switch
325     CANMessage.ui32MsgID = GPIOPinRead(CAN_ADDR_BASE, CAN_ADDR_MASK);
326     CANMessage.ui32MsgIDMask = 0;
327     CANMessage.ui32Flags = MSG_OBJ_TX_INT_ENABLE;
328     // DLC = 3
329     CANMessage.ui32MsgLen = 3; //sizeof(CANMsgDataP);
330     CANMessage.pui8MsgData = CANMsgDataP;
331 }
332
333 // Definition of structure used for output informations
334 typedef struct {
335     // Boolean to indicate the plausibility of signals
336     bool plausible;
337     // Difference between the signals from probes in percents
338     uint16_t difference;
339     // Shift of pedal in percent with one decimal point (0-1000)
340     uint16_t shiftPercentage;
341 } results;
342
343 // Definition of structure used for input informations
344 typedef struct {
345     // Maximum excitation of Hall probes
346     uint16_t probeMax[2];
347     // Minimum excitation of Hall probes
348     uint16_t probeMin[2];
349     // Constant for calculation
350     uint16_t calibratedConst[2];
351     // Measured values from ADC
352     uint32_t ADCValues[2];
353 } input;
354
355 bool SendData2CAN(results *dataStruct)
356 {
357     if (!CANBusy)
358     {
359         // CAN message - the first byte is errCode, the second and third bytes are the
    pedal shift percentage
360         CANMsgData = 0 | (errCode) | (dataStruct->shiftPercentage << 8);
361
362         // Send the CAN message using object number 1
363         CANMessageSet(CAN0_BASE, 1, &CANMessage, MSG_OBJ_TYPE_TX);
```

```
364
365         //Set the busy flag
366         CANBusy = true;
367
368         return true;
369     }
370     else
371     {
372         // CAN controller is busy - fail
373         return false;
374     }
375 }
376
377 void NullOutputStruct(results *dataStruct)
378 {
379     dataStruct->plausible = 0;
380     dataStruct->shiftPercentage = 0;
381     dataStruct->difference = 0;
382 }
383
384 bool CheckCalibratedValues(uint16_t *buffer, input *mainStruct)
385 {
386     uint16_t i, j, index;
387
388     // Find maximum
389     for (i = 1; i < CAL_BUFF_SIZE; i++)
390     {
391         for (j = 0; j < 2; j++)
392         {
393             index = j*CAL_BUFF_SIZE+i; // j*SIZE+i = base pointer+ this
394             if (buffer[index] > mainStruct->probeMax[j])
395                 mainStruct->probeMax[j] = buffer[index];
396         }
397     }
398
399     for(j=0;j<2;j++)
400     {
401         // Assign a maximum and a minimum with certain tolerances as the maximum and
    minimum
402         // Calculate the minimum plus some tolerance
403         mainStruct->probeMin[j] =
    mainStruct->probeMin[j]+(MIN_MAX_TOL*mainStruct->probeMin[j]/100);
404         // Calculate the maximum minus some tolerance
405         mainStruct->probeMax[j] = mainStruct->probeMax[j]-
    (MIN_MAX_TOL*mainStruct->probeMax[j]/100);
406     }
407
408     // Trend of signal
409     bool risingTrend = true;
410
411     // Last trend of signal
412     bool lastTrendWasRising = true;
413
414     short diff = 0;
415
416     // Check that the magnet does not move off the track - check stop settings
417     for(i=DELTA_S; i<CAL_BUFF_SIZE; i++)
418     {
419         for(j = 0; j<2; j++)
420         {
421             index = j*CAL_BUFF_SIZE+i;
422             diff = buffer[index] - buffer[index-DELTA_S];
```

```
423             // If the value is bigger than DELTA_Sth previous (default 30th) - rising track
    trend
424             if(diff > 100)
425             {
426                 // Then set the flag
427                 risingTrend = true;
428             }
429             else if(diff < -100)
430             {   // Decreasing trend, so set flag down
431                 risingTrend = false;
432             }
433             else
434             {
435                 // Trend is constant - nothing to do
436             }
437 
438             // When the trend of signal changes
439             if(lastTrendWasRising != risingTrend)
440             {
441                 // It must be somewhere near of maximum or minimum of the track, else there
    is some ambiguity
442                 // If new trend is rising, then the previous sample must be minimum (with
    some tolerance)
443                 if(risingTrend)
444                 {
445                     if(buffer[index-(DELTA_S/2)] > mainStruct->probeMin[j])
446                     {
447                         return false;
448                     }
449                 }
450                 // If new trend is decreasing, then the previous sample must be maximum
    (with some tolerance)
451                 else
452                 {
453                     if(buffer[index-(DELTA_S/2)] < mainStruct->probeMax[j])
454                     {
455                         return false;
456                     }
457                 }
458             }
459             lastTrendWasRising = risingTrend;
460         }
461     }
462 
463     // If the code reaches down here, the signals must be fine, so return true
464     return true;
465 }
466 
467 
468 results EvaluatePlausibility(input *inputStruct, results *resultStruct)
469 {
470     uint16_t shiftPercentage0, shiftPercentage1, difference;
471     // If the values are in ranges, then...
472     if ((inputStruct->ADCValues[0] >= MIN_RANGE_S1) && (inputStruct->ADCValues[0] <=
    MAX_RANGE_S1)
473             && (inputStruct->ADCValues[1] >= MIN_RANGE_S2) && (inputStruct->ADCValues[1] <=
    MAX_RANGE_S2))
474     {
475         // Calculate the percentages of shifting form both probes
476         shiftPercentage0 = 1000
477                 * (inputStruct->ADCValues[0] - inputStruct->probeMin[0])
478                 / inputStruct->calibratedConst[0];
```

```
479         shiftPercentage1 = 1000
480                 * (inputStruct->ADCValues[1] - inputStruct->probeMin[1])
481                 / inputStruct->calibratedConst[1];
482 
483         // Underflow protection
484         if(inputStruct->ADCValues[0] <= inputStruct->probeMin[0])
485             shiftPercentage0 = 0;
486 
487         if(inputStruct->ADCValues[1] <= inputStruct->probeMin[1])
488             shiftPercentage1 = 0;
489 
490         // Calculate the absolute difference between the probes values
491         if (shiftPercentage0 >= shiftPercentage1)
492             difference = shiftPercentage0 - shiftPercentage1;
493         else
494             difference = shiftPercentage1 - shiftPercentage0;
495 
496         // Are they in tolerance? If yes then we can return the value, if no then sets the
    errCode...
497         if (difference <= DIFF_TOL)
498         {
499             // If shift is higher, than 100% cut it to 100%
500             if(shiftPercentage1 > 1000)
501                 shiftPercentage1 = 1000;
502 
503             // The return value is from probe 1 (3V3 supply), because it is more sensitive
    with lower noise
504             errCode = NO_ERROR;
505             resultStruct->difference = difference;
506             resultStruct->shiftPercentage = shiftPercentage1;
507             resultStruct->plausible = true;
508         }
509         else
510         {
511             // Set the error code - big difference, implausible!
512             errCode = ERR_SENSOR_SIGNAL;
513             resultStruct->difference = difference;
514             resultStruct->shiftPercentage = 0;
515             resultStruct->plausible = false;
516         }
517     }
518     else
519     {
520         // Set the error code - not in ranges, implausible!
521         errCode = ERR_SENSOR_SIGNAL;
522         resultStruct->difference = 0xff;
523         resultStruct->shiftPercentage = 0;
524         resultStruct->plausible = false;
525     }
526 
527     return *resultStruct;
528 }
529 
530 int main(void)
531 {
532     /* ----- Initialization part ----- */
533     FPULazyStackingEnable();
534     FPUEnable();
535 
536     // Set the clocking to run from the PLL at 80MHz
537     SysCtlClockSet(
538     SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
```

```
539
540     // Setup the watchdog timer
541     InitializeWatchdog();
542
543     // Enable the EEPROM
544     SysCtlPeripheralEnable(SYSCTL_PERIPH_EEPROM0);
545
546     // Set pins to their functions
547     PortFunctionInit();
548
549     // Initialize SysTick
550     InitializeSysTick();
551
552     //Initialize ADC for reading signals
553     InitializeADCs();
554
555     //Initialize the CAN Bus for sending information to the control unit
556     InitializeCAN();
557
558     // Clear the interrupt status flag.  This is done to make sure the
559     // interrupt flag is cleared before we sample.
560     ADCIntClear(ADC0_BASE, ADC_SEQ);
561     ADCIntClear(ADC1_BASE, ADC_SEQ);
562
563     // Trigger first conversion
564     ADCProcessorTrigger(ADC0_BASE, ADC_SEQ | ADC_TRIGGER_WAIT);
565     ADCProcessorTrigger(ADC1_BASE, ADC_SEQ | ADC_TRIGGER_SIGNAL);
566
567     /* ----- End of Initialization part ----- */
568
569     /* ----- Main part ----- */
570
571     // State machine variables
572     typedef enum
573     {
574         START,
575         INIT_CHECK,
576         READY,
577         CALIBRATION_ENTER,
578         CALIBRATION_START,
579         CALIBRATION_SAMPLING,
580         CALIBRATION_END,
581         ERR,
582         ERR_SIGNAL
583     } states;
584
585     states state = START;
586     states nextState;
587
588     // State machine time interval variable
589     uint64_t SMTicks = 0;
590
591     // Initialization test boolean
592     bool initCheckFlag;
593
594     // CAN messages time interval variable
595     uint64_t CANTicks = 0;
596
597     // LED control variables
598     uint64_t blinkTicks[2] = {0, 0};
599     uint16_t LEDtimePeriod[2] = {0, 0};
600     uint8_t LEDState[2] = {0, 0};
```

```
601
602     // The time of the last update of values from the ADC
603     uint64_t ADCUpdateTicks = 0;
604
605     // The time of the last processing of the values from the ADC
606     uint64_t LastProcessTime = 0;
607
608     // Initialize the structures
609     input mainStruct;
610     results outputStruct;
611
612     errCode = 0;
613
614     uint16_t buffer[2][CAL_BUFF_SIZE];
615     uint16_t sampleNum = 0;
616
617     // Main loop
618     while (1)
619     {
620         // Firstly read the sensors values from ADCs
621         if (ADCValueReady)
622         {
623             // Copy the measured data to ADCValues array
624             ADCSequenceDataGet(ADC0_BASE, ADC_SEQ, &mainStruct.ADCValues[0]);
625             ADCSequenceDataGet(ADC1_BASE, ADC_SEQ, &mainStruct.ADCValues[1]);
626
627             // Set ADC flag down
628             ADCValueReady = false;
629
630             // Trigger next ADC conversion
631             ADCProcessorTrigger(ADC0_BASE, ADC_SEQ | ADC_TRIGGER_WAIT);
632             ADCProcessorTrigger(ADC1_BASE, ADC_SEQ | ADC_TRIGGER_SIGNAL);
633
634             // Set the time of the last update of values
635             ADCUpdateTicks = _ticks;
636         }
637
638         // State indicators LEDs controlling cycle
639         uint8_t i;
640
641         for (i = 0; i < 2; i++)
642         {
643             switch (LEDtimePeriod[i])
644             {
645             case LED_OFF:
646                 // LED should be OFF
647                 LEDState[i] = 0x00;
648                 break;
649             case LED_ON:
650                 // LED should be ON
651                 LEDState[i] = 0xff;
652                 break;
653             default:
654                 // If the value is in the range between 0 and 0xFFFF, it should be blinking
    periodically
655                 if (_ticks >= blinkTicks[i])
656                 {
657                     // Toggle state of LED
658                     LEDState[i] = ~LEDState[i];
659                     // Set the next time for toggling the LED
660                     blinkTicks[i] = _ticks + LEDtimePeriod[i];
661                 }
```

```
662             }
663         }
664 
665         GPIOPinWrite(ST_LED1, LEDState[0]);
666         GPIOPinWrite(ST_LED2, LEDState[1]);
667 
668         // State machine main block
669         switch (state)
670         {
671             case START:
672                 // Setting the LED indication
673                 LEDtimePeriod[0] = LED_OFF;
674                 LEDtimePeriod[1] = LED_OFF;
675 
676                 // // Null the output structure and error code
677                 NullOutputStruct(&outputStruct);
678 
679                 // Initialize the EEPROM
680                 if(EEPROMInit() != EEPROM_INIT_OK)
681                 {
682                     // Setting down the flag for check
683                     // This step is here, to give a chance of enter the calibration mode
    and repair data
684                     initCheckFlag = false;
685                     errCode = ERR_EEPROM;
686                 }
687                 else
688                 {
689                     // EEPROM is successfully initialized
690                     // Set the flag to true
691                     initCheckFlag = true;
692 
693                     // Create the buffer for the data from EEPROM
694                     uint64_t readBuffer;
695                     // Read 8 bytes from the EEPROM starting at address 0 and store them in
    the read buffer
696                     EEPROMRead((uint32_t*) &readBuffer, 0, 8);
697 
698                     // Split the data from the buffer into variables
699                     mainStruct.probeMin[0] = readBuffer;               // First two bytes
700                     mainStruct.probeMax[0] = (readBuffer >> 16);       // Second pair of
    bytes
701                     mainStruct.probeMin[1] = (readBuffer >> 32);       // Third pair of
    bytes
702                     mainStruct.probeMax[1] = (readBuffer >> 48);       // Fourth pair of
    bytes
703                 }
704 
705                 // Calculate the constants
706                 mainStruct.calibratedConst[0] = mainStruct.probeMax[0] -
    mainStruct.probeMin[0];
707                 mainStruct.calibratedConst[1] = mainStruct.probeMax[1] -
    mainStruct.probeMin[1];
708 
709                 // Setting the time limit for check sequence (init_check)
710                 SMTicks = _ticks + INIT_CHECK_TIME;
711                 nextState = INIT_CHECK;
712                 break;
713             case INIT_CHECK:
714                 // Setting the LED indication
715                 LEDtimePeriod[0] = LED_OFF;
716                 LEDtimePeriod[1] = LED_BLINK_250ms;
```

```
717
718                 // Check if the calibration button is pressed
719                 if(GPIOPinRead(CAL_BTN) == 0)
720                 {
721                     // Then go to commit mode for entering the calibration mode
722                     // Setting the time limit for commit to enter the calibration mode
723                     SMTicks = _ticks + CALIBRATION_ENTER_TIME;
724                     nextState = CALIBRATION_ENTER;
725                     break;
726                 }
727
728                 // If the new measured values are available and the flag is true, then...
729                 if ((ADCUpdateTicks > LastProcessTime) && initCheckFlag)
730                 {
731                     // Evaluate the plausibility of measured values
732                     EvaluatePlausibility(&mainStruct, &outputStruct);
733
734
735                     // If the shift percentage is not null immediately after start,
736                     // it could be bad calibration there, so go to error state.
737                     // Note that the pedal must be left released for a few seconds after
    start of device!
738                     if (outputStruct.shiftPercentage > 0)
739                     {
740                         NullOutputStruct(&outputStruct);
741                         errCode = ERR_BAD_CALIBRATION;
742                         // If only once in this check interval shiftPercentage != 0, then
    flag goes down for going to err mode
743                         initCheckFlag = false;
744                     }
745
746                     LastProcessTime = ADCUpdateTicks;
747                 }
748
749                 if(_ticks >= SMTicks)
750                 {
751                     if(!initCheckFlag)
752                     {
753                         nextState = ERR;
754                     }
755                     else
756                     {
757                         // If the sequence gone well, after the check time limit go to the
    READY state
758                         SMTicks = _ticks + READY_TIME;
759                         nextState = READY;
760                     }
761                 }
762
763
764                 break;
765             case READY:
766                 // State READY - everything is fine - MCU is happy
767                 // Setting the LED indication
768                 LEDtimePeriod[1] = LED_OFF;
769
770
771                 if(ADCUpdateTicks > LastProcessTime)
772                 {
773                     EvaluatePlausibility(&mainStruct, &outputStruct);
774
775                     // Are the values plausible? If yes then we can set the new time limit
```

```
776                     if(outputStruct.plausible)
777                     {
778                         // Setting of LED blinking period for indication of non-zero shift
779                         if(outputStruct.shiftPercentage == 0)
780                             LEDtimePeriod[0] = LED_BLINK_500ms;
781                         else
782                             LEDtimePeriod[0] = LED_BLINK_100ms;
783 
784                         SMTicks = _ticks + READY_TIME;
785                     }
786 
787                     LastProcessTime = ADCUpdateTicks;
788                 }
789 
790                 // If the time limit is gone - Signal is implausible and go to error mode
791                 if(_ticks >= SMTicks)
792                 {
793                     nextState = ERR;
794                 }
795 
796                 break;
797             case CALIBRATION_ENTER:
798                 // Setting the LED indication
799                 LEDtimePeriod[0] = LED_OFF;
800                 LEDtimePeriod[1] = LED_BLINK_100ms;
801 
802                 // If the calibration button is released while commit mode, then cancel the
    attempt and go back to start
803                 if(GPIOPinRead(CAL_BTN) != 0)
804                 {
805                     nextState = START;
806                     break;
807                 }
808 
809                 // If the button is pressed all time during commit mode, then go to
    calibration mode
810                 if(_ticks >= SMTicks)
811                 {
812                     nextState = CALIBRATION_START;
813                 }
814 
815                 break;
816             case CALIBRATION_START:
817                 // Setting the LED indication
818                 LEDtimePeriod[0] = LED_OFF;
819                 LEDtimePeriod[1] = LED_ON;
820 
821                 sampleNum = 0;
822 
823                 if(ADCUpdateTicks > LastProcessTime)
824                 {
825                     // In first phase of calibration, the pedal must be released, so this
    values must be the minimum
826                     mainStruct.probeMin[0] = mainStruct.ADCValues[0];
827                     mainStruct.probeMin[1] = mainStruct.ADCValues[1];
828 
829                     // Set the those values also as maximum
830                     mainStruct.probeMax[0] = mainStruct.ADCValues[0];
831                     mainStruct.probeMax[1] = mainStruct.ADCValues[1];
832 
833                     // Now go to calibration phase two
834                     nextState = CALIBRATION_SAMPLING;
```

```
835                     LastProcessTime = ADCUpdateTicks;
836                 }
837                 break;
838             case CALIBRATION_SAMPLING:
839                 if (sampleNum < CAL_BUFF_SIZE)
840                 {
841                     if ((_ticks >= SMTicks) && (ADCUpdateTicks > LastProcessTime))
842                     {
843                         // Save the values to the buffers
844                         buffer[0][sampleNum] = mainStruct.ADCValues[0];
845                         buffer[1][sampleNum] = mainStruct.ADCValues[1];
846 
847                         // Increment variable and set the new time of saving into the
    buffers
848                         sampleNum++;
849                         SMTicks = _ticks + CALIBRATION_TIME_INT;
850                         LastProcessTime = ADCUpdateTicks;
851                     }
852                 }
853                 else
854                 {
855                     nextState = CALIBRATION_END;
856                 }
857                 break;
858             case CALIBRATION_END:
859                 // Condition for locking state machine in this state after calibration
860                 if (sampleNum == 0)
861                 {
862                     LEDtimePeriod[0] = LED_OFF;
863                     LEDtimePeriod[1] = LED_OFF;
864                     NullOutputStruct(&outputStruct);
865                     break;
866                 }
867 
868                 // After sampling, we must check the right setting of the stops at the ends
    of the track
869                 if (CheckCalibratedValues((uint16_t*) &buffer, &mainStruct))
870                 {
871                     // All right, so we can save the calibrated values to memory
872                     // Prepare the values to the write buffer in correct order for later
    programming into memory
873                     uint64_t writeBuffer = mainStruct.probeMin[0]
874                             | (mainStruct.probeMax[0] << 16)
875                             | ((uint64_t) mainStruct.probeMin[1] << 32)
876                             | ((uint64_t) mainStruct.probeMax[1] << 48);
877 
878                     // Program the data to EEPROM at address 0. Length is 8 bytes.
879                     if (EEPROMProgram((uint32_t*) &writeBuffer, 0, 8))
880                     {
881                         // If there is some error, go to error state
882                         nextState = ERR;
883                         errCode = ERR_EEPROM;
884                         break;
885                     }
886                     // This will lock the state machine in loop - waiting for reset
887                     nextState = CALIBRATION_END;
888                     sampleNum = 0;
889                 }
890                 else
891                 {
892                     // If check result is false, then go to the error state
893                     errCode = ERR_BAD_CALIBRATION;
```

```
894                     nextState = ERR;
895                 }
896                 break;
897             case ERR:
898                 LEDtimePeriod[0] = LED_ON;
899                 LEDtimePeriod[1] = LED_OFF;
900 
901                 NullOutputStruct(&outputStruct);
902 
903                 if(_ticks >= SMTicks)
904                 {
905                     // This time limit ensures how many times the LED flashes (LED blinking
    period = 250 ms)
906                     // 50ms reserve is there to prevent additional blink
907                     SMTicks = _ticks + ((errCode*2*250)-50);
908                     nextState = ERR_SIGNAL;
909                 }
910                 break;
911             case ERR_SIGNAL:
912                 // The LED indicates the error number by the number of flashes
913                 LEDtimePeriod[1] = LED_BLINK_250ms;
914 
915                 NullOutputStruct(&outputStruct);
916 
917                 if(_ticks >= SMTicks)
918                 {
919                     nextState = ERR;
920                     SMTicks = _ticks + ERR_SIGNAL_INT;
921                 }
922                 break;
923             default:
924                 nextState = START;
925         }
926 
927         // Interlock loop check
928         if(GPIOPinRead(LOOP_STATE) && errCode != ERR_INTERLOCK)
929         {
930             errCode = ERR_INTERLOCK;
931             nextState = ERR;
932         }
933 
934         // CAN Messages sending
935         if(_ticks >= CANTicks)
936         {
937             // If the message was sent, then we can set the new time limit,
938             // if it fails, then the new attempts to send another message will be in every
    cycle
939             // until the maximal time limit between messages runs out or message will be
    sent
940             if(SendData2CAN(&outputStruct))
941             {
942                 // The next message will be send in this time:
943                 CANTicks = _ticks + CAN_MSG_INTERVAL;
944             }
945             else
946             {
947                 if(((_ticks - CANTicks + CAN_MSG_INTERVAL) >= CAN_MAX_MSG_INTERVAL) ||
    CANErrFlag)
948                 {
949                     LEDtimePeriod[0] = LED_ON; // Signalize the CAN error - only LED -
    state of the state machine does not change
950                     LEDtimePeriod[1] = LED_OFF;
```

```
951                 }
952             }
953         }
954 
955         state = nextState;
956 
957     }
958 }
959 
```