Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Bakalářská práce

Generátor DCC signálu pro modelovou železnici

Autor práce: Filip Zvonař
Vedoucí práce: Ing. Ondřej Lufinka

Plzeň 2018

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Filip ZVONAŘ**
Osobní číslo:	**E15B0292P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Název tématu:	**Generátor DCC signálu pro modelovou železnici**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

Navrhněte jednotku pro generování DCC signálu pro modelovou železnici. Jednotka musí splňovat standardizované rozměry jednotek pro správu kolejiště. Dále musí obsahovat rozhraní sběrnice CAN vedené přes UTP kabel pro její komunikaci se zbytkem systému. Protokol DCC je standardizovaný k řízení modelových lokomotiv a tento standard je také nutné dodržet.

1. Popište systém modelového kolejiště (blokové schéma apod.) a princip sběrnice CAN.
2. Navrhněte koncept zařízení. Prostudujte nabídku dobře dostupných součástek a konkretizujte zapojení. Při volbě procesorové platformy se zaměřte na nabídku firmy NXP.
3. Realizujte funkční vzorek po hardwarové stránce.
4. Naprogramujte firmware procesoru pro plnění požadovaných funkcí jednotky.
5. Ověřte funkčnost jednotky v systému modelového kolejiště a doplňte práci o její podrobnou specifikaci, která usnadní případné budoucí modifikace.

Rozsah grafických prací:	**podle doporučení vedoucího**
Rozsah kvalifikační práce:	**30 - 40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí bakalářské práce:	**Ing. Ondřej Lufinka** Katedra aplikované elektroniky a telekomunikací
Datum zadání bakalářské práce:	**10. října 2017**
Termín odevzdání bakalářské práce:	**7. června 2018**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 10. října 2017

Abstrakt

Tato bakalářská práce se zabývá návrhem a realizací generátoru DCC signálu, který umožňuje řízení lokomotiv na modelové železnici na FEL ZČU. V teoretické části této práce jsou rozebrány funkční bloky kolejiště, protokol sběrnice CAN a protokol DCC, se kterými jednotka generátoru pracuje. V praktické části této práce se řeší návrh jednotky, výroba desky plošných spojů a její osazení. Dále se řeší vývoj firmware, způsob generování DCC příkazů pomocí mikrokontroléru, zajištění bezpečnosti a přijímání příkazů ze sběrnice CAN. V závěrečné části práce je popsán způsob použití jednotky pro řízení lokomotiv jak pomocí CAN, tak ručně pomocí tlačítek jednotky. Jednotka je funkční. Do budoucna přichází v úvahu drobná vylepšení jejího firmware.

Klíčová slova

CAN, DCC, embedded systém, modelové kolejiště, NXP, řízení lokomotiv, watchdog.

Abstract

Zvonař, Filip. *The proposal of a DCC signal generator for a model railroad* [*Generátor DCC signálu pro modelovou železnici*]. Pilsen, 2018. Bachelor thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Applied Electronics and Telecommunications. Supervisor: Ondřej Lufinka

---

This bachelor thesis is focused on the design and making of a DCC generator for the control of model trains on a model railroad which is at site of the Faculty of Electrical Engineering of the University of West Bohemia. In the theoretical part of this thesis functional blocks of the model railroad, a CAN bus and the DCC protocol are described. The generator unit operates with both the CAN bus and the DCC protocol. In the practical part of this thesis the design of the unit, the making of the PCB and its soldering are described. Further on the thesis focuses on the firmware, the generation of DCC commands by a microcontroller, receiving commands from CAN bus and taking basic safety measures. At the end of this thesis there is a basic manual on how to control the trains with this unit. The unit is operational although there will probably be some firmware updates made in the future.

Keywords

CAN, DCC, embedded system, model railroad, NXP, train control, watchdog timer.

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 5. června 2018

Filip Zvonař

.....

Podpis

Obsah

Seznam obrázků

Seznam symbolů a zkratek

ACR	Acceptance Code Register, registr příjmového kódu
AMR	Acceptance Mask Register, registr příjmové masky
BOM	Bill of material, seznam materiálu
CAN	Controller Area Network
DCC	Digital Command Control
DPS	Deska Plošných Spojů
FTDI	Future Technology Devices International
I/O	Input / Output, vstup / výstup
I2C	Inter-Integrated Circuit
ID	Identifikátor
LED	Light Emitting Diode, svítivá dioda
log.	logická hodnota
LQFP	Low profile Quad Flat Package, nízkoprofilové čtvercové ploché pouzdro
NMRA	National Model Railroad Association
PC	Personal Computer, osobní počítač
PWM	Pulse Width Modulation, pulzně šířková modulace
RAM	Random Access Memory, paměť s libovolným přístupem
RGB	Red Green Blue, červená-zelená-modrá
SCI	Serial Communications Interface, rozhraní sériových komunikací
SMD	Surface Mount Device, povrchově osazené zařízení
SPI	Serial Peripheral Interface, sériové periferní rozhraní
THT	Through Hole Technology, technologie osazování součástek s drátovými vývody
UART	Universal Asynchronous Receiver and Transmitter, druh sériového rozhraní
USB	Universal Serial Bus, univerzální sériová sběrnice
UTP	Unshielded Twisted Pair, nestíněná kroucené dvojlinka
XOR	Exclusive or, logická operace nazývaná exklusivní disjunkce

1

Úvod

Pro úvod této práce je dobré nejprve nastínit motivaci jejího vzniku. Jako se člověk neustále učí a patří to k životu, stejně tak k životu patří i hra. V průniku těchto dvou činností se nachází něco, čemu někteří říkají škola hrou. Je skvělé, když člověka baví hrát si s něčím, na čem se může naučit dovednosti, které jsou ceněné i do budoucí praxe. Takovým něčím může být právě třeba model železnice. Je zajímavé, jak fascinující jsou věci v drobném měřítku. Díky modelu železnice může být zábavné i studium digitálního řízení.

Právě takové modelové kolejiště se nachází v 5. patře laboratoří Fakulty elektrotechnické ZČU v Plzni. Tento model je již několik let nedokončený, což může být bráno jako negativum, ale i jako pozitivum. Jistě je nepříjemné, když se velký projekt nedaří dokončit, na druhou stranu se nyní nově na projektu mohou praktickým způsobem podílet studenti. Na kolejišti je možné se učit implementovat aktuální technologie, které se v dnešní době ve světě používají, jako je například využití sběrnice CAN a decentralizace řídících systémů. Podobné systémy s velkým množstvím jednotek s vlastními mikrokontroléry jsou stále používanější.

Záměrem tohoto modelové kolejiště je vytvořit počítačem řízený železniční systém. Další z konkrétních vizí je následné propojení řízení kolejiště s internetovou stránkou. Na ní bude možné kolejiště řídit a zároveň sledovat jeho stav s možností sledovat i živý přenos videa přímo z kabiny vlaku. Takový systém je vskutku rozsáhlým projektem a zdaleka se nevejde do rozsahu jedné bakalářské práce.

Realizace celkového řízení je možná i s použitím komerčně dostupných produktů. Takové systémy obvykle zahrnují kompletní vybavení, od buzení kolejí až po řídící software běžící na počítači. Na trhu není možné pořídit takovou jednotku, která by zapadala do systému řízení navrženého pro tuto železnici. Navržený systém řízení této železnice sice využívá standardy pro řízení modelových lokomotiv, což umožňuje použití komerčně vyráběných lokomotiv s DCC dekodéry. Vyšší úroveň řízení je však již navržena s použitím sběrnice CAN, která pro modelové železnice není běžná.

Z celého systému kolejiště se tato práce zabývá jen jedním dílčím celkem. Jejím cílem je navrhnout a vyrobit jednu jednotku, jeden funkční blok systému. Jeho úkolem je ge-

nerovat signál do kolejí pro ovládání všech lokomotiv na kolejišti. Hlavní řešený problém je způsob komunikace s centrálním řízením po sběrnici CAN a generování signálu DCC pro lokomotivní dekodéry. Signál DCC na výstupu generátoru není silový a pokračuje do jednotek zesilovačů DCC, které teprve budí koleje. Zvoleným postupem bylo vyrobit desku plošných spojů s mikrokontrolérem a potřebnými rozhraními a následně jednotku naprogramovat pro zvolený účel.

V první teoretické části práce je rozebrán systém kolejiště a jeho řízení pomocí sběrnice CAN. Dále je stručně popsáno způsob digitálního řízení modelových železnic pomocí protokolu DCC. To vytváří základ pro praktickou část práce. V té se řeší návrh jednotky, výběr procesorové platformy a následně výroba a osazení desky plošných spojů a oživení jednotky. Ke konci praktické části je vysvětlen způsob práce s jednotkou pro umožnění jejího budoucího využití. Je zde popsána komunikace jednotky s řízením CAN a princip jejího fungování.

2

Popis systému modelového kolejiště

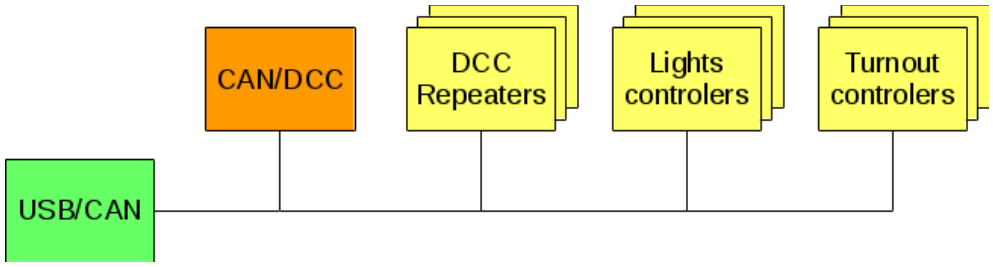
Pro bližší porozumění významu této práce, která tvoří jen dílčí prvek systému kolejiště, je důležité si udělat širší obraz o vlastnostech a funkci celého systému.

Kolejiště je navrženo tak, aby bylo řízeno po sběrnici CAN (Controller Area Network), která je již ověřeným standardem v řízení a komunikaci mezi mikrokontroléry. CAN umožňuje i relativně snadné připojení počítače jako centrálního řízení celého systému.

Všechny jednotlivé funkční bloky kolejiště jsou spojené pomocí kabelu UTP (Unshielded twisted pair). V něm jsou 4 kroucené páry vodičů. Jeden pár je vyhrazený pro sběrnici CAN, druhý pro signál DCC (Digital Command Control) a zbylé dva páry slouží pro rozvod stejnosměrného napájení.

Ne každá jednotka využívá všechny páry, nicméně pro zjednodušení kabeláže se při propojování vždy zapojují všechny. K usnadnění propojování desek, k propojení bez kabelu, slouží hřebínkový konektor (klasický header) na normované pozici na každé desce. Na tento konektor jsou vyvedeny všechny vodiče z konektoru UTP a umožňuje tak propojování desek instalovaných na sebe [1].

2.1 Blokové schéma kolejiště



Obr. 2.1: Blokové schéma koncepce řízení TT vlaků [1]

2.1.1 Blok USB/CAN: Centrální řízení PC

O řízení kolejiště se při běžném provozu stará program běžící na PC. Program disponuje srozumitelným uživatelským prostředím, které slouží pro sledování stavu kolejiště a jeho jednotlivých funkčních jednotek. Program umožňuje manuální ovládání jednotlivých prvků kolejiště (např. ovládání rychlosti jednotlivé lokomotivy) a také nastavování a automatické spouštění komplexnějších sekvencí (např. pravidelné trasy vlakových spojení).

Program sleduje aktivní stav jednotek pomocí zpráv generovaných jejich watchdog časovačem. Tento časovač při správném chodu jednotky pravidelně odesílá předem definovanou zprávu. Řídící program pak při výpadku některé z jednotek musí vyhodnotit chybový stav a provést patřičné kroky pro zajištění maximální bezpečnosti a případný reset jednotek, pokud byl problém odstraněn.

2.1.2 Blok CAN/DCC: Generátor signálu DCC

Generátor DCC přijímá příkazy určené do kolejí v podobě CAN zpráv a z nich vytváří signál DCC. K jednotlivým příkazům přidává redundanci, příkazy ukládá a neustále opakuje.

Jednotka po rozhraní CAN odesílá watchdog zprávy centrálnímu řízení. Centrální řízení při výpadku jednotky vypne v bloku DCC Repeater buzení kolejí, aby se lokomotivy nerozjely na stejnosměrné napětí.

Watchdog časovač této jednotky také hlídá, jestli pravidelně přichází CAN zprávy pro DCC a v případě výpadku nastaví rychlosti všech lokomotiv v paměti příkazů na hodnotu 0.

Rozměry jednotky, rozmístění konektorů a montážních otvorů jsou jednotné s ostatními jednotkami.

2.1.3 Blok DCC Repeater: Zesilovač signálu DCC

Tento blok sestává z množství jednotek, z nichž každá budí až 8 úseků kolejí signálem DCC, který jednotky přijímají z generátoru DCC a zesilují ho pro napájení lokomotiv. Jednotky zároveň sledují odběr proudu na každém úseku, což slouží pro detekci obsazenosti úseků. Tyto hodnoty jsou odesílány centrálnímu řízení po rozhraní CAN.

Buzení kolejí zajišťuje step-down měnič napětí s H-můstkem, který je řízený přímo signálem DCC. To umožňuje funkci buzení i při zaseknutí programu v mikrokontroléru jednotky.

Měření proudu je realizováno lineárně pomocí sériově zařazeného odporu, zesilovače a AD převodníku s odpovídající rychlostí a přesností. Tento systém je schopen detekovat odběr již od řádu mA.

Rozměry jednotek, rozmístění konektorů a montážních otvorů jsou jednotné s ostatními jednotkami.

2.1.4 Blok Light controlers: Řízení semaforů

Každá z jednotek řízení semaforů přijímá příkazy z CAN rozhraní, podle kterých budí osvětlení a semafory na vymezené části kolejiště.

Rozměry jednotek, rozmístění konektorů a montážních otvorů je jednotné s ostatními jednotkami.

2.1.5 Blok Turnout controlers: Řízení výhybek

Budiče výhybek přijímají příkazy z CAN rozhraní, podle kterých přehazují výhybky.

Do tohoto bloku také spadá řídící jednotka točny, která obdobně řídí pohyb točny.

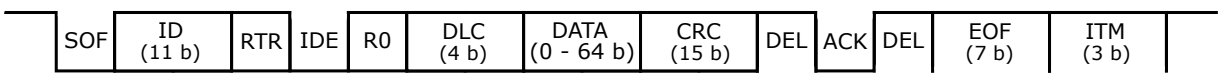
Rozměry jednotek, rozmístění konektorů a montážních otvorů jsou jednotné s ostatními jednotkami.

2.2 Princip sběrnice CAN

2.2.1 Komunikační protokol CAN

CAN (Controller Area Network), v našem případě standardní formát CAN 2.0 B, je sériový komunikační protokol, který se využívá ke komunikaci jednotek po sběrnici. Původně byl tento protokol vyvinut pro automobilový průmysl, kde bylo potřeba, aby byl spolehlivý, ne příliš složitý. Bylo potřeba, aby byl odolný vůči elektromagnetickému rušení, dokázal propojit vysoké množství jednotek a zároveň poskytoval dostatečně vysokou přenosovou rychlost. Komunikace přes CAN probíhá formou krátkých zpráv obsahujících 0-8 datových bajtů. Tyto zprávy může posílat jakákoliv jednotka. Při vysílání zprávy ostatní jednotky mlčí, zprávu přijímají a buď ji zpracovávají nebo ignorují. Jedná se tedy o protokol typu multi-master, kde kterákoliv jednotka může řídit chování jiných jednotek. [2]

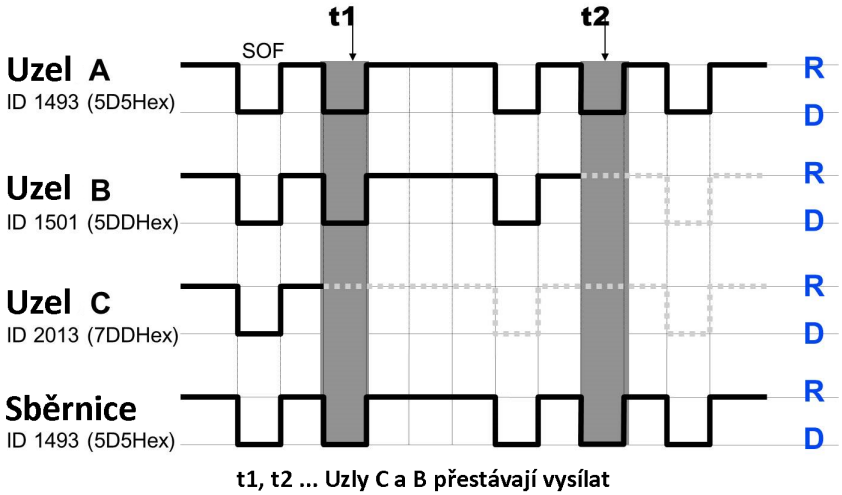
Na sběrnici CAN je v klidovém stavu (bus free) stav log. 1, což je tzv. recesivní stav. Tento stav může kterákoliv jednotka aktivně přepnout do log. 0, což je tzv. dominantní stav, který již jiná jednotka zpět do stavu log. 1 přepnout nemůže. Logická hodnota bitu na sběrnici je tedy logickým součinem výstupů všech připojených jednotek. Toto se uplatní, když nedopatřením začnou ve stejný okamžik vysílat dvě jednotky naráz. Každá z vysílajících jednotek porovnává vysílanou logickou hodnotu se stavem na sběrnici a v případě neshody ukončí vysílání. Ve vysílání pak pokračuje pouze jednotka, která v ten okamžik vysílala dominantní bit.



Obr. 2.2: Zpráva protokolu CAN

Odesílaná zpráva sestává z několika částí. První se vždy odešle dominantní bit označený SOF (Start Of Frame), neboli začátek rámce. Následuje 11 bitů, kterým říkáme ID

neboli identifikátor zprávy. ID slouží pro identifikaci zprávy a zároveň určují její prioritu v případě, že začne vysílat více jednotek naráz. Při současném vysílání přestanou vysílat všechny jednotky, kromě té, jejíž zpráva má ze všech vysílaných nejnižší hodnotu ID. Nižší čísla mají vyšší prioritu, protože log. 1 je recesivním stavem (viz obrázek 2.3).



Obr. 2.3: Kolize zpráv s různým ID [5]

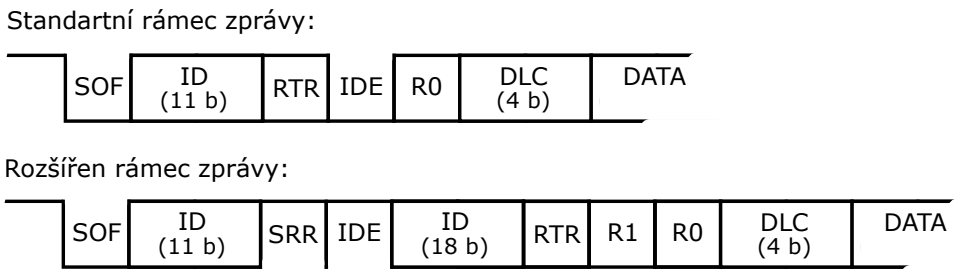
Po identifikátoru následuje bit označený RTR (Remote Transmission Request). Tento bit určuje, jestli se jedná o datovou zprávu (log. 0), nebo žádost o data (log. 1). Bit označený IDE (Identifier Extension) určuje, zda má zpráva standardní (log. 0) nebo rozšířený (log. 1) formát rámce. Následuje rezervovaný bit označený R0 a po něm 4 bity označené DLC (Data Length Code). Číslo uložené v bitech DLC určuje počet následujících datových bajtů ve zprávě. Po datech je ve zprávě zařazeno 15 bitů označených CRC (Cyclic Redundancy Check). CRC slouží pro kontrolu chyb ve zprávě. Po bitech CRC následuje recesivní bit označený DEL (Delimiter), který musí být recesivní (log. 1) pro oddělení následujícího bitu označeného ACK (Acknowledge). Bit ACK slouží k potvrzení bezchybného odeslání zprávy. Vysílající jednotka vždy posílá bit ACK recesivní (log. 1) a ostatní jednotky potvrzují bezchybný příjem zprávy tím, že vyšlou dominantní bit (log. 0). Jednotky zprávu kontrolují vždy, i když zpráva nemusí být určena pro ně. To slouží jako zpětná vazba pro vysílající jednotku. Bit ACK je oddělen dalším recesivním bitem DEL a celá zpráva je ukončena 7 bity označenými EOF (End Of Frame) s hodnotou log. 1 a třemi bity označenými ITM (Intermission) s hodnotou log. 1, které tvoří jakousi pomlku, než může začít další rámec zprávy.

2.2.1.1 Rozšířený rámec zprávy

Identifikátor CAN zprávy může být rozšířen až na celkovou délku 29 bitů, použitím rozšířeného rámce zprávy. Použití rozšířeného rámce se označuje bitem IDE (log. 1), po kterém následuje dalších 18 bitů identifikátoru. V rozšířeném rámci se také přesouvá bit označený RTR až za poslední bit ID, kde nahradí bit označený R0. Na jeho původním místě je pak

místo bitu RTR bit označený SRR (Substitute Remote Request) v recesivním stavu (log. 1). Bit SRR zajišťuje, aby v případě kolize zpráv standardního a rozšířeného rámce se stejným 11 bitovým ID, měla přednost zpráva se standardním rámcem.

Zbytek rozšířeného rámce je shodný se standardním rámcem.



Obr. 2.4: Rozdíl mezi standartním a rozšířeným rámcem

2.2.1.2 Synchronizace

U rozhraní CAN se používá bitová synchronizace. Každý přijímač je vybaven fázovým závěsem, který se průběžně ladí s každou hranou ve zprávě na sběrnici. Aby se zajistilo, že při přenosu nebude ve zprávě dlouhou dobu chybět hrana, vkládá se do zprávy vždy po pěti shodných bitech jeden bit s opačnou hodnotou. Tomuto postupu se říká bit stuffing.

2.2.1.3 Zpracování chyb

Chyby na sběrnici CAN jsou ošetřeny komplexním systémem pravidel pro zajištění maximální spolehlivosti. Jednotky, které detekují chybu, mohou odeslat zprávu o chybě, tzv. error frame. Error frame sestává buď z několika dominantních, nebo recesivních bitů a následujících několika recesivních bitů jako mezera před další zprávou. Chybové zprávy s dominantními bity smí jednotky odesílat jen omezeně.

Chyby můžou být následující:

- `Chyba v bitu` – neshoda vzorků při vícenásobném vzrokování
- `Chyba vkládání bitů` – chybějící synchronizační hrany
- `Chyba CRC` – kontrola chyb pomocí CRC nevychází
- `Chyba rámce` – dominantní bity v částech DEL, EOF nebo ITM
- `Chybící potvrzení` – bit ACK zůstal recesivní [3]

Aby se zabránilo tomu, že by některá jednotka zahltila sběrnici chybovými zprávami, má každá jednotka dva čítače chyb, čítač chyb příjmu a čítač chyb odesílání. Tyto čítače se inkrementují a dekrementují podle sofistikovaných pravidel. Tato pravidla zajišťují, aby čítače vadné jednotky dosáhly stanovených hodnot rychleji, než u funkčních jednotek.

Chybující jednotka pak nejprve ztratí právo vysílat chybové zprávy z dominantních bitů a následně ztratí právo vysílat jakékoliv zprávy.

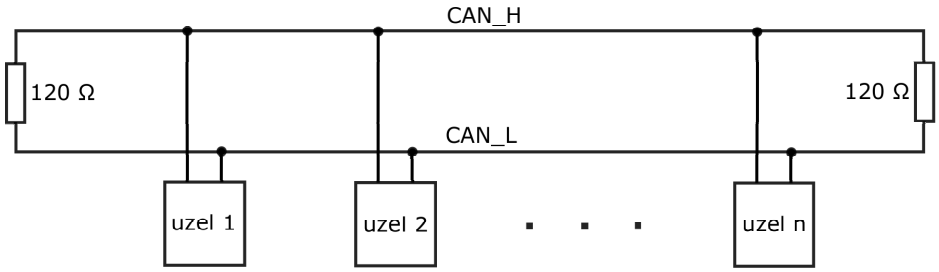
Tyto čítače se dekrementují úspěšným příjmem. Díky tomu se například jednotka, která byla dočasně silně zarušená a ztratila právo vysílat, může postupně vrátit zpět do hry.

2.2.1.4 Zprávy o přetížení

Podobně jako zpráva o chybě, existuje v protokolu CAN i zpráva o přetížení. Tato zpráva vypadá obdobně jako zpráva o chybě, ale odesílá se mezi ostatními rámci, do kterých tudíž nezasahuje. Tuto zprávu odesílá jednotka, která je přetížená a potřebuje více času na zpracování, pro oddálení vysílání následujících zpráv. [2]

2.2.2 Fyzická vrstva sběrnice CAN

Sběrnice pro CAN musí zajistit realizaci logického součinu, tedy aby se na sběrnici vyskytoval v klidu stav recesivní (log. 1), a kdykoliv kterákoliv z jednotek vysílá dominantní bit (log. 0), se již tento stav ovlivnit nedal.

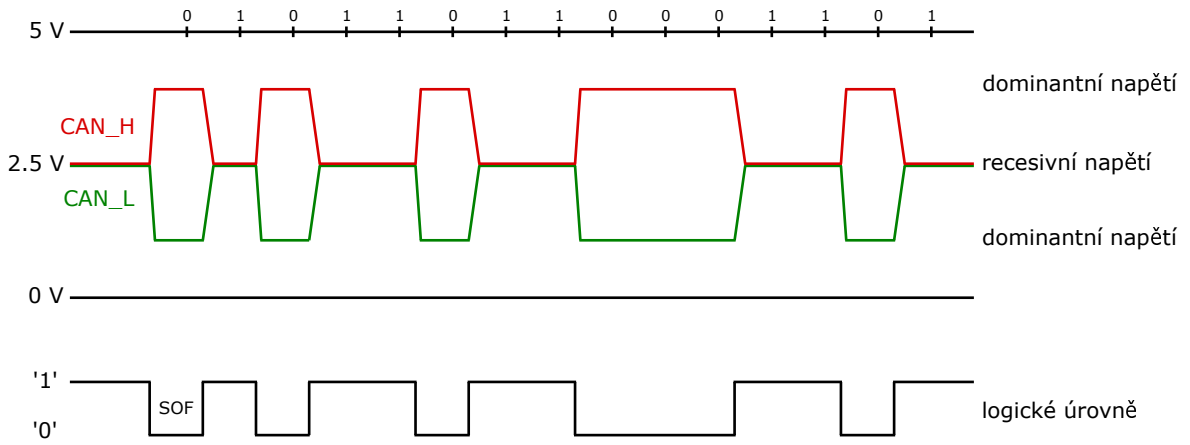


Obr. 2.5: Síť sběrnice CAN

Pro realizaci takové sběrnice, včetně vysílacích budičů a přijímačů, se nejčastěji používá diferenciální sběrnice definovaná normou ISO 11898. Tato norma definuje také elektrické vlastnosti budičů, princip časování, kódování jednotlivých bitů a synchronizaci přenosu zpráv. Sběrnice sestává ze dvou vodičů, typicky označených jako CAN_H a CAN_L. Tyto dva vodiče jsou na koncích spojeny přes zakončovací odpory o velikosti 120 Ω, tzv. terminátory. Ty zajišťují impedanční přizpůsobení, a tím i eliminaci odrazů na vedení.

Hodnota bitu na sběrnici je definována rozdílem napětí na těchto vodičích (viz obrázek 2.5). Recesivnímu stavu (log. 1) odpovídá rozdíl napětí blízký nule a dominantnímu stavu (log. 0) odpovídá rozdíl napětí větší než 1,2 V, typicky 2 V. Aby se rozdíl napětí automaticky vracel k nule, musí být na koncích vedení zakončovací odpory. Buzení vodičů je napájeno stejnosměrným napětím 5 V. Při recesivním stavu je na obou vodičích CAN_H i CAN_L napětí 2,5 V a při dominantním stavu je typicky na CAN_H napětí 3,5 V a na CAN_L napětí 1,5 V (viz obrázek 2.6).

Vodiče jsou nejčastěji vedeny v podobě kroucené dvojlinky, což zajišťuje vysokou odolnost vůči elektromagnetickému rušení. Protože se vodiče na svých místech pravidelně stří-



Obr. 2.6: Úrovně napětí na vodičích CAN_H a CAN_L [4]

dají, indukuje se do obou stejné indukované napětí. Díky tomu není rozdíl napětí v uzlech na jednotlivých vodičích ovlivněn rušením.

Přenosová rychlost u rozhraní CAN není pevně daná, jelikož vysoce závisí na délce vedení. Pro vzdálenosti do 40 m se dá počítat i s rychlostí 1 Mbit/s, ale pro vzdálenosti vyšší než 1 km se rychlost pohybuje v řádech desítek kbit/s a nižších. Toto omezení vyplývá již ze samotného účelu sběrnice, která byla prvotně navržena pro komunikace uvnitř automobilu a nikoliv na velké vzdálenosti. [2]

2.2.2.1 Maskování příjmu zpráv

Každý přijímač má k dispozici příjmový filtr (Acceptance filter), ke kterému patří registr příjmového kódu ACR (Acceptance Code Register) a registr příjmové masky AMR (Acceptance Mask Register). Pomocí masky AMR se nastavují bity adresy, které má filtr při komparaci ignorovat. Ostatní bity, ty které se neignorují, se musí shodovat s registrem ACR. Každý přijímač sice přijímá a kontroluje všechny zprávy na sběrnici, ale pouze ty, které projdou filtrem, ukládá do vyrovnávací paměti nebo registru. Přijetí takové zprávy pak signalizuje procesoru stavovým bitem, případně přerušením. [3]

3

Digitální řízení modelové železnice - DCC

DCC (Digital Command Controll) je standard používaný pro řízení modelových železnic definovaný společností NMRA (National Model Railroad Association) v jejich standardech pod kapitolou S-9.2. [7] Toto digitální řízení bylo navrženo pro řízení velkého množství lokomotiv, celých souprav i výhybek nezávisle na sobě. Pro rozvod signálu DCC stačí použít stejné dva vodiče (koleje) jak pro řízení, tak pro napájení řízených zařízení.

Komunikace pomocí DCC je v základu jednosměrná, signál DCC se generuje centrálně a je přijímán dekodéry v lokomotivách a instalovaných zařízeních.

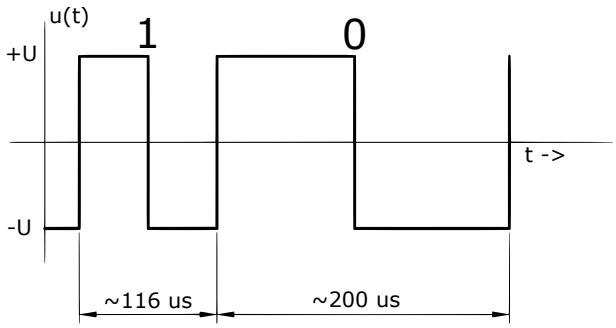
3.1 Fyzická vrstva řízení DCC

Významnou vlastností řízení pomocí DCC je využití kolejí jak pro napájení, tak pro přenos signálu. Pro zakódování signálu do napájení na kolejích se mění periodicky jeho polarita. Jeden znak je kódován délkou jedné periody signálu. Signál má standardně střídu 50 %. Tím je dosaženo nulové stejnosměrné složky napětí na kolejích a lokomotivy řízené stejnosměrným napětím stojí. Aby stejnosměrná složka napětí zůstávala nulová, musí se napětí na kolejích střídat nepřetržitě. To se zajišťuje buď neustálým opakováním příkazů, nebo odesíláním prázdných (idle) paketů.

Další výhodou systému DCC je také to, že nezáleží jakým směrem je na kolejích orientována lokomotiva. Protože je signál v podstatě kódován frekvencí a ne polaritou, obě koleje jsou si rovnocenné. To na druhou stranu může způsobit potíže s identifikací napájecích vodičů jednotlivých kolejí při stavbě kolejiště, aby při úpravách nedošlo k jejich záměně.

Pro logickou jedničku je stanovena délka periody 110 - 122 μs. Pro logickou nulu je možná délka periody 190 - 12000 μs, konkrétně 95 - 9900 μs pro trvání první polarity a 95 - 9900 μ pro trvání opačné polarity. [6]

Prodlužováním trvání první nebo druhé polarity v nulových bitech se dá modulovat

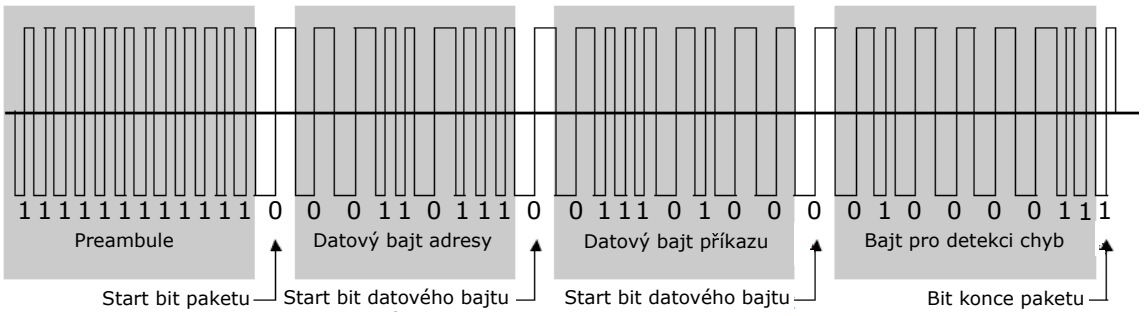


Obr. 3.1: Kódování logických úrovní DCC [10]

stejnosměrná složka signálu a tak řídit lokomotivy, které nepodporují DCC, pomocí stejnosměrného napětí. Řízení pak probíhá jakousi formou PWM (Pulse Width Modulation). Prodlužováním signálů se ale významně snižuje přenosová rychlost, a tak se v dnešní době tato možnost příliš nevyužívá a používají se výhradně lokomotivy podporující DCC.

3.2 Komunikační standard DCC

Řízení pomocí DCC probíhá formou zpráv, neboli paketů (viz obrázek 3.2). Každý paket obsahuje adresu a data s instrukcemi pro cílový dekodér. Každý paket má také na začátku preambuli a start bit označující jeho začátek a na jeho konci je vždy bajt pro detekci chyb a bit konce paketu.



Obr. 3.2: Struktura základního paketu DCC [7]

Preambule paketu (viz obrázek 3.2) musí mít pro příjem minimálně 10 po sobě jdoucích jedniček (centrála jich standardně musí odeslat minimálně 14). Po preambuli následuje nulový bit „Start bit paketu“, po kterém následují osmibitové bajty. Bajty jsou navzájem od sebe oddělené dalšími nulovými bity „Start bit datového bajtu“. Start bity zajišťují, že se jinde než v preambuli nenahromadí více jedniček za sebou. Poslední bajt sloužící pro detekci chyb je jednoduše logickou funkcí XOR všech předchozích bajtů. Celý paket je ukončen bitem konce paketu, který má hodnotu 1. Bit konce paketu se dá považovat již za preambuli následujícího paketu, pokud takový paket existuje.

Jelikož lokomotivy nemají vždy ideální kontakt s kolejemi a často dochází ke krátko-

dobým odpojením, například při nadskočení nápravy na výhybce nebo kvůli nečistotám, je výhodné příkazy pro jejich dekodéry opakovat.

3.2.1 Adresy DCC

Každý dekodér má obvykle přiřazenou jednu adresu. Adresa se pro většinu dekodérů vejde do jednoho adresního bajtu (v případě příslušenství jsou ještě 3 nebo 5 adresních bitů uvnitř prvního datového bajtu). Pro delší adresy lokomotivních dekodérů se používá ještě druhý adresní bajt.

Adresa	**Rozsah**	**Použití**
00000000	0	Broadcast – paket pro všechny dekodéry
0AAAAAAA	1-127	Lokomotivní (multifunkční) dekodéry se 7 bitovou adresou
10AAAAAA	128-191	Dekodéry příslušenství (s 9 nebo 11 bitovou adresou)
110AAAAA	192-223	Lokomotivní (multifunkční) dekodéry se 14 bitovou adresou
11100AAA	224-231	
11101AAA	232-254	Vyhrazeno pro budoucí použití
11111111	255	Idle packet (paket pro nikoho)

Tab. 3.1: Adresy v prvním adresovém bajtu paketu DCC [8]

Adresa jednotlivých dekodérů se obvykle uvádí v dokumentacích bez předpon, tedy jen jako číslo reprezentované bity „AAA…“ (viz tabulku 3.1). Z toho důvodu je také nutné dbát, aby se nepletly adresy lokomotivních dekodérů se 7 bitovou a 14 bitovou adresou. Obvykle stačí pro krátké adresy vyhradit nižší čísla a pro delší adresy nechat vyšší čísla.

Idle packet je paket, který má jeden prázdný (nulový) datový bajt a není určený pro žádný dekodér. Používá se, když centrála neví, jaké příkazy posílat, například v čase krátce po zapnutí, když centrála ještě nepřijala žádné příkazy. Dá se používat také pro řízení kolejiště analogově pomocí prodloužených nul, jakousi formou PWM. Pro tento účel je paket pro nikoho vhodný, protože obsahuje vysoké množství nul.

Broadcast je paket, jehož příkaz vykonají všechny dekodéry, které jsou toho schopny. Dá se použít například pro nouzové zastavení všech lokomotiv.[8]

3.2.2 Základní pakety DCC

Preambule		**Adresa**		**Data**		**Kontrolní bajt**	
111111111111	0	0AAAAAAA	0	01DCSSSS	0	EEEEEEEE	1

Tab. 3.2: Paket rychlosti a směru pro lokomotivní dekodéry s krátkou adresou [9]

Základní paket pro rychlost a směr (viz tabulku 3.2) lokomotivních dekodérů obsahuje jeden datový bajt, jehož nejvyšší dva bity (předpona) mají vždy hodnotu 01. Následující

bit označený D (Direction) určuje směr. Když má bit D, neboli bit 5, hodnotu 1, znamená to jízdu vpřed, hodnota 0 znamená jízdu vzad.

Bity 0-4 označené CSSSS ovládají rychlost, a to poněkud neobvyklým způsobem. Bit označený C[1] byl totiž v minulosti používán pro ovládání světel. V dnešní době ale většinou slouží jako nejnižší rychlostní bit pro zvýšení počtu rychlostních stupňů. Rychlost je při použití bitu C dána 5 bitovým binárním číslem s bity v pořadí „SSSSC“. V tomto čísle jsou jeho 4 nejnižší hodnoty (decimálně 0-3) použity pro definici čtyř různých druhů zastavení. Všechny možné stupně rychlosti jsou s hodnotami bitů 0-4 zobrazené v tabulce 3.3.

CSSSS	**Rychlost**	CSSSS	**Rychlost**	CSSSS	**Rychlost**	CSSSS	**Rychlost**
00000	stop	00100	stupeň 5	01000	stupeň 13	01100	stupeň 21
10000	stop(I)	10100	stupeň 6	11000	stupeň 14	11100	stupeň 22
00001	E-stop	00101	stupeň 7	01001	stupeň 15	01101	stupeň 23
10001	E-stop(I)	10101	stupeň 8	11001	stupeň 16	11101	stupeň 24
00010	stupeň 1	00110	stupeň 9	01010	stupeň 17	01110	stupeň 25
10010	stupeň 2	10110	stupeň 10	11010	stupeň 18	11110	stupeň 26
00011	stupeň 3	00111	stupeň 11	01011	stupeň 19	01111	stupeň 27
10011	stupeň 4	10111	stupeň 12	11011	stupeň 20	11111	stupeň 28

Tab. 3.3: Stupně rychlosti lokomotivy v datovém bajtu paketu DCC [7]

E-stop (Emergency stop) je označení pro nouzové zastavení, při kterém lokomotiva okamžitě vypne napájení motoru. Oproti tomu při běžném zastavení lokomotiva zastavuje plynule, s rychlostní rampou, což vypadá realističtěji.

Označení „(I)“ znamená, že se při zastavení ignoruje hodnota směrového bitu „D“. To má za následek, že směrově závislé funkce, například světla, zůstanou ve stavu podle posledního paketu, u kterého se směrový bit neignoroval. Toho se využívá například při nouzovém zastavení všech lokomotiv, za účelem zachování orientace jejich světel nezávisle na směru zadaném Broadcast paketem.

Ukázkový paket na obrázku 3.2 je příkaz pro lokomotivu s adresou 55 pro jízdu vpřed rychlostí 6.

Preambule		**Adresa**		**Data**		**Kontrolní bajt**	
111111111111	0	0AAAAAAA	0	100LFFFF	0	EEEEEEEE	1

Tab. 3.4: Paket instrukcí pro první skupinu funkcí [9]

Paket s instrukcemi pro první skupinu funkcí (viz tabulku 3.4) je velmi podobný paketu pro rychlost a směr, od kterého se liší pouze předponou v datovém bajtu. Je určen pro lokomotivní multifunkční dekodéry a ovládá až 5 jejich funkcí pomocí bitů

[1]Funkce, kterou bit označený C plní, je definována nastavením konfigurační proměnné #29 konkrétního dekodéru podle standardu S-9.2.2 společnosti NMRA

označených LFFFF. Hodnota každého z těchto bitů odpovídá stavu jedné funkce. Bity označené F odpovídají uživatelským funkcím F1 - F4. Bit označený L[2] obvykle ovládá světla lokomotivy.

3.2.3 Další pakety DCC

Preambule		**Adresa 1**		**Adresa 2**		**Data**		**Kontrolní**	
111111111111	0	110AAAAA	0	AAAAAAAA	0	01DCSSSS	0	EEEEEEEE	1

Tab. 3.5: Paket rychlosti a směru pro lokomotivní dekodéry s dlouhou adresou [9]

Paket rychlosti a směru s dlouhou adresou (viz tabulku 3.5) se od paketu s krátkou adresou liší pouze v počtu a tvaru adresních bajtů. První bajt adresy obsahuje vyšší bity adresy a druhý bajt obsahuje nižší bity adresy.

Preambule		**Adresa**		**Data**		**Kontrolní bajt**	
111111111111	0	10AAAAAA	0	1AAACDDD	0	EEEEEEEE	1

Tab. 3.6: Základní formát paketu pro dekodéry příslušenství [9]

Dekodéry příslušenství jsou používány například pro ovládání výhybek. Mohou být nastaveny pro krátkodobé spínání výstupů nebo pro ovládání stále buzených zařízení. Nastavení délky sepnutí se provádí pomocí konfiguračních proměnných dle standardu S-9.2.2 společnosti NMRA.

Pakety pro dekodéry příslušenství (viz tabulku 3.6) mají adresu rozdělenou do adresního a datového bajtu. Důležité je vědět, že adresní bity v datovém bajtu jsou zapsány inverzně a že jsou to 3 horní bity z celkové 9 bitové adresy. Jsou to tedy adresní bity $A_8 - A_6$. V adresním bajtu jsou pak adresní bity $A_5 - A_0$.

Bit označený C slouží pro aktivaci nebo deaktivaci konkrétního výstupu a bity označené DDD slouží k adresování konkrétního výstupu dekodéru. Každý dekodér tak může ovládat 8 výstupů, pomocí kterých se dá ovládat 4 výhybky. Každá výhybka obvykle disponuje dvěma vstupy, mezi kterými zpravidla vybíráme nejnižším bitem datového bajtu D_0.

[2]Funkce bitu „L“ je opět definována nastavením dekodéru. Pokud má konfigurační proměnná #29 hodnotu 1, bit označený L ovládá světla.

4

Realizace jednotky generátoru DCC

Abychom začali s vlastní realizací generátoru DCC signálu pro modelovou železnici, musíme pochopit, jaký účel bude tato jednotka plnit. Při návrhování jednotky dle zadání této práce je třeba brát ohled na způsob jejího vývoje, testování, implementace i provozu.

Hlavní požadovanou funkcí této jednotky je generování příkazů pro lokomotivy modelového kolejiště. Příkazy je zapotřebí přijímat od centrálního řízení na PC, se kterým komunikuje pomocí sběrnice CAN (Controller Crea Network). Výstupní příkazy pro řízení lokomotiv mají formu DCC (Digital Command Control).

Další v zadání definovanou vlastností jednotky jsou její rozměry, které jsou standardizované pro všechny jednotky pro správu kolejiště. Požadováno je také jednotné rozvržení konektorů. To podstatně určuje fyzické provedení jednotky. Znamená to, že půjde o desku plošných spojů osazenou součástkami kolem předdefinovaného rozvržení.

K realizaci požadovaných funkcí a splnění požadavků je zapotřebí vybrat vhodný mikrokontrolér, rozumně dostupný, schopný vyhovět našim požadavkům. Z účelu generátoru vyplývá také potřeba dvou rozhraní - CAN a DCC, po kterých dokáže jednotka správně komunikovat. Například k tvorbě signálu DCC je zapotřebí generovat obdélníkové signály s frekvencí přibližně 5 - 9 kHz. Pochopitelně je také potřeba zajistit stabilní napájení jednotlivých komponentů jednotky.

Jelikož se jedná o první iteraci tohoto zařízení a bude je nutné testovat, ladit a kontrolovat, je vhodné jednotku opatřit i jednoduchými uživatelskými vstupy a výstupy. Myšleny jsou například v podobě tlačítek a LED diod pro indikaci a kontrolu různých stavů a funkcí jednotky. Vstupy jednotky můžeme rozšířit i o rozhraní pro připojení přímo k PC a vytvořit tak alternativní způsob zadávání příkazů. K tomu se hodí rozhraní USB (Universal Serial Bus).

4.1 Návrh

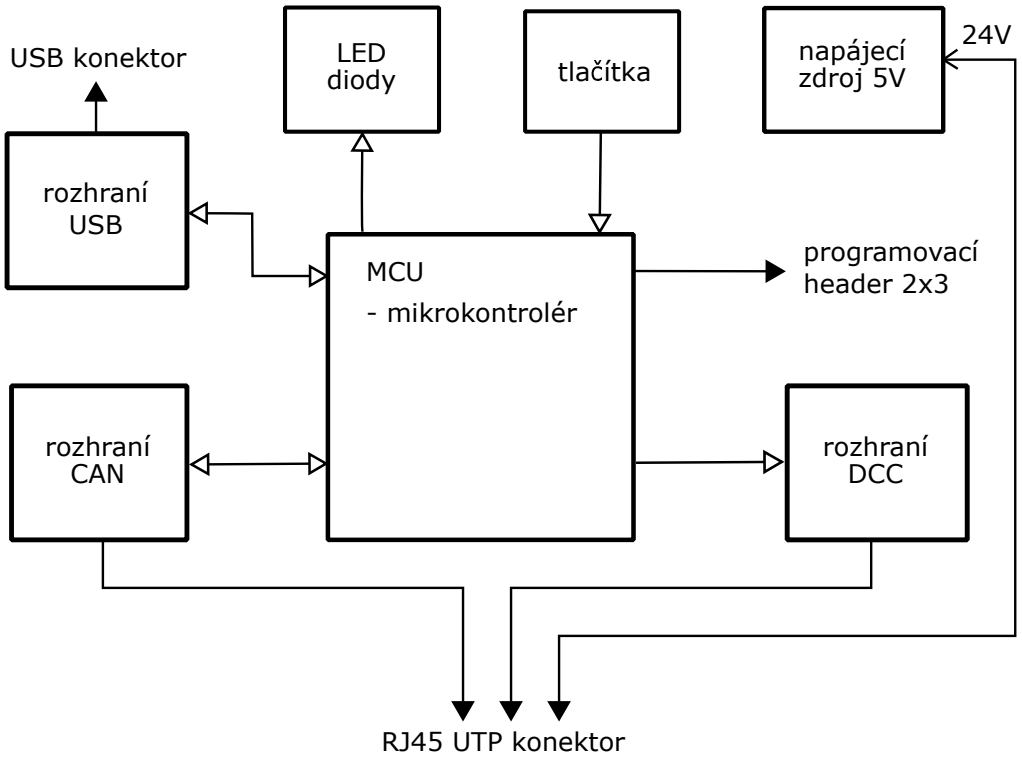
K návrhu desek plošných spojů se v dnešní době téměř výhradně používají návrhové systémy, které umožňují na počítači vytvořit velice podrobný model finálního produktu. Umožňují také export podkladů, podle kterých se následně zadává výroba desky a podle

kterých se postupuje i při nákupu součástek a osazování. Takových návrhových systémů je na trhu více a jsou mezi nimi podstatné rozdíly jak v efektivitě práce a podporovaných funkcích, tak převážně v cenách. Freeware nástroje mohou být pro jednodušší projekty postačující, ale práce s nimi je složitější. Pro rozsáhlejší projekty se v praxi nehodí, ani se nepoužívají. Na druhou stranu profesionální programy stojí nemalé peníze. Naštěstí společnosti často poskytují studentské licence na rok i více. Takové možnosti bylo využito v případě návrhu této jednotky. Možnost učit se s nástrojem, který je využitelný i v budoucí praxi, byl jeden z hlavních důvodů výběru návrhového softwaru. [11]

Pro návrh jednotky byl použit návrhový software Altium Designer od společnosti Altium Ltd. Altium Designer je profesionální návrhový systém pro elektroniku, který je běžně používán společnostmi v praxi k vývoji hardware. Jeho možnosti daleko přesahují rozsah této práce a nebudeme je všechny do hloubky rozebírat, pouze pro přiblížení postupu práce na projektu uvedeme postupy, které byly použity při návrhu.

4.1.1 Blokové schéma

Ještě před výběrem součástek je pro usnadnění návrhu a pochopení funkce hardwaru jednotky výhodné si vytvořit blokové schéma, kde jednotlivé bloky zastupují každý jednu funkci.



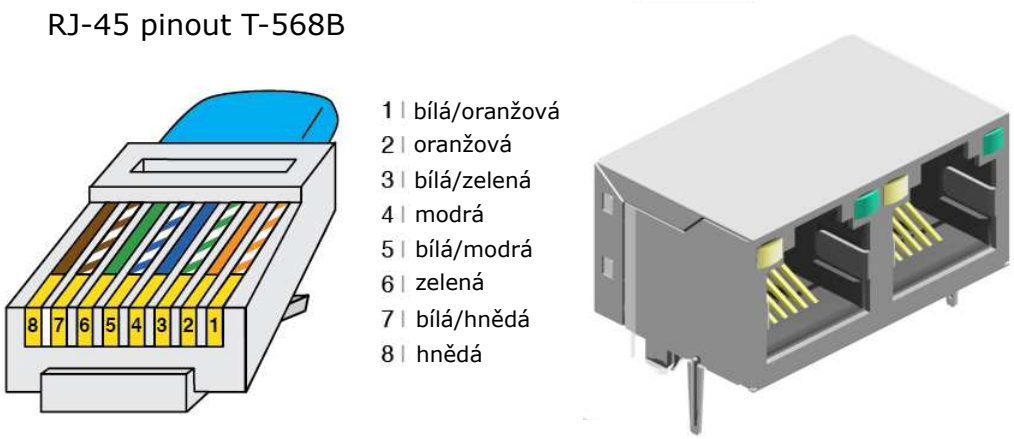
Obr. 4.1: Blokové schéma generátoru DCC

Na blokovém schéma (viz obrázek 4.1) jsou naznačeny všechny vstupy a výstupy mikrokontroléru a také je zde znázorněno zapojení konektorů na desce. Mnohé z funkčních

bloků se dají realizovat pomocí integrovaných obvodů, které jsou běžně dostupné. K jejich úspěšnému využití často pouze stačí přidat pár pasivních součástek podle doporučení v jejich technických podkladech.

Konektory slouží k propojení s dalšími jednotkami, k připojení k PC a k programování. Od jejich umístění se odvíjí umisťování a propojování dalších bloků jednotky.

Konektor a rozvod kabelu UTP



Obr. 4.2: Konektor RJ-45 pinout T-568B [13] [14]

Ve standardizovaném rozvržení konektorů jednotky je použit dvojitý konektor RJ-45 (viz obrázek 4.2). Tento konektor umožňuje snadné propojování jednotlivých jednotek pomocí kabelu UTP. Tento kabel obsahuje čtyři barevně odlišené páry vodičů zapojené podle standardu T-658B. Dva páry slouží pro rozvod napájení a druhé dva páry pro rozvod komunikačních sběrnic. Každý z vodičů konektoru RJ-45, resp. kabelu UTP je zapojený pro účel definovaný v tabulce 4.1.

Číslo pinu	**Barva**	**Označení**
1	bílá/oranžová	CAN_H
2	oranžová	CAN_L
3	bílá/zelená	24V
4	modrá	GND
5	bílá/modrá	24V
6	zelená	GND
7	bílá/hnědá	DCC+
8	hnědá	DCC-

Tab. 4.1: Zapojení konektoru RJ-45 [1]

4.1.2 Výběr mikrokontroléru

Mikrokontrolér neboli jednočipový počítač, anglicky microcontroller, je zpravidla integrovaný obvod kompaktních rozměrů obsahující celý mikropočítač. Ten obvykle obsahuje procesor, potřebné paměti, interní datové sběrnice, I/O (Input / Output) obvody a často také AD (Analog - Digital) převodník a hardware pro ovládání sběrnic jako jsou I2C, SPI nebo CAN. Mikrokontroléry se hojně využívají pro realizaci vestavěných (embedded) systémů. Jedním z nich je například ten, který je předmětem této práce. Mikrokontroléry se v embedded systémech využívají pro svou vysokou spolehlivost, malé rozměry a velký rozsah využití.

Mikrokontrolér je v našem projektu jádrem celé jednotky. Vykonává běh programu, propojuje všechny ostatní funkční bloky jednotky a řídí jejich chod. Vyplývající nároky na mikrokontrolér jsou tedy značně rozsáhlé. Mikrokontrolér musí být dostatečně výkonný a musí dokázat obsloužit dostatečné množství vstupů a výstupů s potřebnými funkcemi. Potřebujeme, aby byl přiměřeně rychlý, měl dostatečnou paměť a abychom ho mohli bez potíží programovat a následně program ladit. Zároveň chceme, aby byl mikrokontrolér cenově dostupný a bylo možné ho snadno pořídit.

Na doporučení vedoucího práce jsme se při výběru zaměřili na nabídku firmy NXP. Pro naše účely postačí 8 bitový mikrokontrolér. Jednotka generátoru nebude provádět příliš složitých výpočtů ani nebude zpracovávat analogové signály. Další požadovanou vlastností je hardwarová podpora sběrnice CAN. S těmito parametry vyrábí firma NXP mikrokontroléry rodiny S08. Tyto mikrokontroléry mají maximální frekvenci hodin 40 MHz, což je postačující pro generování DCC signálu, který je frekvenčně řádově na 10 kHz.

Vybrali jsme tedy mikrokontrolér řady S08DZ s označením MC9S08DZ96CLF, jehož základní vlastnosti jsou následující:

- `8 bit` šířka datové sběrnice
- `40 MHz` maximální frekvence hodin
- `6 kB` datové RAM paměti
- `96 kB` programovací paměti typu FLASH
- `48 pinů` na poudře LQFP[1]
- `40 I/O` vstupů / výstupů
- `2.7 - 5.5 V` napájecí napětí
- `Podpora rozhraní:` CAN, I2C, SCI, SPI [12]

Tento mikrokontrolér také disponuje třemi čítači/časovači, které můžeme využívat pro generování PWM signálu na výstupech. Toho se dá využít při generování signálu DCC.

[1] Low profile Quad Flat Pack

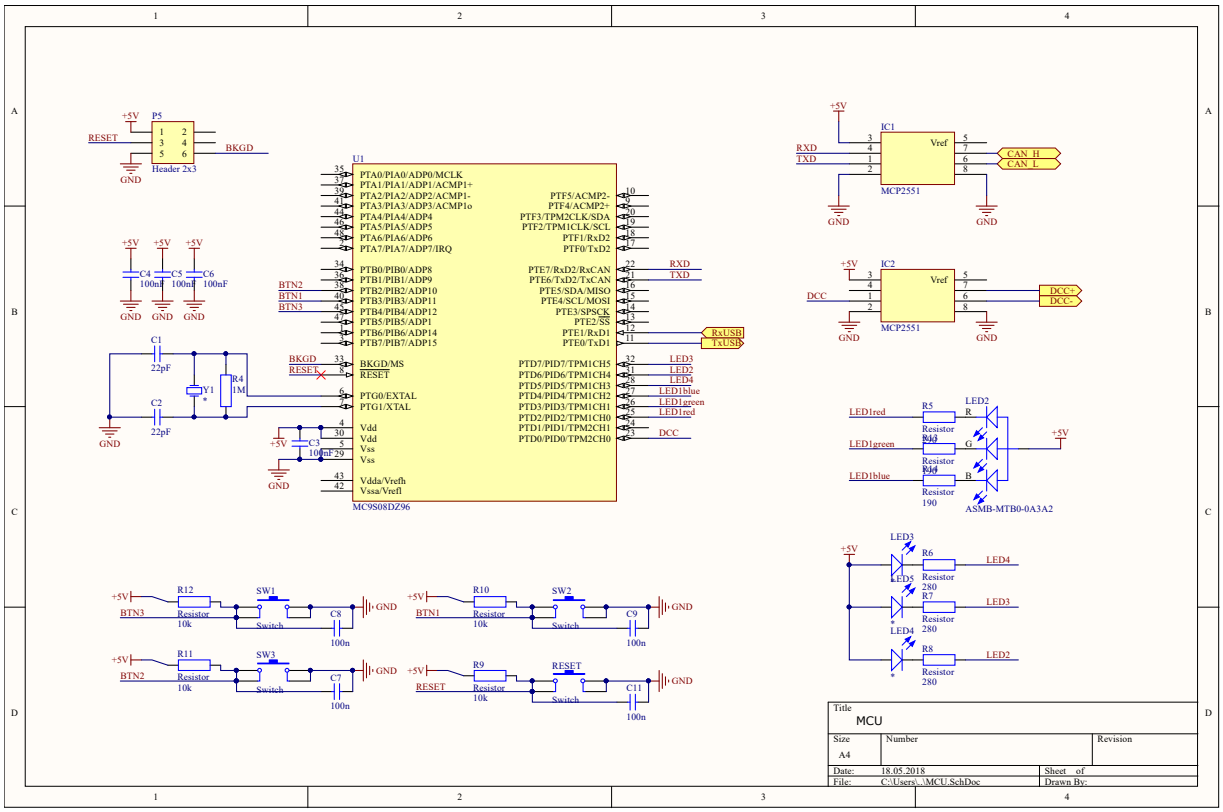
4.1.3 Postup návrhu desky plošných spojů

Při návrhu desky plošných spojů bylo nutné nejprve vybrat vhodné součástky pro potřebné funkce. Byl vybrán CAN transciever (vysílač + přijímač), tvořící rozhraní mezi kontrolérem CAN protokolu v mikrokontroléru a jeho fyzickou sběrnicí. Stejný transciever se zároveň dá použít i pro rozhraní DCC. Je od výrobce Microchip a nese označení MCP2551. Pro rozhraní USB byl vybrán integrovaný obvod od firmy FTDI, s označením FT232R, který převádí USB na sériovou komunikaci pomocí UART (Universal Asynchronous Receiver and Transmitter).

K napájení všech integrovaných obvodů je využit step-down regulátor napětí na 5V s proudovou zatížitelností 1A.

Návrh elektronického schématu

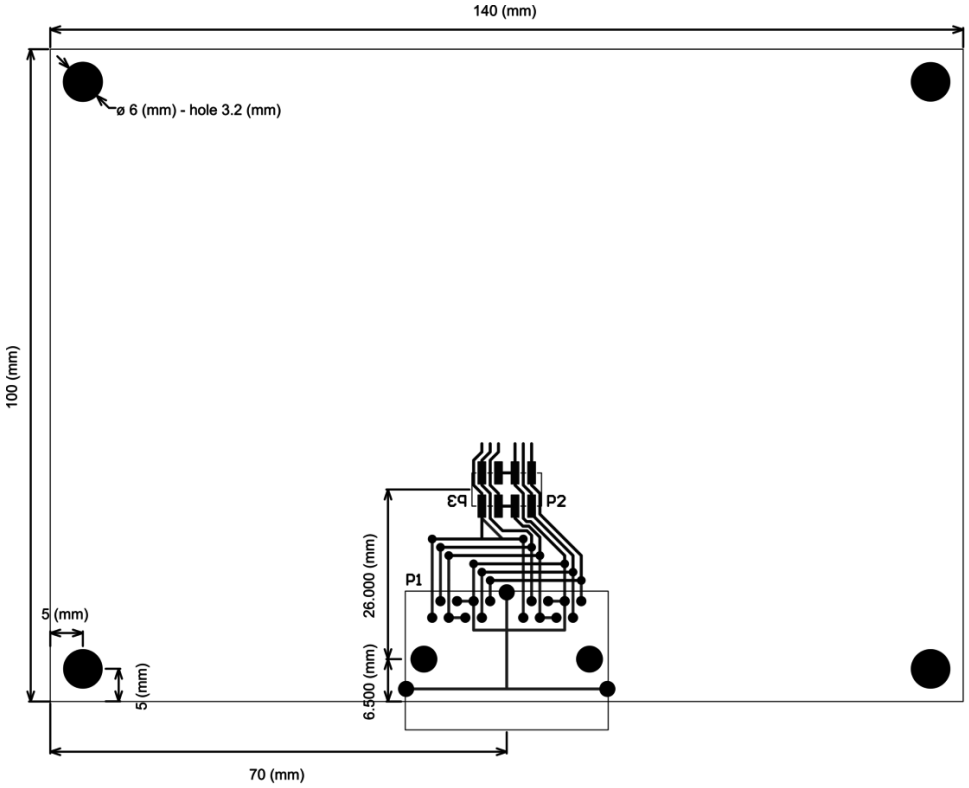
S použitím knihoven v návrhovém systému Altium Designer je následně nutné implementovat všechny potřebné integrované obvody a další pasivní a aktivní součástky do elektronického schématu nového projektu. Propojování a přidávání jednotlivých prvků se provádí s ohledem na přehlednost a především elektrickou správnost. Pasivní součástky jako jsou blokovací kondenzátory, pull-up rezistory a cívky se přidávají k jednotlivým integrovaným obvodům podle potřeby. Část našeho návrhu elektronického schématu kolem mikrokontroléru je vidět na obrázku 4.3 níže.



Obr. 4.3: Návrh elektrického schéma zapojení mikrokontroleru

Návrh plošného spoje

Po návrhu elektrického schématu přecházíme k návrhu plošného spoje. To znamená rozmisťování součástek a rozvádění vodivých cest. Pro začátek návrhu byl importován vzorový projekt se standardizovanými rozměry desky a základním rozvržením konektorů (viz obrázek 4.4). Součástí vzorového projektu jsou i montážní otvory pro sloupky, které spolu se hřebínkovým konektorem na určené pozici ve středu desky umožní instalaci více jednotek nad sebe a zároveň jejich propojení bez použití kabelu.

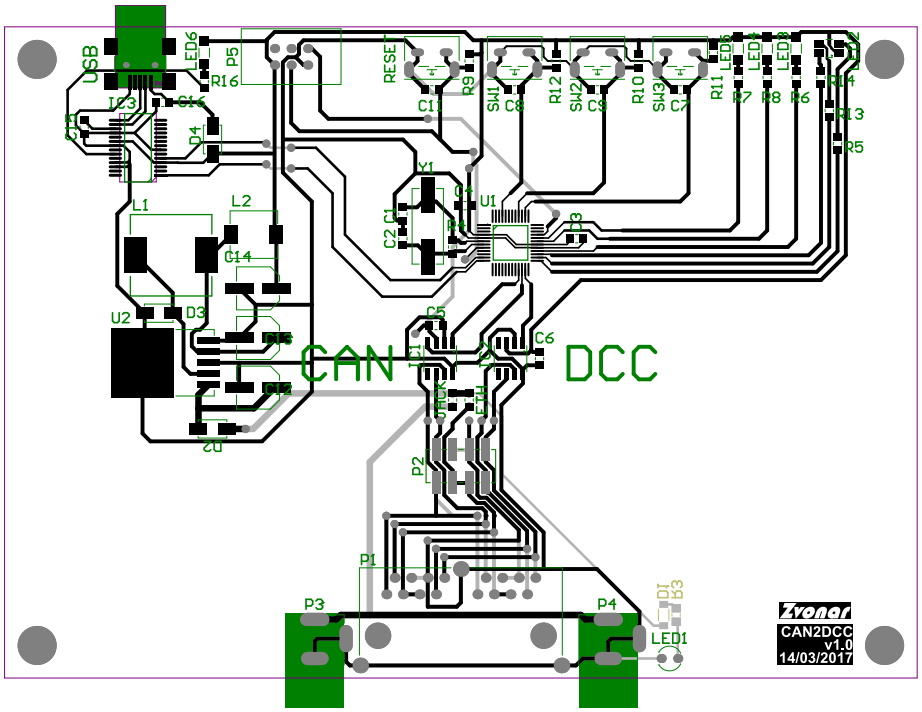


Obr. 4.4: Standardizovaný rozměr DPS s konektory

Prvním krokem při návrhu je nastavení pravidel návrhu jako jsou minimální vzdálenosti mezi ploškami nebo minimální a typické tloušťky cest. Při rozmisťování součástek u našeho návrhu nebylo zapotřebí používat příliš složitého propojování. Součástky se snadno vejdou na jednu stranu desky. Pro několik potřebných křížení byla využita i druhá strana.

Tlačítka a LED diody byly pro snazší obsluhu umístěny ke kraji desky. Výsledný návrh plošného spoje (pro přehlednost ještě bez rozlité mědi) je na obrázku 4.5.

Na desce byla nakonec rozlita zem i s ostrůvky mědi. Rozlití mědi usnadní výrobu desky tím, že není potřeba odstraňovat tolik materiálu. Zároveň zlepšuje chlazení součástek, pod kterými je rozlita, a také působí jako stínění proti elektromagnetickému rušení.



Obr. 4.5: Návrh DPS (bez rozlité mědi)

4.2 Výroba jednotky

Pro zadání výroby jednotky se používají podklady, které se exportují z návrhového systému. Pro výrobu plošného spoje se exportují podklady obsahující zvlášť jednotlivé vrstvy plošného spoje a také podklady pro vrtání děr.

Pro následné osazení se používá osazovací diagram, podle kterého se umisťují součástky na plošný spoj. Z projektu se také exportuje seznam součástek BOM (Bill of material), podle kterého se objednají potřebné součástky. Pro jednodušší nákup součástek se do BOM uvádí přímo konkrétní produkty dostupné u dodavatele.

V našem případě byla výroba plošného spoje pro první verzi jednotky zadána školní fréze na fakultě ZČU. Výroba tímto způsobem byla rychlá s kvalitou pro výrobu prototypu naprosto postačující.

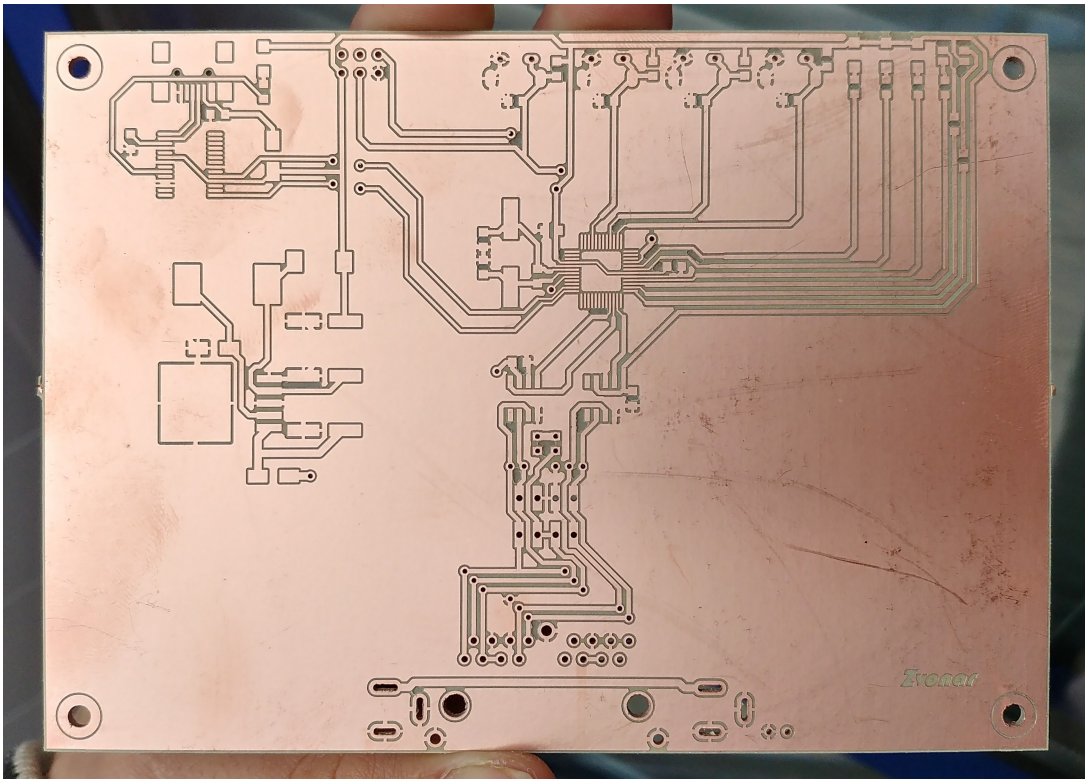
4.2.1 Způsoby výroby plošných spojů

Materiál pro výrobu desek plošných spojů se zpravidla skládá ze substrátu a jedné nebo dvou vodivých vrstev na něj nalepených nebo nalisovaných. Vodivé vrstvy jsou nejčastěji z měděné folie o tloušťce řádově desítek μm. Jako substráty se nejčastěji používají kompozitní materiály. Nejběžnějším materiálem pro výrobu DPS je materiál FR-4, který se skládá ze dvou měděných vrstev a substrátu se sklo-epoxidového kompozitu. Jednovrstvé plošné spoje se v dnešní době příliš nepoužívají, protože neumožňují snadné křížení vodivých cest. Proto jsme i pro naši jednotku použili dvouvrstvý plošný spoj z materiálu FR-4.

K vytvoření plošného spoje na desce existuje několik metod. Spoje se leptají, frézují nebo vypalují laserem. Leptání se používá spíše pro výrobu ve větších sériích, jelikož je to proces o mnoha krocích. Je zapotřebí vyrobit masku, nanést fotorezist, vyvolat ho a mezitím desku oplachovat, atd... Výroba frézováním je oproti tomu proces daleko jednodušší, jelikož téměř veškerou práci vykoná fréza najednou. Nevýhody této metody jsou, že fréza může zanechávat otřepy a také je její jemnost omezena tloušťkou vrtáku. Nebývá tedy tak přesná, ale je vhodná i pro kusovou výrobu.

Pro prokovení potřebných otvorů po vrtání se používá chemické a elektrolytické prokovení.

Po vytvoření plošných spojů se na desky standardně nanáší ještě další vrstvy. Ty zajišťují ochranu vůči vnějším vlivům, pomáhají při pájení a nebo jde o potisk pro osazování a popis desky. Nejčastěji se na desku nanáší nepájivá maska, která pokrývá celý povrch kromě pájecích plošek. Na tuto vrstvu se nepřichytává pájka a slouží také jako ochranná vrstva. Pájecí plošky se ve výrobě běžně ošetřují tenkou vrstvou cínu nebo jiného materiálu, který konzervuje měď pod ním a pomáhá při následném pájení.



Obr. 4.6: Deska plošných spojů před osazením

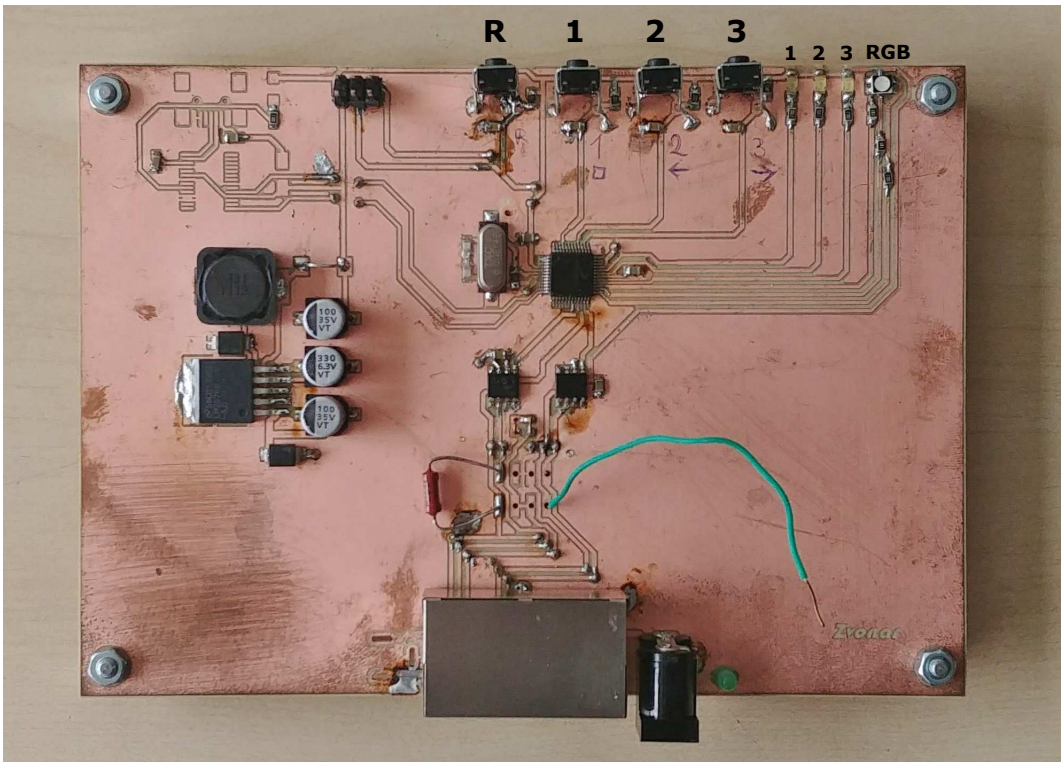
Pro naši jednotku generátoru byla použita dvouvrstvá deska FR-4 vyrobená na fréze (viz obrázek 4.6). Otvory byly vyrobeny bez prokovení a měď bez povrchových úprav a bez dalších vrstev jako je nepájivá maska. Protože surová měď je náchylná k oxidaci, použili jsme ochranný sprej, obdobu roztoku kalafuny, který oxidaci z větší části zabránil.

4.2.2 Osazování desky plošných spojů

Součástky pro osazování se dělí dle způsobu osazování na typ THT (Through-hole technology) a SMD (Surface mount device). Součástky THT mají vývody v podobě drátů, které se prostrkují skrz díry v pájecích ploškách. K osazení součástek SMD se díry nepoužívají a jejich vývody se pájí přímo na pájecí plošky. SMD součástky také bývají levnější a mají menší rozměry. Výhodou součástek THT může být jejich pevné ukotvení na desce. Proto se dnes THT součástky používají většinou pro konektory, tlačítka a jiné součástky u kterých hrozí jejich fyzické namáhání. Ostatní součástky se většinou používají v pouzdrech SMD, které šetří místo a usnadňují výrobu desky.

Osazování malých sérií a prototypů se obvykle provádí ručně pomocí pájecí stanice. Pro součástky s těžko dostupnými vývody se dá použít horkovzdušná páječka s pájecí pastou.

Při osazování naší jednotky, jejíž otvory nebyly prokovené bylo zapotřebí je ručně prokovit připájením kousků drátu. Osazování THT součástek netradičně ze stejné strany jako SMD bylo také obtížnější.



Obr. 4.7: Osazená jednotka generátoru DCC s označením tlačítek a LED diod

4.2.3 Testování jednotky

Při výrobě desky plošných spojů je nutné kontrolovat její vlastnosti, aby se případné problémy včas odhalily. Při výrobě na fréze můžou vzniknout otřepy, které zkratují jednotlivé

cesty, nebo se může některá z cest přerušit. Pro kontrolu se u prototypů běžně používá multimetr a cesty se testují jednotlivě.

Kontrola cest se provádí opakovaně i po připájení jedné nebo menšího počtu součástek. Obzvlášť u desek bez nepájivé masky se totiž stává, že se pájka rozlije a vytvoří zkrat mezi pájecími ploškami. Při pájení může také dojít k odtržení plošky ze substrátu, což má za následek přerušení cesty.

Po osazení je výhodné otestovat nejdříve napájecí zdroj jednotky. Při jeho testování je vhodné ho dočasně oddělit od zbytku obvodů.

Následně se testuje funkčnost ostatních funkčních bloků jednotky. Začíná se standardně oživením mikrokontroléru a rozblikáním jedné z LED diod.

4.2.4 Úpravy oproti návrhu

Při vlastní realizaci bylo největší změnou oproti návrhu neosazování rozhraní USB, které bylo plánováno pro použití jako alternativní způsob připojení k PC a zadávání příkazů pro lokomotivy. Jelikož ale s námi po celou dobu spolupracoval kolega vyvíjející software pro řízení kolejiště po rozhraní CAN, nebylo rozhraní USB pro vývoj vůbec potřeba.

K další drobným úpravám došlo u osazování tlačítek, která měla v knihovně součástek v době návrhu prohozené vývody. Na desce bylo také propojeno napájení z napájecího konektoru s napájením v UTP kabelu napevno. To bylo provedeno pomocí drátu namísto plánovaných zkratovacích propojek. K vodičům signálu CAN_H a CAN_L byl také přidán odpor (terminátor) pro impedanční přizpůsobení.

4.3 Firmware jednotky

K vývoji firmware jednotky byl využit software CodeWarrior Development Studio pro mikrokontroléry ve verzi 10.6.4 od firmy NXP (dříve Freescale). Tento vývojový program je dobrým nástrojem pro programování všech mikrokontrolérů od firmy NXP. K programování byl z nabídky jazyků použit jazyk C.

Velmi užitečnou vlastností CodeWarrioru je, že obsahuje modul Processor Expert, který usnadňuje práci s inicializací a nastavováním mikrokontroléru. Například místo ručního nastavování řídících registrů stačí v modulu Processor Expert vybrat požadovaná nastavení z nabízených možností a to v přehledném uživatelském prostředí.

Druhý hojně používaný modul je Debugger, který usnadňuje ladění programu a hledání případných chyb v kódu i v hardware. S jeho pomocí se dá běh programu pozastavit a přečíst hodnoty proměnných v mikrokontroléru, a tím ověřit funkčnost programu. Umožňuje také program zastavit na potřebném místě v kódu, nebo ho krokovat. [15]

Funkce programu

Hlavní funkcí programu je generování příkazů pro lokomotivy. Jelikož kvůli nečistotám často dochází k dočasným přerušením kontaktu s kolejemi, je vhodné příkazy pro lokomotivy opakovat. K tomu slouží matice příkazů, ve které se příkazy ukládají pro každou lokomotivu zvlášť a která se odesílá neustále dokola. Námi použitá matice má 26 řádků pro příkazy pro 13 adres školních lokomotiv. Prvních 13 slouží pro rychlost a druhých 13 pro první skupinu funkcí dekodérů lokomotiv. 27. řádek je vyhrazen pro příkazy mimo adresy lokomotiv.

Druhou důležitou funkcí programu je odesílání tzv. watchdog zpráv, neboli zpráv hlídače, které po rozhraní CAN neustále signalizují centrálnímu řízení funkčnost generátoru. Je to důležité, protože při výpadku signálu DCC by se bez vypnutí napájení kolejí lokomotivy mohly přepnout na řízení stejnosměrným napětím. Například by se mohlo stát, že se po odpojení generátoru všechny lokomotivy rozjedou maximální rychlostí jedním směrem. Proto musí řídící software kolejiště hlídat funkčnost generátoru a při jeho výpadku vypnout buzení kolejí v jednotkách zesilovačů DCC. To je funkce prvního (DCC) watchdog čítače.

Druhý watchdog čítač hlídá naopak, že nedošlo k výpadku centrálního řízení a případně zadá příkaz pro zastavení všech lokomotiv.

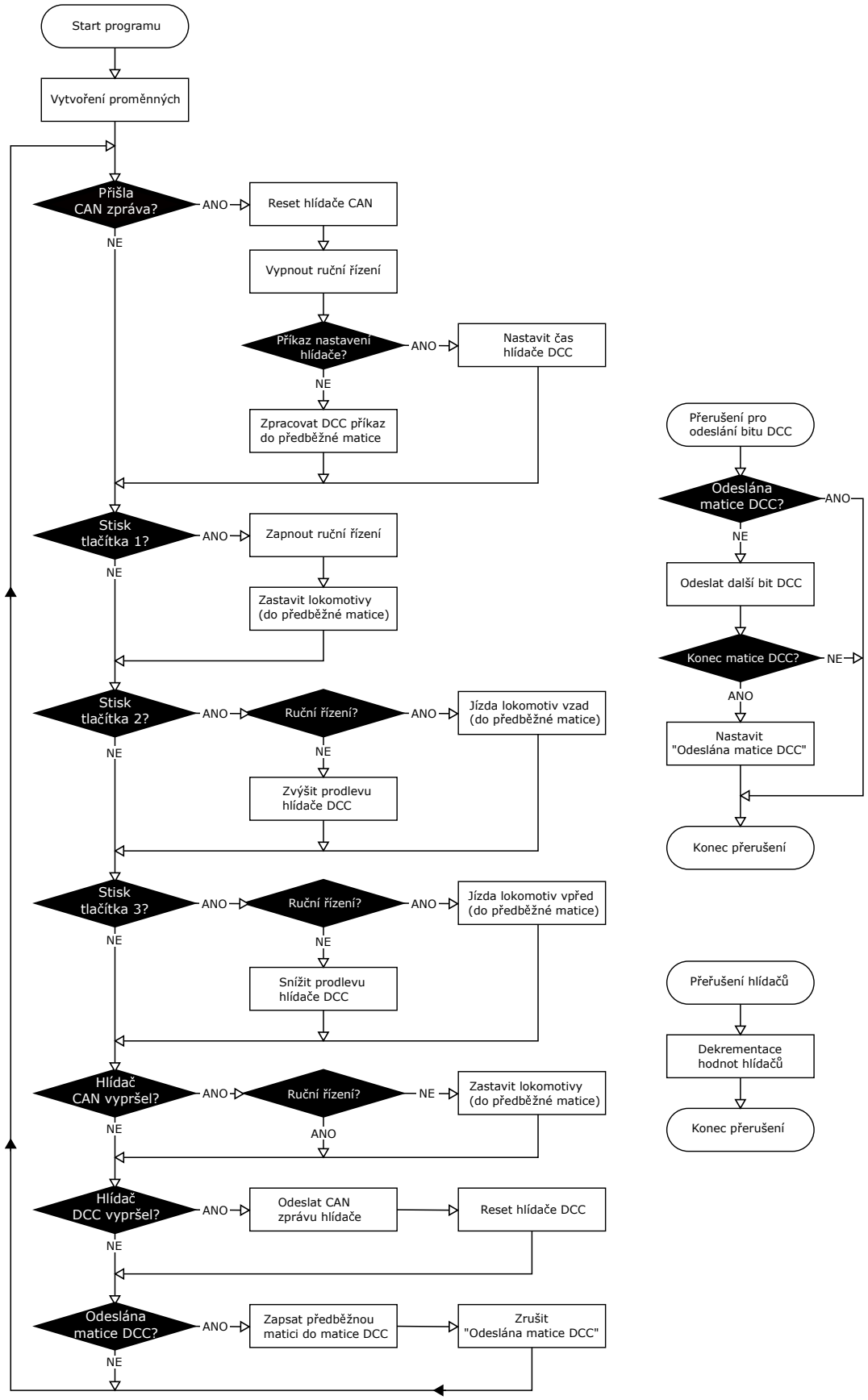
4.3.1 Vývojový diagram programu

Vývojový diagram (viz obrázek 4.8) ukazuje fungování programu ve zjednodušené podobě. V hlavní smyčce programu se kontroluje stav tlačítek, stav watchdog čítačů, neboli hlídačů, a také se se zde zpracovávají přijaté zprávy CAN.

Změny v příkazech pro lokomotivy se nejprve zapisují do předběžné matice, která se zkopíruje k odeslání do DCC až po odeslání předchozí verze matice. To trvá řádově desetinu vteřiny.

Odesílání jednotlivých bitů zajišťuje přerušení (interrupt) vyvolané PWM časovačem po každé periodě PWM signálu. V tomto přerušení se okamžitě nastaví délka následující periody signálu podle hodnoty následujícího bitu v matici příkazů. Takto vzniklý signál má podobu DCC.

Druhé přerušení je vyvolané časovačem s periodou 1 ms a zajišťuje dekrementaci watchdog čítačů.



Obr. 4.8: Vývojový diagram

4.3.2 Ovládání jednotky pomocí sběrnice CAN

Pro zadávání příkazů lokomotivám pomocí generátoru se používají zprávy CAN s 11 bitovým identifikátorem. Více o protokolu CAN se píše v kapitole 2.2 Princip sběrnice CAN.

Identifikátory zpráv CAN

Identifikátor slouží k rozlišení účelu zprávy. Každá jednotka si podle něj může pomocí příjmového filtru odfiltrovat zprávy pro ní určené.

V našem systému kolejiště se použitý 11 bitový identifikátor skládá ze dvou částí. Vyšší 4 bity určují typ jednotky a směr komunikace a dolních 7 bitů udává číslo jednotky. Číslování jednotek 7 bitovým číslem se využije například pro jednotky zesilovačů DCC, kterých systém potřebuje řádově desítky.

ID se vytváří podle následujících tabulek 4.2 a 4.3.

	Horní bity ID	**Typ jednotky**	**Směr**
0	0000	Emergency	W
1	0001	Generátor DCC	W
2	0010	Generátor DCC	R
3	0011	Zesilovač DCC 1	R
4	0100	Zesilovač DCC 2	R
5	0101	Zesilovač DCC 1	W
6	0110	Zesilovač DCC 2	W
7	0111	Výhybky	W
8	1000	Výhybky	R
9	1001	Návěstidla	W
10	1010	Točna	W
11	1011	Točna	R

Tab. 4.2: Typy jednotek v identifikátoru CAN zprávy - horní 4 bity ID

Písmeno W (Write) označuje směr dat z centrálního řízení do jednotky a písmeno R (Read) označuje směr dat z jednotky pro centrální řízení.

	Typ = 1	**Číslo = 1**	**Identifikátor**
bin	0001	0000001	00010000001
hex	1	1	0x081

Tab. 4.3: Příklad ID zprávy CAN pro zápis do generátoru DCC

Řízení lokomotiv

Pro ovládání lokomotiv se používají dva datové bajty zprávy CAN. První bajt obsahuje adresu lokomotivy a druhý data pro dekodér lokomotivy. Adresy lokomotiv jsou uvedeny v tabulce 4.4. Použitím adresy 1 se data pro dekodér zapíší do všech 13 adres. Tato adresa tak nahrazuje funkci standardního DCC broadcastu, který používá adresu 0 a jeho překlad je v současné verzi jednotky vynechán.

Lokomotiva	**Adresa**
Všechny	1
Brejlovec	3
Desiro (Kamera)	5
Taurus Railion	6
DB204.274-5	7
Ragulin	9
ICE	10
Taurus DHL	11
Herkules Priessnitz	12
Para 555	13
T334 Rosnicka	14
Desiro DB642.133-3	15
Taurus EVB	16
ES363	17

Tab. 4.4: Adresy lokomotiv

Datový bajt pro ovládání směru a rychlosti má oproti bajtu v protokolu DCC přesunutý nejnižší bit rychlosti (C) na nejnižší pozici v bajtu. Tím je pro CAN zjednodušeno ovládání rychlosti. Bajt pro rychlost má tedy tvar **01DSSSSC**, kde bit "D" ovládá směr. Hodnota 1 znamená jízdu vpřed a hodnota 0 jízdu vzad. Bity označené SSSSC ovládají rychlost. čtyři nejnižší hodnoty rychlosti (decimálně 0-3) jsou vyhrazeny pro různé druhy zastavení (viz následující tabulku 4.5). Příklad CAN zprávy pro rychlost lokomotivy je v tabulce 4.6.

Rychlost	Bitově	**Druh zastavení**
0	00000	zastavení
1	00001	zastavení bez změny směru (ignoruje se bit "D")
2	00010	nouzové zastavení (okamžité vypnutí motorů lokomotivy)
3	00011	nouzové zastavení bez změny směru

Tab. 4.5: Stupně rychlosti lokomotivy pro různé druhy zastavení

	ID	**1. datový bajt**	**2. datový bajt**
bin	00010000001	00001010	01101001
hex	0x081	0x0A	0x69

Tab. 4.6: Příklad zprávy pro lokomotivu ICE pro jízdu vpřed rychlostí 9

Datový bajt pro ovládání první skupiny funkcí lokomotivního dekodéru zůstává stejný jako je definován v protokolu DCC, viz kapitolu 3.2.2 Základní pakety DCC. Tento bajt se používá například pro ovládání světel lokomotivy.

Watchdog zprávy DCC

Jedinou zprávou, kterou jednotka generátoru po rozhraní CAN odesílá, je watchdog zpráva. Její ID má hodnotu 0x101 a obsahuje jeden nedůležitý datový bajt s hodnotou 0x55.

Pro nastavení intervalu odesílání watchdog zpráv, které informují o funkčnosti generátoru, se používá zpráva podobná zprávě pro nastavení rychlosti lokomotivy. Hodnota prvního datového bajtu této zprávy je 0. Pro nastavení intervalu se používá druhý datový bajt. Ten může mít hodnotu 0-255 decimálně, což odpovídá intervalu 15-4095 ms.

Interval odesílání lze také měnit pomocí tlačítek 2 a 3 na desce generátoru, pokud ale jednotka zrovna není v režimu nouzového ručního řízení lokomotiv.

Watchdog CAN řízení

Pro ujištění o funkčnosti nadřazeného řízení potřebuje jednotka generátoru pravidelně přijímat jakoukoliv zprávu CAN s jejím identifikátorem. Interval, po kterém vyprší watchdog čítač CAN, je nastaven na 1,5 s. Pokud jednotka nepřijme žádnou zprávu po dobu delší než 1,5 s, okamžitě zastaví všechny lokomotivy.

4.3.3 Nouzové ruční řízení lokomotiv

Při výpadku řízení lokomotiv pomocí CAN je možné tlačítkem 1 spustit ruční řízení, viz vývojový diagram obr. 4.8. Tlačítka jsou číslována jako na obrázku 4.7. Při ručním řízení se mění příkazy hromadně pro všech 13 lokomotiv. Pro vypnutí ručního řízení je třeba stisknout zároveň tlačítka 1 a 3. Ruční řízení se vypne také, když jednotce přijde jakákoliv pro ni adresovaná CAN zpráva.

Pro zastavení lokomotiv slouží tlačítko 1, pro jízdu vpřed tlačítko 3 a pro jízdu vzad tlačítko 2. Rychlost při ručním řízení má dva stupně. Maximální rychlosti se dosáhne opětovným stiskem.

4.3.4 LED signalizace stavů

LED diody jsou číslovány jako na obrázku 4.7.

Barvy RGB LED diody signalizují následující stavy:

- `Zelená` – režim ovládání pomocí CAN
- `Modrá` – režim ručního ovládání
- `Červená` – výpadek řízení CAN - jednotka více jak 1,5 s nepřijala zprávu

Pro snížení jasu RGB diody je použita PWM modulace, která je prováděna přímo ve smyčce programu. Tím je zároveň zajištěna signalizace plynulého běhu programu. Pokud se program zastaví, RGB dioda zhasne, případně se rozsvítí plným jasem. Stejná modulace je použita pro diodu 2, která pouze signalizuje plynulý běh programu.

Dioda 1 mění svůj stav při každém příjmu zprávy CAN. Při ručním řízení svítí při stisku kteréhokoliv tlačítka a také při nenulové rychlosti lokomotiv.

Dioda 3 mění svůj stav při každém odeslání zprávy DCC watchdog.

5

Závěr

Systém modelového kolejiště byl popsán v kapitole 2. Bylo popsáno jeho blokové schéma a stručně vysvětlena funkce jednotlivých bloků. V podkapitole 2.2 byl popsán princip fungování sběrnice CAN, který se používá pro řízení systému kolejiště. Pro pochopení požadavků na funkci generátoru byl v kapitole 3 rozebrán protokol DCC pro řízení modelové železnice.

Bylo navrženo zařízení na desce plošných spojů o stanovených rozměrech s mikrokontrolérem od firmy NXP. Postup navrhování byl popsán v podkapitole 4.1.

Zařízení bylo realizováno a zprovozněno. Způsob jeho výroby je uveden v podkapitole 4.2. Při výrobě nebylo osazeno rozhraní USB, protože se ukázalo, že není potřebné. Při výrobě jsme si ověřili, že osazování THT součástek ze stejné strany jako SMD součástky na desku bez prokovených otvorů je o stupeň obtížnější, než osazování THT součástek z opačné strany.

Firmware jednotky byl naprogramován v jazyce C. Jeho stručný popis jeho funkce je v podkapitole 4.3. Zdrojové kódy jsou v příloze B.

Funkčnost jednotky byla ověřena na modelovém kolejišti. Jednotka byla úspěšně využita pro generování příkazů pro lokomotivy z programu centrálního řízení na PC. Bylo také vyzkoušeno nouzové ruční řízení pomocí této jednotky. V tomto režimu je možné ovládat lokomotivy bez centrálního řízení. Pro usnadnění budoucí práce s jednotkou byl v podkapitolách 4.3.2 - 4.3.4 popsán způsob používání jednotky.

Do budoucna přichází v úvahu drobné úpravy ve firmware jednotky, které by umožnily její širší uplatnění.

Literatura

[1] Weissar P., Žahour J., Lufinka O.: *Vlaky TT* [online]. Poslední změna 23. 4. 2018, 17:48 [cit. 1. 5. 2018]. Dostupné z: http://projekty.fel.zcu.cz/index.php/Vlaky_TT

[2] Polák K.: *Sběrnice CAN* [online]. Ústav telekomunikací, VUT FEKT Brno. Poslední změna 16. 6. 2003 [cit. 1. 5. 2018]. Dostupné z: http://www.elektrorevue.cz/clanky/03021/index.html

[3] Pinker J.: *Sériové vstupní a výstupní obvody CAN* [online]. Literatura k předmětu KAE/MPP [cit. 1. 5. 2018]. Dostupné z: courseware.zcu.cz/portal/studium/courseware/kae/mpp/studijni-materialy.html

[4] Wikipedia: *CAN bus* [online]. Poslední změna 1. 6. 2018, 18:55. [cit. 10.5.2018]. Dostupné z: https://en.wikipedia.org/wiki/CAN_bus

[5] Copyright Warwick Control Technologies: *CAN Access & Collisions* [online]. Poslední změna 24. 1. 2017 [cit. 10. 5. 2018]. Dostupné z: https://manual.xanalyser.com/CAN%20Access%20&%20Collisions.html

[6] National Model Railroad Association: *Standard S-9.1* [online]. Verze 2006 [cit. 12. 5. 2018]. Dostupné z: https://nmra.org/index-nmra-standards-and-recommended-practices

[7] National Model Railroad Association: *Standard S-9.2* [online]. Vydání červenec 2004 [cit. 12. 5. 2018]. Dostupné z: https://nmra.org/index-nmra-standards-and-recommended-practices

[8] Fulda: *Digitální řízení modelové železnice – DCC* [online]. Poslední změna 28.3.2016 [cit. 12. 5. 2018]. Dostupné z: http://robodoupe.cz/2016/digitalni-rizeni-modelove-zeleznice-dcc/

[9] Fulda: *Digitální řízení modelové železnice – DCC, 2. část* [online]. Poslední změna 30. 3. 2016 [cit. 12. 5. 2018]. Dostupné z: http://robodoupe.cz/2016/digitalni-rizeni-modelove-zeleznice-dcc-2-cast/

[10] Černý M.: *Připojování boosterů a úsekové dělení – část 1* [online]. Poslední změna 4. 2. 2016 [cit. 14. 5. 2018]. Dostupné z: http://www.modulybrno.cz/pripojovani-boosteru-1

[11] Tošovský P.: *Úvod do Altium Designeru I.* [online]. Poslední změna 27. 3. 2010, 21:32 [cit. 20. 5. 2018]. Dostupné z: https://vyvoj.hw.cz/navrh-obvodu/software/uvod-do-altium-designeru-i.html

[12] Freescale Semiconductor, Inc.: *MC9S08DZ128 Series Data Sheet with Addenda* [online]. Poslední změna 7. 2. 2015 [cit. 20. 5. 2018]. Dostupné z: http://cz.farnell.com/nxp/mc9s08dz96clf/mcu-8bit-s08-40mhz-lqfp-48/dp/1622707

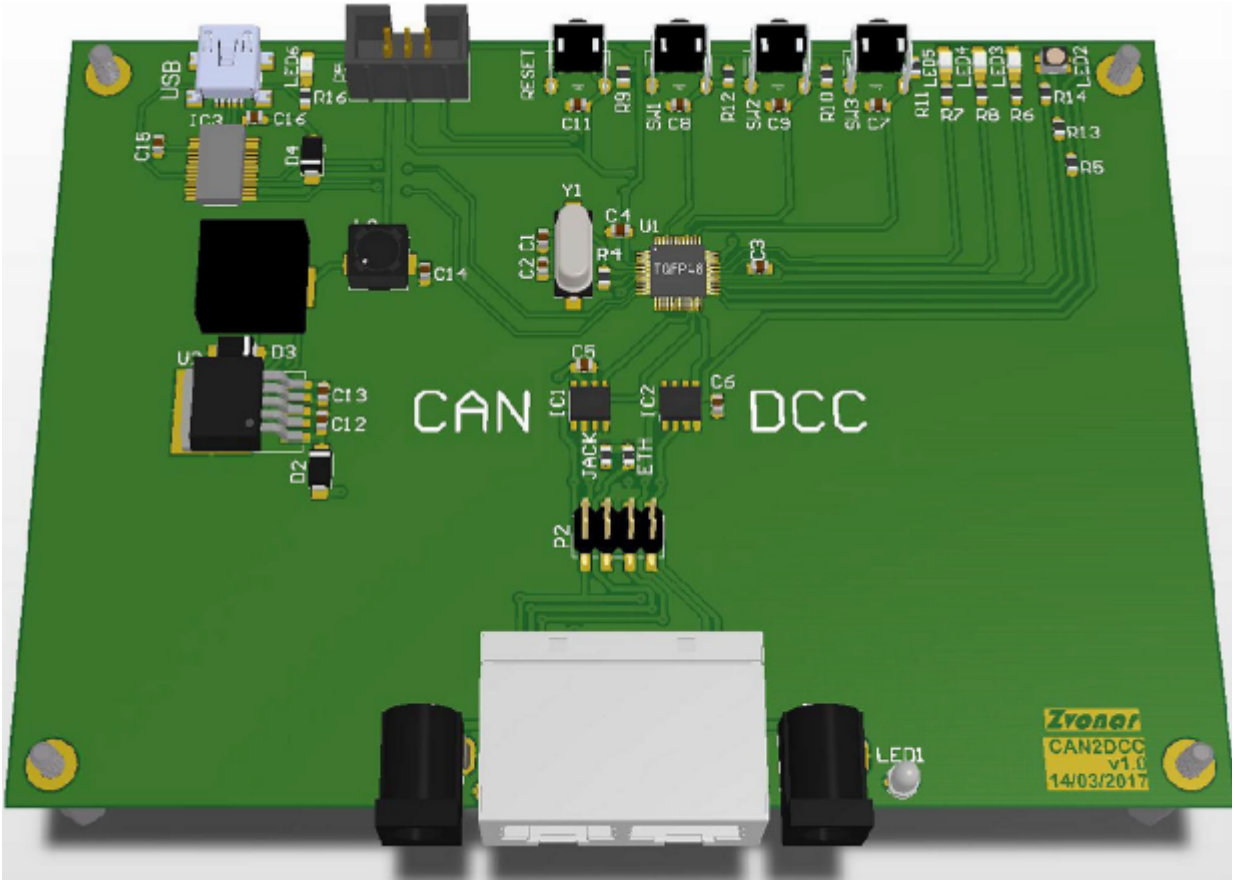
[13] Warehouse Cables: *RJ45 Connector Pin-out* [online]. [cit. 24. 5. 2018]. Dostupné z: http://www.modulybrno.cz/pripojovani-boosteru-1

[14] Electro Mechanical Components: *Modular Jacks & Plugs* [online]. [cit. 24. 5. 2018]. Dostupné z: https://www.emc.de/en/products/io-connectors/modular-jacks-plugs.html

[15] Vojáček A.: *Program běžícího textu pro MCU MC9S08LC60* [online]. Poslední změna 22. 10. 2007, 8:03 [cit. 25. 5. 2018]. Dostupné z: vyvoj.hw.cz/teorie-a-praxe/program-beziciho-textu-pro-mcu-mc9s08lc60.html

Příloha A

3D model jednotky generátoru



Obr. A.1: Snímek 3D modelu jednotky

Příloha B

Výpis zdrojového kódu

B.1 main.h

```
1
2  /*
3   * main.h
4   *
5   *  Created on: Nov 13, 2017
6   *      Author: Filip
7   */
8
9  #ifndef MAIN_H_
10 #define MAIN_H_
11
12 #ifdef MAIN_C_
13 #define EXT
14 #else
15 #define EXT extern
16 #endif
17
18 void send_CAN(uint32_t ID, unsigned char extended_ID, unsigned char *pt_data,unsigned char length_data);
19
20 void read_CAN();
21
22 #define DEFAULT_CAN_WATCHDOG_DELAY                1500  //1500ms = 1.5s
23 #define DEFAULT_DCC_WATCHDOG_DELAY                1024  //100ms = 1s
24
25 /*globalni promenne*/
26
27 EXT uint16_t CAN_wdg;
28 EXT uint16_t DCC_wdg;
29 EXT uint16_t DCC_wdg_delay;
30
31 EXT uint8_t CAN_ziju;
32 EXT uint8_t manual;
33 EXT uint8_t CAN_prijem;
34 EXT uint8_t CAN_rxrdy;
35 EXT uint8_t CAN_RX[13];
36 EXT uint8_t CAN_DATA[8];
37 EXT uint16_t CAN_id;
38 EXT uint8_t CAN_dlr;
39 EXT uint8_t CAN_typ;
40 EXT uint8_t CAN_stitek;
41 EXT uint8_t DCC_flag;
42 EXT uint16_t DCC_DATA[27][5];
43 EXT uint8_t DCC_i;
44 EXT uint8_t DCC_j;
45 EXT uint16_t DCC_VLAK[27][3];
46 EXT uint8_t DCC_id;
47
48 #define TYP 1
49 #define STITEK 1
```

```
50 #define ledR PTDD_PTDD2
51 #define ledG PTDD_PTDD3
52 #define ledB PTDD_PTDD4
53 #define led1 PTDD_PTDD7
54 #define led2 PTDD_PTDD6
55 #define led3 PTDD_PTDD5
56 #define btn1 PTBD_PTBD4
57 #define btn2 PTBD_PTBD3
58 #define btn3 PTBD_PTBD2
59
60 #endif /* MAIN_H_ */
61
```

B.2 main.c

```
1
2  /* ###################################################################
3   **     Filename    : main.c
4   **     Project     : CAN2DCC
5   **     Processor   : MC9S08DZ96CLF
6   **     Version     : Driver 01.12
7   **     Compiler    : CodeWarrior HCS08 C Compiler
8   **     Date/Time   : 2017-10-30, 14:57, # CodeGen: 0
9   **     Abstract    :
10  **         Main module.
11  **         This module contains user's application code.
12  **     Settings    :
13  **     Contents    :
14  **         No public methods
15  **
16  ** ###################################################################*/
17
18 /* Including needed modules to compile this module/procedure */
19 #include "Cpu.h"
20 #include "Events.h"
21 #include "PTD.h"
22 #include "PTB.h"
23 #include "MSCAN.h"
24 #include "TPM2.h"
25 #include "TPM3.h"
26 /* Include shared modules, which are used for whole project */
27 #include "PE_Types.h"
28 #include "PE_Error.h"
29 #include "PE_Const.h"
30 #include "IO_Map.h"
31
32 /* User includes (#include below this line is not maintained by Processor Expert) */
33
34 #define MAIN_C_
35 #include "main.h"
36
37 void send_CAN(uint32_t ID, unsigned char extended_ID, unsigned char *pt_data,
38                 unsigned char length_data) {
39         unsigned char * pt_TX_data_buff;
40         unsigned char i;
41
42         CANTBSEL = 0x01;
43
44         //wait for empty buffer
45         if (!CANTFLG_TXE0) //buffer full
46         {
47                 //wait until next call
48                 return;
49         }
50
51         if (extended_ID) {
52                 //set transmit ID
53                 CANTIDR0 = ID >> 21;
54                 CANTIDR1 = ((ID >> 15) | 0xF8) & ((ID >> 13) | 0x0F);
55                 CANTIDR2 = ID >> 7;
```

```
56                  CANTIDR3 = (ID << 1) + 1;
57          } else {
58                  //set transmit ID
59                  CANTIDR0 = ID >> 3;
60                  CANTIDR1 = ID << 5;
61          }
62
63          //set length of data frame
64          CANTDLR = length_data;
65
66          //preparing or variables for transmitter buffer filling
67          pt_TX_data_buff = &CANTDSR0;
68          i = 0;
69
70          //filling of transmit data buffer, the maximal number of bytes is 8
71          while (i < 8) {
72                  if (i >= length_data) {
73                          break;
74                  }
75
76                  *(pt_TX_data_buff + i) = *(pt_data + i);
77                  i++;
78          }
79
80          //CANTFLG_TXE0 = 1; //data ready for transmitting from buffer 0
81          CANTFLG = 0x01;
82  }
83
84  void read_CAN() {
85
86          int i;
87
88          //test ze can zprava neni EXTENDED ID
89          if (!(CAN_RX[1] & 0x08)) { //CAN_RX[1] ~ CANIDR1 ... 3. bit je IDE
90
91                  CAN_dlr = CAN_RX[12];
92
93                  CAN_id = CANRIDR0;
94                  CAN_id = (CAN_id << 3);
95                  CAN_id = CAN_id + (CANRIDR1 >> 5);
96                  CAN_typ = (CANRIDR0 >> 4);
97                  CAN_stitek = (uint8_t) CAN_id & 0x7f; //CAN_stitek se asi nikde nepouziva
98
99                  for (i = 0; i < 8; i++) {
100                         CAN_DATA[i] = CAN_RX[i + 4];
101                 }
102         }
103         CAN_rxrdy = 0;
104 }
105
106 void send_DCC() {
107
108         int i;
109         //zapis z predbezne matice
110         for (i = 0; i < 27; i++) {
111                 DCC_DATA[i][3] = DCC_VLAK[i][1];
112                 DCC_DATA[i][4] = DCC_VLAK[i][2];
113         }
114
115         TPM2C0V = 60;
116         DCC_flag = 1;
117         DCC_i = 0;
118         DCC_j = 0;
119 }
120 void main(void) {
121
122         /* Write your local variable definition here */
123         int i = 0;
124         int bt1 = 0;
125         int bt2 = 0;
126         int bt3 = 0;
127         int zastav_vlaky = 0;
128         unsigned char data[8];
```

```
129         uint32_t ID;
130         CAN_rxrdy = 0;
131         led1 = 1;
132         DCC_wdg_delay = DEFAULT_DCC_WATCHDOG_DELAY;
133
134         /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
135         PE_low_level_init();
136         /*** End of Processor Expert internal initialization.                    ***/
137
138         /* Write your code here */
139
140 //udaje pro watchdog CAN
141         data[0] = 0x55;
142         ID = 0x0101;
143
144 //nastaveni adres vlaku
145         DCC_DATA[0][2] = 0x003 << 1; //Brejlovec
146         DCC_DATA[1][2] = 0x005 << 1; //Desiro (Kamera)
147         DCC_DATA[2][2] = 0x006 << 1; //Taurus Railion
148         DCC_DATA[3][2] = 0x007 << 1; //DB204.274-5
149         DCC_DATA[4][2] = 0x009 << 1; //Ragulin
150         DCC_DATA[5][2] = 0x00a << 1; //ICE
151         DCC_DATA[6][2] = 0x00b << 1; //Taurus DHL
152         DCC_DATA[7][2] = 0x00c << 1; //Herkules Priessnitz
153         DCC_DATA[8][2] = 0x00d << 1; //Para 555
154         DCC_DATA[9][2] = 0x00e << 1; //T334 Rosnicka
155         DCC_DATA[10][2] = 0x00f << 1; //Desiro DB642.133-3
156         DCC_DATA[11][2] = 0x010 << 1; //Taurus EVB
157         DCC_DATA[12][2] = 0x011 << 1; //ES363
158
159 //nastaveni adres vlaku pro periferie
160         for (i = 13; i < 26; i++) {
161                 DCC_DATA[i][2] = DCC_DATA[i - 13][2];
162         }
163
164 //posledni radek s adresou FF (idle packet) pro pouziti s jinymi adresami
165         DCC_DATA[26][2] = 0xFF;
166
167         for (i = 0; i < 27; i++) {
168
169                 //preambule
170                 DCC_DATA[i][0] = 0x1FF;
171                 DCC_DATA[i][1] = 0x1FE;
172
173                 //zakladni rychlost nebo stav svetel
174                 //jizda 01DCSSSS prislusenstvi 100LFFFF (vzdy posunute << 1)
175                 DCC_DATA[i][3] = (i < 13) ? 0x080 : 0x120;
176
177                 //XOR
178                 DCC_DATA[i][4] =
179                                 (((DCC_DATA[i][2] ^ DCC_DATA[i][3]) & 0x01FE) | 0x0001);
180
181                 //nastaveni matice/bufferu DCC_VLAK
182                 DCC_VLAK[i][0] = DCC_DATA[i][2];
183                 DCC_VLAK[i][1] = DCC_DATA[i][3];
184         }
185
186         while (1) {
187
188                 if (DCC_wdg % 20 == 0) //SW PWM ledek indikujicich práci programu
189                                 {
190                         led2 = 0;                 //program funguje = led2
191                         if (manual)
192                                 ledB = 0;         //manual mod = modra
193                         else if (CAN_wdg)
194                                 ledG = 0;         //CAN mod = zelena
195                         else
196                                 ledR = 0;         //vypadek CAN = cervena
197                 } else {
198                         led2 = 1;
199                         ledR = 1;
200                         ledG = 1;
201                         ledB = 1;
```

```
202                 }
203
204                 if (manual) {                               //indikace stavu pri manual modu pomoci led1
205                         if (bt1 | bt2 | bt3)         //stisk tlacitka
206                                 led1 = 0;
207                         else if (((DCC_VLAK[0][1] >> 1) & 0x0F) == 0x0F)         //max rychlost
208                                 led1 = 0;
209                         else if (((DCC_VLAK[0][1] >> 1) & 0x0F) != 0 && DCC_wdg % 20 == 0) //nenulova rychlost
210                                 led1 = 0;
211                         else
212                                 led1 = 1;
213                 }
214
215                 if (!CAN_wdg && !manual) {
216                         //zastavit vse pri vypadku CAN v CAN modu
217                         for (i = 0; i < 13; i++) {
218                                 DCC_VLAK[i][1] = 0x0C0; //zastavit //0x0FE; //naplno vpred
219                                 DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1]) & 0x01FE)
220                                                 | 0x0001);
221                         }
222                 }
223
224                 if (CAN_rxrdy) {
225
226                         CAN_wdg = DEFAULT_CAN_WATCHDOG_DELAY;
227                         led1 = !led1;                         //indikace prijmu CAN pomoci led1
228                         manual = 0;
229                         read_CAN();
230
231                         //nastaveni DCC watchdog
232                         if (CAN_DATA[0] == 0x00) {
233                                 DCC_wdg_delay = (((uint16_t) CAN_DATA[1] << 4) + 0xF);
234                 //základní 'broadcast' adresa zablokována pro zabránění opakovanému přepisování příkazů
235                 //využita pro nastavení frekvence odesílání watchdog zpráv
236
237                         } else if (CAN_DATA[0] == 0x01) {
238                                 //'broadcast' pokud adresa vlaku = 1
239                                 if ((CAN_DATA[1] >> 6) == 0x01) {         //pokud prikaz pro vlaky
240                                         for (i = 0; i < 13; i++) {
241                                                 //zapsat do všech 13 vlaků
242                                                 DCC_VLAK[i][1] =
243                                                                 ((((uint16_t) CAN_DATA[1]) & 0xE0) << 1)
244                                                                         + (CAN_DATA[1] & 0x1E)
245                                                                         + ((CAN_DATA[1] & 0x01) << 5);
246                                                 DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1])
247                                                                 & 0x01FE) | 0x0001);
248                                         }
249                                 } else {
250                                         for (i = 13; i < 26; i++) {
251                                                 //zapsat do všech 13 periferií
252                                                 DCC_VLAK[i][1] = ((uint16_t) CAN_DATA[1]) << 1;
253                                                 DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1])
254                                                                 & 0x01FE) | 0x0001);
255                                         }
256                                 }
257
258                         } else if ((CAN_DATA[0] <= 0x7F) && (CAN_DATA[1] >> 6) == 0x01) {
259                                 //standartni prikaz pro lokomotivu
260                                 for (i = 0; i < 13; i++) {
261                                         //zapsat pro konkretni 1 vlak z 13
262                                         if (DCC_VLAK[i][0] == (((uint16_t) CAN_DATA[0]) << 1)) {
263                                                 DCC_VLAK[i][1] =
264                                                                 ((((uint16_t) CAN_DATA[1]) & 0xE0) << 1)
265                                                                         + (CAN_DATA[1] & 0x1E)
266                                                                         + ((CAN_DATA[1] & 0x01) << 5);
267                                                 DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1])
268                                                                 & 0x01FE) | 0x0001);
269                                         }
270                                 }
271
272                         } else if ((CAN_DATA[0] <= 0x7F) && (CAN_DATA[1] >> 5) == 0x04) {
273                                 //prikaz pro periferie lokomotiv
274                                 for (i = 13; i < 26; i++) {
```

```
275                                     //zapsat pro konkretni 1 vlak z 13
276                                     if (DCC_VLAK[i][0] == (((uint16_t) CAN_DATA[0]) << 1)) {
277                                             DCC_VLAK[i][1] = ((uint16_t) CAN_DATA[1]) << 1;
278                                             DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1])
279                                                     & 0x01FE) | 0x0001);
280                                     }
281                             }
282
283                     } else if (CAN_DATA[0] > 0x7F) {
284                             //prikaz s adresou mimo adresy dekoderu 13 lokomotiv
285                             DCC_VLAK[26][0] = (((uint16_t) CAN_DATA[0]) << 1);
286                             DCC_VLAK[26][1] = ((uint16_t) CAN_DATA[1]) << 1;
287                     }
288             } /*if (CAN_rxrdy) */
289
290             if (btn1 == 0) {
291                     if (bt1 == 0) {
292                             manual = 1;
293                             for (i = 0; i < 13; i++) {
294                                     DCC_VLAK[i][1] = 0x0C0; //zastavit
295                                     DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1])
296                                             & 0x01FE) | 0x0001);
297                             }
298                     }
299                     bt1 = 1;
300             } else {
301                     bt1 = 0;
302             }
303             if (btn2 == 0) {
304                     if (bt2 == 0) {
305                             if (manual) {
306                                     for (i = 0; i < 13; i++) {
307                                             if (DCC_VLAK[i][1] > 0x09E)
308                                                     DCC_VLAK[i][1] = 0x080; //nula na zpatecce
309                                             if (DCC_VLAK[i][1] == 0x080)
310                                                     DCC_VLAK[i][1] = 0x8E; //stredni rychlost
311                                             else if (DCC_VLAK[i][1] == 0x08E)
312                                                     DCC_VLAK[i][1] = 0x09E; //naplno vzad
313                                             DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1])
314                                                     & 0x01FE) | 0x0001);
315                                     }
316                             } else {
317                                     if (DCC_wdg_delay < 1024)
318                                             DCC_wdg_delay *= 2;
319                             }
320                     }
321                     bt2 = 1;
322             } else {
323                     bt2 = 0;
324             }
325
326             if (btn3 == 0) {
327                     if (bt3 == 0) {
328                             if (bt1) {
329                                     manual = 0;
330                             } else if (manual) {
331                                     for (i = 0; i < 13; i++) {
332                                             if (DCC_VLAK[i][1] < 0x0C0)
333                                                     DCC_VLAK[i][1] = 0x0C0; //nula vpred
334                                             if (DCC_VLAK[i][1] == 0x0C0)
335                                                     DCC_VLAK[i][1] = 0x0CE; //stredne vpred
336                                             else if (DCC_VLAK[i][1] == 0x0CE)
337                                                     DCC_VLAK[i][1] = 0xDE; //naplno vpred
338                                             DCC_VLAK[i][2] = (((DCC_VLAK[i][0] ^ DCC_VLAK[i][1])
339                                                     & 0x01FE) | 0x0001);
340                                     }
341                             } else {
342                                     if (DCC_wdg_delay > 32)
343                                             DCC_wdg_delay /= 2;
344                             }
345                     }
346                     bt3 = 1;
347             } else {
```

```
348                         bt3 = 0;
349                 }
350
351                 //opakovani odesilani DCC
352                 if (DCC_flag == 0) {
353                         send_DCC();
354                 }
355
356                 if (!DCC_wdg) {
357                         DCC_wdg = DCC_wdg_delay; //watchdog timer
358                         send_CAN(ID, 0, data, 1);
359                         led3 = !led3;
360                 }
361         }
362
363 //wait for empty buffer
364
365         /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
366         /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
367 #ifdef PEX_RTOS_START
368         PEX_RTOS_START(); /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
369 #endif
370         /*** End of RTOS startup code.  ***/
371         /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
372         for (;;) {
373         }
374         /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
375 } /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
376
377
```

B.3 events.c

```
1
2  /* ###################################################################
3   **     Filename    : Events.c
4   **     Project     : CAN2DCC
5   **     Processor   : MC9S08DZ96CLF
6   **     Component   : Events
7   **     Version     : Driver 01.02
8   **     Compiler    : CodeWarrior HCS08 C Compiler
9   **     Date/Time   : 2017-10-30, 14:57, # CodeGen: 0
10  **     Abstract    :
11  **         This is user's event module.
12  **         Put your event handler code here.
13  **     Settings    :
14  **     Contents    :
15  **         No public methods
16  **
17  ** ###################################################################*/
18
19 #include "Cpu.h"
20 #include "Events.h"
21
22 /* User includes (#include below this line is not maintained by Processor Expert) */
23 #include "main.h"
24
25 #ifdef ISR_IN_NONBANKED
26 #pragma CODE_SEG __NEAR_SEG NON_BANKED
27 #endif
28 /*
29  ** ===================================================================
30  **     Interrupt handler : RXCAN
31  **
32  **     Description :
33  **         User interrupt service routine.
34  **     Parameters  : None
35  **     Returns     : Nothing
36  ** ===================================================================
37  */ISR(RXCAN) {
```

```
38          int i;
39          if (!CAN_rxrdy) {
40                  for (i = 0; i < 13; i++) {
41                          CAN_RX[i] = *(&CANRIDR0 + i);
42                  }
43                  CAN_rxrdy = 1;
44          }
45          CANRFLG_RXF = 1;
46  }
47
48  #ifdef ISR_IN_NONBANKED
49  #pragma CODE_SEG DEFAULT
50  #endif
51
52  #ifdef ISR_IN_NONBANKED
53  #pragma CODE_SEG __NEAR_SEG NON_BANKED
54  #endif
55  /*
56   ** ===================================================================
57   **     Interrupt handler : TPM2_int
58   **
59   **     Description :
60   **         User interrupt service routine.
61   **     Parameters  : None
62   **     Returns     : Nothing
63   ** ===================================================================
64   */ISR(TPM2_int) {
65          if (DCC_flag) {
66                  if (DCC_DATA[DCC_j][DCC_i / 9] & (0x100 >> (DCC_i % 9))) {
67                          //TPM2MOD = 119;
68                          TPM2SC_PS0 = 0;
69                          //TPM2C0V = 60;
70                  } else {
71                          //TPM2MOD = 239;
72                          TPM2SC_PS0 = 1;
73                          //TPM2C0V = 120;
74                  }
75                  DCC_i++;
76                  if (DCC_i == 45) {
77                          DCC_i = 0;
78                          DCC_j++;
79                          //TPM2C0V = 0; //zakomentovano zmena
80                  }
81                  if (DCC_j == 27) {
82                          DCC_flag = 0;
83                  }
84          } else {
85                  TPM2SC_PS0 = 0;
86          }
87          TPM2SC_TOF = 0;
88  }
89  #ifdef ISR_IN_NONBANKED
90  #pragma CODE_SEG DEFAULT
91  #endif
92
93  #ifdef ISR_IN_NONBANKED
94  #pragma CODE_SEG __NEAR_SEG NON_BANKED
95  #endif
96  /*
97   ** ===================================================================
98   **     Interrupt handler : TPM3OVF
99   **
100  **     Description :
101  **         User interrupt service routine.
102  **     Parameters  : None
103  **     Returns     : Nothing
104  ** ===================================================================
105  */ISR(TPM3OVF) {
106         if (CAN_wdg) {
107                 CAN_wdg--;
108         }
109         if (DCC_wdg) {
110                 DCC_wdg--;
```

```
111         }
112         TPM3SC_TOF = 0;
113 }
114 #ifdef ISR_IN_NONBANKED
115 #pragma CODE_SEG DEFAULT
116 #endif
117
118 /* END Events */
119
120 /*!
121  ** @}
122  */
123 /*
124  ** ###################################################################
125  **
126  **     This file was created by Processor Expert 10.3 [05.09]
127  **     for the Freescale HCS08 series of microcontrollers.
128  **
129  ** ###################################################################
130  */
131
132
```