FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Diplomová práce

Elektronická řídicí jednotka samořízeného modelu vozu

Studijní program:	N2612 Elektrotechnika a informatika
Studijní obor:	Informační technologie
Autor práce:	**Bc. Petr Vošoust**
Vedoucí práce:	Ing. Jan Koprnický, Ph.D. Ústav mechatroniky a technické informatiky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání diplomové práce

Elektronická řídicí jednotka samořízeného modelu vozu

Jméno a příjmení:	**Bc. Petr Vošoust**
Osobní číslo:	M20000177
Studijní program:	N2612 Elektrotechnika a informatika
Studijní obor:	Informační technologie
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	2021/2022

Zásady pro vypracování:

1. Proveďte rozbor aktuální elektronické řídicí jednotky automaticky řízeného vozu na autodráhu.
2. Na základě rozboru navrhněte nový HW řídicí jednotky.
3. Naprogramujte základní SW rutiny pro přímou komunikaci s HW a navrhněte řešení systémové části.
4. Vytvořte funkce umožňující jednoduché programování zařízení.
5. Na laboratorní dráze ověřte funkčnost celého zařízení zvoleným algoritmem řízení.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	40–50 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] HONS, Petr. Elektronická řídicí jednotka pro účely samořízeného modelu auta: Electronic control unit for driverless car model. Liberec: Technická univerzita v Liberci, 2017. Diplomové práce. Technická univerzita v Liberci. Vedoucí práce Petr Mrázek.

[2] DIETSCHE, Karl-Heinz a Konrad REIF, ed. Automotive Handbook. 10th edition, revised and extended. Karlsruhe: Robert Bosch, 2018, 1750 s. ISBN 978-111-9530-817.

[3] MCGRATH, Michael E. Autonomous vehicles: opportunities, strategies, and disruptions. Spojené státy americké: [nákladem vlastním], 2018. ISBN 978-1-9803-1385-4.

Vedoucí práce:	Ing. Jan Koprnický, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	22. května 2023

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci dne 12. října 2021

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

30. května 2023 Bc. Petr Vošoust

Elektronická řídicí jednotka samořízeného modelu vozu

Abstrakt

Tato diplomová práce se zabývá možnou renovací tématu ročníkové práce na bakalářském studiu FM. Navrhuje postup a možné řešení jak obnovit téma v nabídce ústavu MTI. Nejdříve se zaměřuje na nahrazení zastaralého a nepodporovaného HW – ECU. Dále navrhuje HW nový v podobě nové platformy ECU a to včetně možných úprav či rozšíření. Nový hardware zprovozňuje na základě ukázkového SW a jeho popisu – navrhuje postup při založení nového jádra programu, jeho samotných rutin a užití metodiky návrhu. S ECU navazuje kontakt pomocí BLE a mobilní aplikace. V závěru práce se testuje funkčnost řešení.

Klíčová slova: Model vozu na autodráhu, elektronická řídící jednotka, bezpečnostní norma ČSN EN 61508-3, mobilní aplikace, bluetooth low energy, arduino, android, algoritmus řízení

Electronic control unit for self-driving car model

Abstract

This diploma thesis deals with the possibilities of renovation of the subject of the qualification work (double semestral project) at the bachelor's degree programme at FM. It proposes a procedure and a possible solution to renew the offered topic at the MTI institute project board. It first focuses on replacing outdated and unsupported HW. Then proposes new HW in the form of a new ECU platform, including possible modifications or extensions suggestions. Then getting the new hardware into operation on the basis of a sample SW and its description – suggesting a procedure for establishing a new framework, its routines and the use of the design methodology. It establishes contact with the ECU using BLE and a mobile application. At the end of the work, the functionality of the solution is tested.

Keywords: Slot car, electronic control unit, safety standard ČSN EN 61508-3, mobile application, bluetooth low energy, arduino, android, control algorithm

Poděkování

Rád bych poděkoval všem, kteří přispěli ke vzniku této práce a to pak jmenovitě vedoucímu práce, za doplnění mezer v praxi a vzdělání, vedení a vysoké nároky na dokumentaci.
Dále pak zejména své životní partnerce, Petře Staňkové, že měla a stále má se mnou trpělivost. Skloubit podnikání, studium a osobní život byla stresující disciplína, kterou bych bez ní nezvládl – bez ceny ztráty jednoho ze změněných aspektů.

Obsah

Seznam zkratek

ACK	Acknowledge – zpravidla potvrzující zpráva
API	APlication Interface – aplikační rozhraní
APK	Android Aplication Package – instalační balíček pro operační systém Android
BLE	Bluetooth Low Energy – moderní bluetooth technologie s ohledem na nízkou spotřebu
ECU	Electronic Control Unit – (elektronická) řídicí jednotka
FM	Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci
FSM	Finite State Machine – (konečný) stavový automat
HW	Hardware
IDE	Integrated Development Environment – (integrované) vývojové prostředí
ITE	Ústav informačních technologií a elektroniky na Technické univerzitě v Liberci
IT	Informační technologie
LED	Light-Emitting Diode – světelná dioda
MCU	MicroController Unit – mikrokontroler
MTI	Ústav mechatroniky a technické informatiky na Technické univerzitě v Liberci
NTI	Ústav nových technologií a aplikované informatiky na Technické univerzitě v Liberci
OOP	Object-Oriented Programming – objektově orientované programování
OS	Operation System – operační systém
OSVČ	Osoba samostatně výdělečně činná
PWM	Pulse Width Modulation – pulzně šířková modulace
R/W	Read or Write – operace čtení nebo zápisu
SD	Secure Digital – použito ve smyslu paměťové karty
SW	Software
TUL	Technická univerzita v Liberci
USB	Universal Serial Bus – sériová sběrnice
VŠ	Vysoká škola (institut pro vzdělání a získání akademického titulu)

Úvod

Motivací pro zvolení téma diplomové práce jsou dva pilíře.
Jednak bych chtěl, aby téma práce mělo smysl, užitek a přínos, aby si případný čtenář mohl odnést poznatky, know-how, způsob myšlení či adaptaci metodologie postupu. Též bych chtěl, aby případný uživatel výsledků a náplně diplomové práce plně využil k vlastní tvorbě.
Druhým pilířem je zvolení si samotného tématu/náplně a to bylo obnovit jedno z tradičních témat ročníkových (popř. ročníkových) projektů pro studenty bakalářského Informačních technologii, kteří si vybrali zastřešující ústav pro roč. projekt u ústavu MTI.
S vedoucím této práce sdílíme představu, že téma projektu bylo zajímavé a jeho renovace má smysl.
Téma ročníkové práce bude rozebráno v samostatné kapitole, ale náznakem bych popsal, že se jednalo o návrh řídicího algoritmu pro model vozidla na autodráhu a cílem byla co největší autonomnost řízení. Byl k tomu vymezen dedikovaný hardware a byla dostupná autodráha, na které lze algoritmus testovat a ladit.
Bohužel téma s odstupem času zastaralo stejně jako samotný HW, na kterém se projekt realizoval. Nebyl dostupný (podpora skončila) a tak nevyhnutelně zmizelo téma projektu v nabídce MTI. Má práce popisuje návrh pro renovaci tohoto projektu a demonstruje na kódu mj. i ukázku užití a implementaci jednoduchého řídicího algoritmu – možnou cestu řešení studenta, který by si v budoucnu projekt vybral.
Práce by měla sloužit i jako odrazový můstek pro seberealizaci studenta.

1 Ročníkový projekt

1.1 Historie projektu

Ročníkový projekt, pod názvem ne nepodobný tématu této práce, byl uveden na ústavu MTI (na *TUL*) jako jedna z možností výběru tématu pro studenty (i skupiny studentů) bakalářského studia oboru Informační technologie zhruba v roce 2008. Studenti univerzity měli v té době povinně volitelný ročníkový projekt (pod zkratkou *PRJ* a to v druhém ročníku studia z celkových tří) a mohli si vybrat ze 4 ústavů, které v té době byly pro studenty dostupné.

Obrázek 1.1: Logo TUL z roku 2012

Obrázek 1.2: Logo FM z roku 2012

Každý ústav měl různorodá témata – každý podle zaměření ústavu. Např. ústav NTI nebo ITE se zaměřovala spíše na modernější technologie a zejména vyšší programovací jazyky a témata s nimi spojovaná (bez hyperboly by potřeboval student na více než polovinu z nich znalost Matlabu). MTI na druhou stranu nabízela témata, která mají blíže k hardwaru a elektrotechnickým až mechatronickým principům. Tím bych nerad vsugeroval čtenáři, že MTI je v jakémkoli ohledu zastaralý ústav. Naopak. Jen je jeho zaměření více elektrotechnické a je to takový průnik elektra, IT a strojírenství.
Já, jako student bakalářského studia nastupující v roce 2011 jsem byl vždycky fascinován videohrami a robotikou. Vývoj a principy fungování videoher v té době nebylo příliš rozšířeným tématem (myšleno v univerzitním prostředí) a tak ve studijním roce 2012/2013 jsme si ve dvoučlenném týmu zvolili právě ústav MTI

a popisovaný projekt pod vedením pana Ing. Jana Koprnického, Ph.D. Učinili tak i další 3 skupiny – tato informace se zdá, že má nízkou informativní hodnotu, ale dále bude hrát roli jako kuriozita a demonstrace "nedostatků" a argument, pro další motivaci diplomové práce.

Odkaz na webové stránky ústavu MTI

Odkaz na webové stránky ústavu NTI

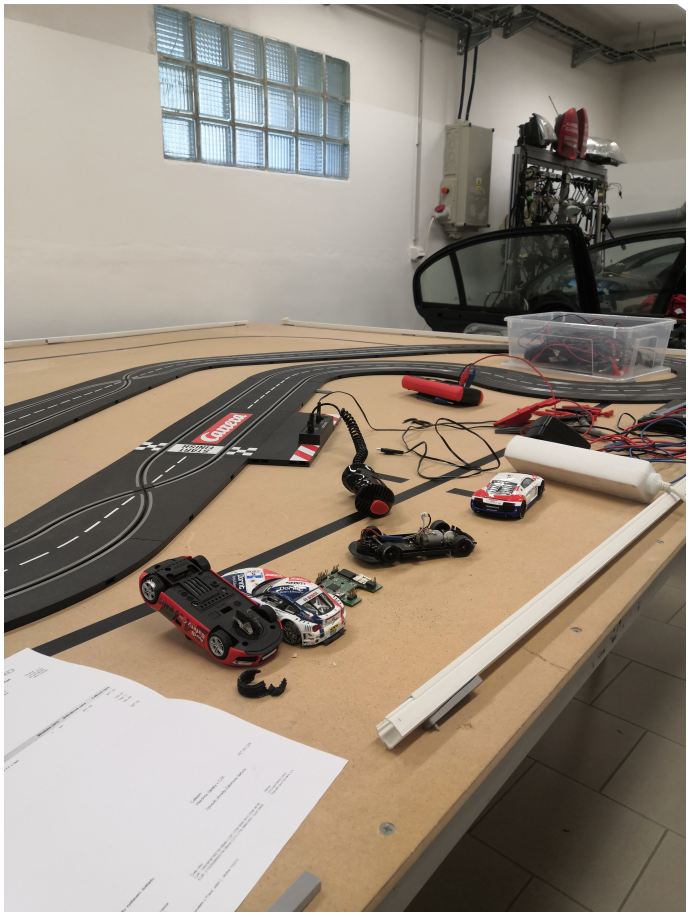
Odkaz na webové stránky ústavu ITE

1.2 Zadání projektu

Zadáním projektu bylo fiktivní účast v soutěži pro studenty technických univerzit, které jsou součástí tzv. EMEA – Europe Middle East Asia, spolupráce univerzit v těchto oblastech. Taktéž samozřejmě i splnění zadání soutěže pro účast. Více o soutěži v následuící kapitole 1.3.
Soutěž studentů nesla název Freescale Race Challenge 2009, později s iterací (názvu) roku pro každý následující rok až do 2012 Každá iterace měla mírně jiné podmínky pro účast, ale smysl zůstával stejný. Já osobně se řídil pravidly pro rok 2011. Materiály jsou stále dostupné online a lze je dohledat na internetu. Soutěž (vícero soutěží obecně) zastřešovala společnost Freescale Semiconductors, která v současné době neexistuje. Resp. je nyní součástí NXP Semiconductors od roku 2015 [1]. Hardware byl zcela v režii společnosti Freescale a nyní není dostupný.

1.3 Soutěž FRC

V hrubém popisu náplně bych soutěž popsal jako návrh autonomního algoritmu řízení modelu vozidla na autodráze. Tj. skupina studentů (i třeba jednočlenná) má za úkol na doručeném hardwaru vytvořit takový algoritmus, aby byla interakce s uživatelem blížící se nule – tj. velké zaměření na automnost vozidla potažmo algoritmu. Aby si vozidlo zmapovalo neznámou dráhu a projelo ji co nejefektivněji (v tomto případě nejrychleji – s určitou parametrizací, kterou můžeme nyní zanedbat).
K dispozici měla soutěžící skupina díly pro dráhu (viz ilustrace 1.3), startovní díl s joysticky pro řízení napětí v drážce dráhy (až 14 V) a model vozidla. Model vozidla se lišil podle roku iterace. Byl např. dostupný model BMW Z4 (ilustrace 1.4) a Audi R8 (v měřítku 1:30). Byla zde deska plošných spojů a elektromotor, který byl napájený z dráhy přes regulátor na desce. Deska byla vybavena dalšími periferiemi a moduly jako akcelerometr, čtečka SD karty, H-můstek…v mém případě jsem měl také k dispozici schéma desky, návod na úpravu karoserie a vnitřku vozidla a vývojové prostředí Codewarrior, ve kterém se kompiloval kód a nahrával do vozidla přes USB.



Obrázek 1.3: Sestavená dráha

1.4 Ukončení projektu na MTI

Na ukončení nabídky projektu na ústavu MTI se podílo několik faktorů, u kterých nejsem schopen číselně vyjádřit jejich zastoupení, nicméně jejich součet je "měřitelný" (v booleovském smyslu projekt není nabízen). Zde vypíši faktory:

- Nízký počet studentů – u statistiky (ze systému Stag) je vidět výrazně nižší počet studentů u oboru Informační technologie od roku 2015 Jestli se jedná o nižší zájem o obor Informační technologie, technické vědy obecně, studium na TUL…To nechám čistě na čtenáři. Zájem o studium na VŠ naopak (z mého osobního pohledu) a zejména pak humanitární a jednodušší obory významně trendově roste, stejně jako klesá zájem o řemeslné obory. Ale to je spíše spekulace.
- Nedostupnost HW – je dostupný omezený počet kusů HW, zejména desek uvnitř vozidla. Výrobce a dodavatel projekty již nezastřešuje, HW se nevyrábí. To přináší obavy a nejistoty.
- Nízký zájem studentů – opět spíše subjektivní pohled, ale trendově jsou na vzestupu spíše modernější technologie než konvenční programování embedded.

A i to se v dnešní době přesouvá pod mikrokontroléry programované v abstraktních vrstvách (Python apod.).

- Poruchovost – desky jsou dedikované pro soutěž, ale mají svá úskalí. Dráha je napájená až 14 v (v propojeném scénaří vidlicí bez joysticku vždy 13,7 až 14,5 V), ale elektronika jednotky klasickými 3,3 V. Často docházelo k nevratnému poškození zapříčiněné nechtěným kontaktem mezi napájením (kolejnicí) a elektronikou (obvody) – efektivním zničením obvodů desky. Např. ze zmiňovaných 4 skupin jsme 3 procesory zničili (v mém případě nešikovným měřením s multimetrem).
- Složitost výměny – zničený obvod nebo čip se pracně vyměňuje. Nutné jisté podmínky včetně odbornosti. Prostor pro nejistoty.
- Složitost realizace projektu – nejen z mého pohledu doba silně pokročila a nároky na studenty se mění. Např. dobře si pamatuji v mém prvním zaměstaní jako embedded programátor jsme programovali přes registry. Dnes má většina výrobců svůj framework.

1.5 Cíl projektu

Z mého pohledu je cílem připravit studenta na klíčové součásti budoucí kariéry jakožto programátora. Dovolil bych si takového programátora označit jako embedded programátora, tedy programátora vestavěných systému (typická klíčová slova: bare-metal, C/**C++**, čtení schémat, alespoň částečná znalost elektra obecně,…). Tím nechci dotyčného studenta nálepkovat, ale ujasnit, že takový student má zájem spíše o elektroniku než vyšší technologie a taktéž je spíše nakloněn získání znalostí fungování věcí a jejich použití v praxi namísto použití předpřipravených technologií.
Sám jsem se embedded programováním v minulosti živil, a i dnes čas od času nějaký embedded projekt řeším. Např. se mi naskytla příležitost naprogramovat automat na šipky. Domnívám se, že mě to dostatečně kvalifikuje a mám tak solidní základ vyhodnotit přínosy takového projektu. Jaké klíčové součásti identifikuji:

- Izolovaný úkol s rozšířením na obecnost – student pochopí rozdíl mezi obecností a pragmatismem – tedy mezi účelovým a obecným kódem. Představa přestavěné dráhy ho nutí přemýšlet obecně.
- Základy HW – student musí načíst datasheety a zjistit, jak jednotlivé komponenty fungují. Jaká data produkují a případně jaká přijímají. Zde uplatní také základy fyziky a elektra obecně. Velkým tématem je znalost konk. MCU-/procesoru.
- Základy standardních periferií – student v praxi používá znalosti v nastavení a použití modulů periferií – PWM, sériová komunikace, digitální signalizace…
- Základy regulace a řízení – student si osvojuje systém vyhodnocování vstupů a reakcí na ně.

- Základy architektury – student navrhuje svá první řešení – bude jádrem FSM nebo jiná struktura…?
- Interakce s IDE – student se setkává s vývojářským prostředím, které má své funkce a určitou míru intuitivnosti. Též se setkává s různými rozšířeními a balíčky třetích stran.
- Základy metodiky v embedded – student se snaží o debugging a automatizaci některých procedur. Snaží se zároveň ušetřit systémové prostředky. Reaguje na způsob nahrávání a kompilace kódu. Osvojuje si princip hlavní smyčky a taktu krystalu.
- Externí periferie, 3rd party code – student získává zkušenosti s komunikací s externími periferiemi jako je např. akcelerometr nebo R/W na SD kartu. Je nucen použít rozhraní třetích stran nebo si vlastní vyrobit. Popř. jít střední cestou a vytvořit si wrappery.
- Čtení ze schémat a dokumentací – student fyzicky nachází spojitosti a cesty mezi procesorem a periferiemi a periferiemi samotnými. Učí se značky a označení.
- Mechatronika – zde použit trochu obecnější pojem, ale v zásadě spojuje dovednosti, které si student může rozšířit během realizace – pájení, technická dokumentace, vlastní rozšíření, mechanické úpravy,…Vyžaduje (nejen) mentální ohebnost a zručnost. Prostor pro nový HW (LEDky, nové desky, mezikusy,…).

1.6 Motivace

Důvodem motivace, resp. zvolení si tohoto tématu diplomové práce, je cíleně hmatatelný a použitelný výsledek. Můj cíl je obnovit, nebo alespoň co nejvíce se přiblížit obnově (případně připravit stabilní půdu pro obnovu) ročníkováho projektu popisovaného v předešlých kapitolách.
Předešlé řešení má nyní několik nevýhod, které bych rád eliminoval a projekt renovoval/připravil do moderní a modulární podoby. Chci, aby se student minimálně věnoval fázi tzv. “rozchození” a následné heureky ala “svítí mi dioda”. Chci, aby student projevil svůj potenciál v návrhu řídicího regulátoru v podobě algoritmu a nemusel řešit okolní související věci. Samozřejmě za předpokladu, že sám nechce, protože mj. motivací je i rozšiřitelnost, dle fantazie studenta. Ergo moji motivací je i vyvinout nástroje a postupy, které studentovi jeho počínání se splněním projektu pomohou.
Já se naopak nechci seberealizovat ve vytvoření dokonalého algoritmu řízení, ale usnadnit přesně toto studentovi.

1.7 Plán renovace

1.7.1 Základy předešlé práce

Má diplomová práce se opírá o výsledek magisterského projektu z roku 2022 („Elektronická řídicí jednotka samořízeného modelu vozu: Electronic control unit for self-driving car model“ [2]).
V podstatě se jedná o stejnou motivaci jako nynější práce. Zaměření bylo na samotný hardware – tedy ve zkratce: Čím nepodporovanou desku uvnitř vozidla nahradit (lze číst jako “Příprava pro renovaci”). Výsledkům předcházela rešerše v podobě studia současného hardwaru a to v několika bodech: Zjistit funkčnost a součásti současného HW (vozidlo, ECU, podmínky,...) Zajistit náhradní HW (novou desku jako ECU) s ohledem na:

- Cenu
- Dostupnost
- Vyměnitelnost
- Funkčnost
- Rozměry
- Funkcionalitu (nejvíce se přiblížit požadavkům)

Jakmile je nová deska pro ECU vybrána, tak sepsat základní kód, který by rozchodil periférie.

1.7.2 Magisterský projekt

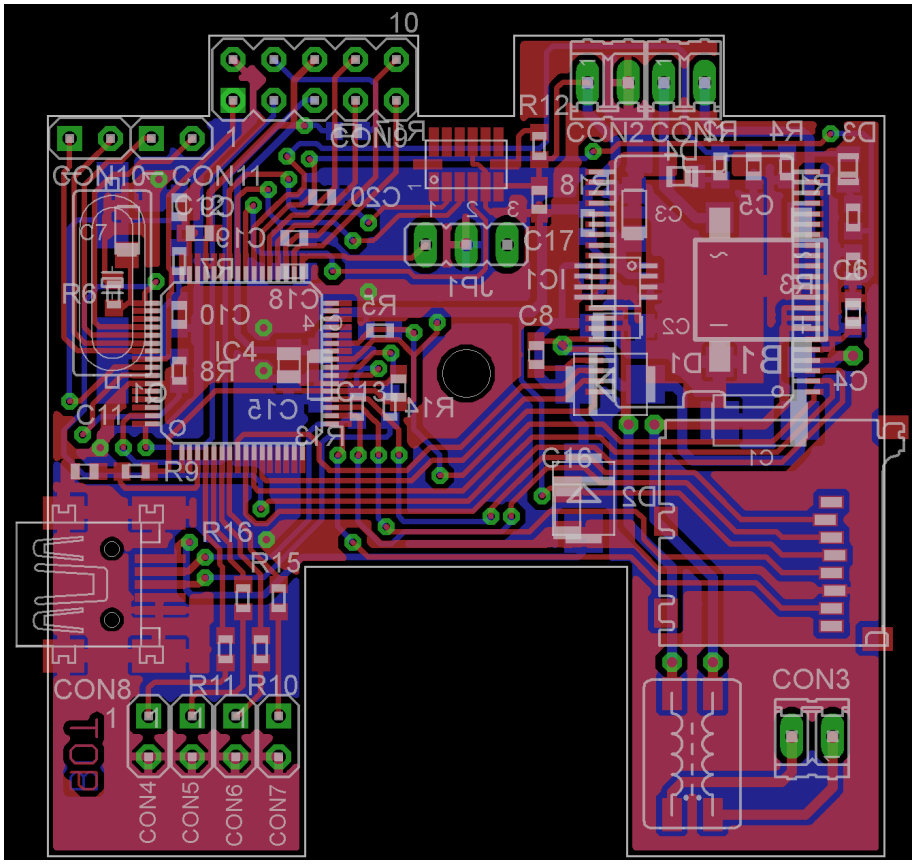
V průběhu magisterského projektu mi byl dodán HW – a to zejména model vozidla ve dvou variantách (přesněji Audi R8 LMS a BMW Z4 M Coupé), který byl vybavenou starou deskou, ke které se v zápětí dostaneme.
Modely jsou v měřítku 1:30 a váží okolo 90 g. Mají 4 pogumovaná kola ve dvou nápravách a to klasicky v podobě jedné vpředu a druhé v zadní části vozidla (jako u skutečného vozidla/automobilu). Obě mají shodně volnou přední nápravu a zadní je poháněná elektromotorem (kartáčový stejnosměrný). Před přední nápravou se nachází napájení z kontaktů s dráhou. Obě jsou vybaveny stejnou dříve zmíněnou deskou, lze je tedy považovat za totožné z hlediska funkcionality.
S odkazem diplomovou práci Ing. Honse, který se zabýval podobnou problematikou [3], ale pro jiný mikrokontrolér, bylo potřeba určit, jaké vlastnosti desky (ECU) z pohledu projektu hrají roli. To tedy i znamená jaké vlastnosti zachovat (hledat podobné řešení), od kterých upustit nebo najít alternativní řešení. Analizoval jsem vlastnosti desky a MCU v mém případě a vytvořil rozdělení jednotlivých vlastností do kategorií priorit. Schéma desky je vyzobrazeno na ilustraci 1.5.

V dřívější kapitole (viz. 1.7.1) jsem vypsal seznam vlastností, které musí být zohledněny, udělám takový shrnující průnik kategorie nejvyšší priority a seznamu zohlednění.
Najít podobné řešení bylo téměř nemožný úkol neboť deska byla vyvinuta primárně pro účely vývoje autonomního řízení modelu vozu na autodráze. Přesto jsem se pokusil během výzkumu najít desku s alespoň podobnou výbavou. Prověřil jsem jak samotného výrobce modelů, tak jiné velké i malé hráče na trhu s MCU. Bezúspěšně. Nic se ani vzdáleně nepodobalo výbavou staré ECU. Alternativou bylo najít základní platformu (desku s procesorem) a dané/chtěné periférie popř. dokoupit a zapojit. A tou cestou jsem se posléze vydal.
Nejpalčivějším úkolem tak zůstalo najít takovou novou desku, aby její rozměry ne-

(a) Model s karosérií (b) Podvozek modelu

Obrázek 1.4: Model vozidla – BMW

přesahovaly vnitřek vozidla a která by byla dostupná v horizontu alespoň několika let a to za přijatelnou cenu. Byl mi dodán seznam kandidátů desek od studentů, kteří v rámci podobného tématu jiného oboru na MTI, řešili podobnou problematiku. Ve zkratce žádné řešení nevyhovovalo, každé z návrhu mělo fatalní nedostatky, co se týče požadavků (z hlediska nerealizovatelnosti). Nejčastěji se vyskytoval problém s nerespektováním rozměrů, což je jeden ze základních aspektů splnitelnosti renovace projektu. Pustil jsem se do vlastního hledání.
Metodika hledání, byla především hrubá filtrace na základě požadavků. Pomyslným sítem ve většině případů neprošly doslova velké kusy. Většina desek i několikanásobně převyšovala jeden nebo více rozměrů. Druhým sítem byla filtrace těch kusů, které neumí přijmout napětí přímo z kolejnice bez mezikusu (napájení 14 V). Rozhodl jsem se pro tento krok, aby výměna byla jednodušší. Znamenalo by to další součástku a další práci při výměně.
Nakonec se mi podařilo takové řešení najít [4] a tou byla deska od společnosti Arduino (ilustrace 2.1). Celý postup výběru je popsán ve zmíněné magisterské práci, já se zde chci primárně opřít o faktický výsledek. A ten ve zkratce je, že byla vybrána



Obrázek 1.5: Stará deska ECU

deska Arduino Nano 33 BLE. Řešení na kterém lze stavět a tato práce tak činí.

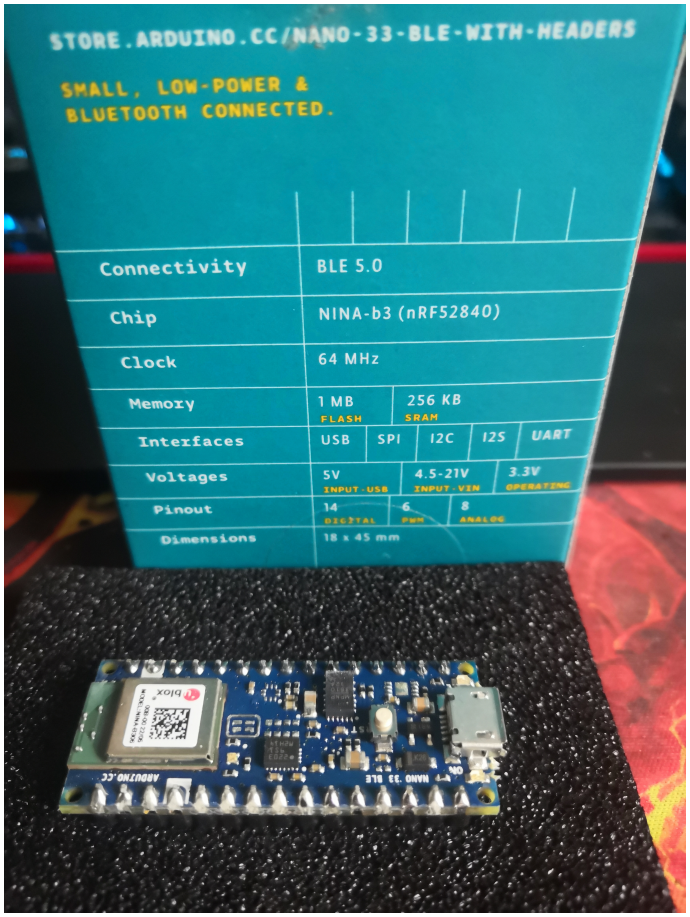
1.7.3 Nový projekt

Jako se magisterský projekt opírá o hledání ideálního HW řešení, se tato práce opírá o nalezené HW řešení a hledání cílů usnadnění pro studenta volící ročníkový projekt. Renovace předpokládá, že se použije stejné vozidlo z předešlých let, v mém případě bude test-subject model vozu BMW Z4 či Audi R8, dráha zůstává stejná a ECU se bude ze základu sestávat ze zvolené desky Arduino Nano 33 BLE (viz ilustrace 2.1).
Dalším předpokladem je usnadnění a renovace principů přístupu a řešení (ze zhotovilského hlediska) z předešlých let. Tj. Z mého pohledu vytvořit usnadňující procesy a nástroje. Student by se měl primárně zaměřit pouze (avšak ne nutně) na algoritmus řízení a využít proto poskytnuté nástroje.
Mým cílem je vytvořit mobilní aplikaci na (nejen) debugging a příjem dat z vozidla, ale taktéž i smysluplnou základní architekturu dlouhodobě udržitelného projektu, do které student může volně sahat a rozšiřovat ji. Zvolil jsem, že se budu při návrhu opírat o nějaký bezpečnostní standard pro automotive nebo podobně.
Jednotlivé věci budou popsány v následujících kapitolách o realizaci.

2 Realizace

V zásadě jsem chtěl kopírovat zadání v jeho pěti bodech a ty postupně řešit. Co pro mě jako vyhotovitele zadání oněch 5 bodů znamená bude rozepsáno v následujících kapitolách.

2.1 Analýza nového HW



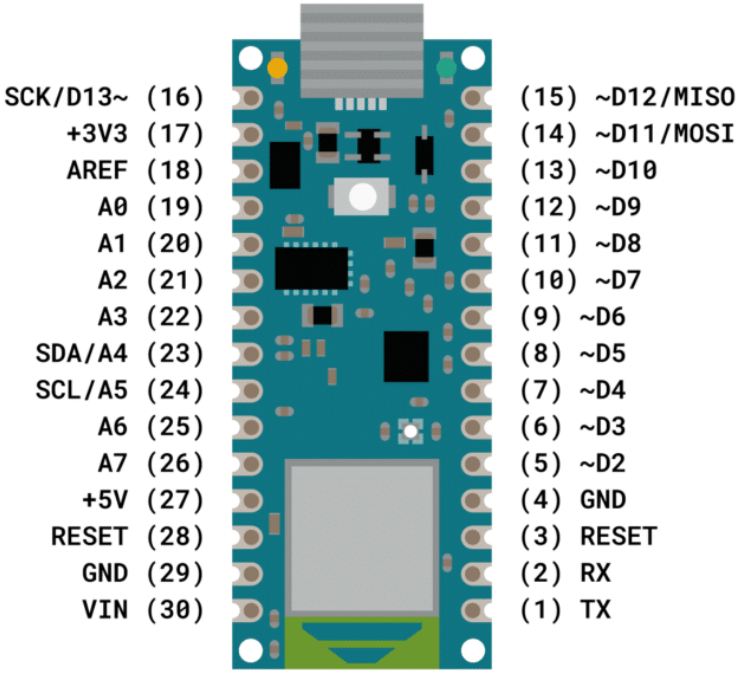
Obrázek 2.1: Jádro ECU – Arduino Nano 33 BLE

V krátké rešerši jsem ověřil, že současná deska Arduino Nano 33 BLE je stále ucházejícím řešením. Nutno podotknou, že s několika výhradami a to z těchto

důvodů:

- Největším/nejcitelnějším kritériem omezení je rozměr, který lze vměstnat do útrob modelu vozu – 33 BLE stále ideální řešení.
- Dostupnost – hraje též velkou roli, velcí hráči na poli embedded řešení nemají desky dostupné nebo nevyhovují.
- Cena – stále v rozumných mezích (do 1000 za kus).
- Efektivita/ideálnost řešení – deska v mnoha ohledech předčí bývalý dedikovaný HW, jsou zde zjištěné nedostatky, více v následující kapitole (2.2).
- Napájení desky – v drtivé většině nepřekročí konkurenční řešení 5 V – 33 BLE je skoro až "industriální řešení" (ty jsou často mnohonásobně dražší).

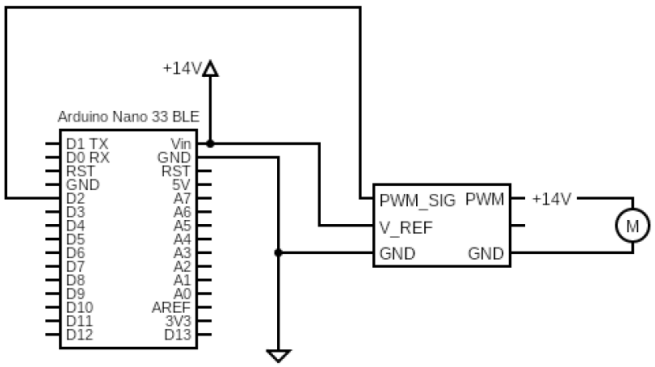
2.2 Nový HW



Obrázek 2.2: Pinout desky

Vybraná nová ECU má v zásadě 2 (potažmo 3) nedostatky.
Prvním je konfigurace PWM na desce. Autor magisterského projektu byl po výběru a rozchození desky upozorněn, že daná PWM situovaná v obvodech desky je tzv. signalizační (ergo nízkovýkonová). Dle dokumentace [5] nelze nesouhlasit, umožňuje odběr pouze do 15 mA (viz ilustrace pinoutu 2.2 – D2, 5). Společným měřením bylo objeveno, že současný HW (vozidlo s elektromotorem) má odběr okolo 150 mA, spíše předpokládám více. To v souladu s elektro zákony by způsobilo "usmažení" obvodů desky.
Existuje však řešení. Tím je PWM řízený H-můstek. Ten je napájený přímo od zdroje napětí (tj. V našem případě dráhy) a jeho konfigurace řízená klasickou signalizační PWM (jakou máme v desce). Trocha pájení a problém je skoro vyřešen (schéma na ilustraci 2.3). Slovíčko skoro je zde použito, protože většina H-můstkových řešení nezvládne napětí nad 10 V. Při sepisování této práce není úplně jisté jestli se napětí na dráze omezí na 10 v nebo bude doručen odolnější tištěný H-můstek, který dané napětí zvládne. Opět se jedná o industrial-řešení a je tak v jiné cenové kategorii. Detaily naleznete v samostatné kapitole 2.2.1.

Druhým bodem v kategorii nedostatků je chybějící snímač okolního světa. V zásadě řečeno, nalezená deska nemá žádný "nativní" senzor, který by umožňoval autonomnost. Není zde žádné optočidlo, žádný akcelerometr. Řešení je de facto holé na základy. Zde to ale nehodnotím jako negativní věc, protože holé řešení donutí studenta přemýšlet a vybrat si další součástky podle svého.



Obrázek 2.3: Schéma zapojení H-můstku

Třetím (a účelně označené "potenciálním") problémem je komunikativnost BLE. BLE znamená Bluetooth low energy. Je nutné mít na paměti, že BLE není v principu kompatibilní s klasickým BT (ani nejvyšší verzí) [6]. Zařízení s BLE není kompatibilní s BT a naopak. Pouze v případě, že je zařízení označeno jako Dual mode nebo podobně, pak tam má jistou zpětnou kompatibilitu. Mobilní zařízení nevyjímaje. Nové smartphony často Dual mode mají.

2.2.1 Řízení motoru a výběr H-můstku

Bylo již předestřeno, že výběr H-můstku nebude triviální záležitostí a že některé parametry výběr učiní obtížnějším úkolem.
Naznačen byl parametr v podobě vstupního napětí. V takovém případě se lze rozhodovat de facto dvojím způsobem.

1. Lpět na vstupním napětí dráhy a vybrat řešení, které požadavek splňuje
2. Upravit parametry zadání a snížit napětí na dráze

První způsob uvažování nás na první pohled velmi omezuje. Na druhou stranu získáme řešení na míru bez úpravy podmínek. U druhého způsobu je tomu přesně naopak, snažíme se porušit podmínky zadání, ale přináší to širší manévrovací prostor.
Já se rozhodl pro možnost první. V praxi sice dochází k určitým kompromisům, co se vývoje obecně týká (druhá možnost je taková více na první pohled přijatelná), ale v první řadě je dobré si stanovit omezující podmínky a vybičovat se k hledání optimálního řešení. Ne vždy je možné podmínky ohébat a je na to potřeba neustále myslet.

Mluvím z vlastní zkušenosti z praxe jako OSVČ, když tvrdím, že je to tzv. "50 na 50". Někdy je možné navrhnouzt alternativní řešení – příkl. bych uvedl poptávku vývoje data crawleru a nakonec klient souhlasil s desktop aplikací, kterou by zásoboval uloženými stránkami (HTML kódem). ve výsledku jednodušším

na realizaci a i levnější pro klienta (nízké využití aplikace, jinak by byl výhodnější crawler samozřejmě).
Nyní vyjádření k druhému způsobu a zde je příklad z praxe. Klient poptával aplikaci pro Android, ale zásádní podmínka byla, že musí aplikace bezpodmínečně fungovat na všech mobilních telefonech s OS Android. Zde jsem si nemohl vybrat, že build pro verzi 12.0 prostě nefunguje, tak se upraví podmínky, že danou aplikaci může stáhnout jen ten či onen dotyčný s tou či onou verzí. Pokud by bylo známo, že Android 12.0 je exotickou verzí se známými chybami a i samotný vývojář doporučuje přejít na verzi nižší či vyšší, třeba by klient uvažoval jinak.

Tím, že jsme se rozhodli jít cestou striktních podmínek, a tím netvrdím, že je druhý způsob není správně, většina řešení tak byla ve výběru eliminována. Postupně vznikl seznam cca 20 potenciálních kandidátů.
Nicméně většina z nich selhala na rozměrech nebo dostupnosti. Taktéž se často vyskytovala jen duální verze, řešení, které umí řídit 2 motory současně. To pro nás znamená místo navíc, které si nemůžeme dovolit. Zde je seznam nejzajímavějších kandidátů:

- L298N dual [7] – vstupní napětí: 7 V – 35 V, výstupní proud: až 2 A na kanál, přibližná cena 150 Kč – nevyhovuje velikostí
- DRV8833 [8] – vstupní napětí: 2 V – 11 V, výstupní proud: Až 1,5 A na kanál, přibližná cena 100 Kč – nesplňuje požadavek na napětí, kdyby se upravili podmínky na 10,8 V, bylo by to ideální řešení
- L293D [9] – vstupní napětí: 4,5 V – 36 V, výstupní proud: Až 600 mA na kanál, 1.2 A, přibližná cena 70 Kč – za předpokladu, že si elektromoer modelu nepřekročí 600 mA je toto ideální řešení
- BTS7960B [10] – vstupní napětí: 5,5 V – 27 V, výstupní proud: Až 43 A na kanál, přibližná cena 200 Kč – velikost je problém
- VNH2SP30 [11] – vstupní napětí: 5,5 V – 16 V, výstupní proud: Až 30 A na kanál, přibližná cena 300 Kč – velikost je nejspíše problém
- SN754410NE [12] – vstupní napětí: 4,5 V – 36 V, výstupní proud: Až 1 A na kanál, přibližná cena 40 Kč – pouze čip bez řešení ¨(složitá výměna)
- MC33886 [13] – vstupní napětí: 5 V – 28 V, výstupní proud: Až 5 A na kaná, přibližná cena 250 Kč – pouze čip bez řešení ¨(složitá výměna)
- IBT-2 [14] – vstupní napětí: 6 V – 27 V, výstupní proud: Až 43 A na kanál, přibližná cena 150 Kč – velikost je problém

2.3 Základy SW

2.3.1 Vývojové prostředí

Pro vývoj ukázkového základního kódu jsem si vybral Arduino IDE (Legacy). V podstatě je to přímo dělaný nástroj pro programátory, kteří si zvolili Arduino jako svou cílovou desku.
Samotné prostředí si lze nastavit, např. lze nastavit barevné téma. Je zde několik

Obrázek 2.4: Logo Arduina z roku 2023

nástrojů pro debugging, uvedl bych monitor a plotter pro sériovou komunikaci.
Arduino IDE je zcela zdarma a je stáhnutelné přímo na stránkách Arduina. Výhodou je dostupná verze pro nižší Windows než jsou Windows 10. Podporuje též Linux a Mac.

Proto, abyste mohli IDE použít, je potřeba pár základních nastavení.

1. V první řadě je potřeba otevřít IDE, v něm by měl být základní soubor už předvytvořen. o souborech a jazyku se budu zmiňovat v následující kapitole 2.3.2 a 2.3.1, avšak pokud by se základní soubor nevytvořil, stačí jej vytvořit manuálně přes CTRL + N nebo přes nabídku v horní liště: File -> New.

2. Připojte desku přes USB. V našem případě Arduino Nano 33 BLE. Měla by se objevit v Tools -> “Board: <Název desky>”

3. Je možné, že Vám desku sám nedetekuje, protože mu bude chybět knihovna. Jděte do Tools -> Board(s) -> Board Manager. Najděte knihovnu “Arduino mbed os nano boards”, stáhněte a nainstalujte (automatický proces).

4. Pokud desku neobjevil, odpojte ji a znovu připojte. Bude pravděpodobně potřeba ji vybrat manuálně a to přes Tools -> Board(s). Zde najděte v nabídce.

5. Po detekci nebo výběru desky není potřeba nastavovat procesor (v našem případě). Ovšem, co je potřeba, je nastavit programátor na Stlink. Toto nastavení naleznete v Tools -> Programmer.

6. Tento krok je zcela dobrovolný, ale máte možnost ověřit, že deska je správně připojená a správně komunikuje. V Tools -> Get board info. Pokud nedostanete základní info od desky, opakujte celý proces, příp. připojte jinou desku.

Pro nahrání kódu do desky je potřeba kód zkompilovat a spustit nahrávání (upload). k tomu slouží zkratky CTRL+R a poté CTRL+U. Progres je vidět v liště s výstupem. Zde jsou popsány i případné chyby.
IDE se může lišit konfigurací zkratek, tak raději popíšu i alternativní postup.
V horní liště najdete Sketch a v něm jako první v nabídce Verify/Compile a pod ním Upload.

2.3.2 Programovací jazyk

Programovacím jazykem je zde jazyk, který je derivovaný od klasického **C++**. V podstatě ho lze brát jako **C++** s knihovnou a frameworkem Arduina. Podporuje všechny klasické vlastnosti jako běžné **C++**:

- datové typy
- knihovny
- prototypy
- šablony
- OOP

Chybí zde "main" v klasickém pojetí. Zde je to trochu jiné, ale stále pochopitelné i pro začátečníka. Základem zde jsou tzv. sketche. Žádný solution, žádný projekt. Jediný soubor s příponou ".ino". V něm se nachází 2 předpřipravené funkce setup() a loop().
Jedna slouží k volání Vašich init funkcí a druhá k běhu samotné smyčky.
Kdybych to měl přepsat do klasického main.c, vypadal by kód asi nějak takto:

```
1  void setup() {
2      // setup inits
3  }
4
5  void loop() {
6      // do~something each loop
7  }
8
9  int main() {
10
11     setup();
12
13     while(1) {
```

```
14          loop();
15      }
16
17  }
```

Výhodou Arduino IDE je také ta, že máte k dispozici ukázkové sketche přímo v IDE. V horní liště zvolte Sketch -> Include library a vyberte ukázkový kód pro dané téma.

2.3.3 Bezpečnostní norma

Chtěl jsem, aby kód, který student použije nebyl jen kostrou. Chtěl jsem, aby kód splňoval obsahově nějakou z bezpečnostních norem např. pro vývoj v automotive. Původně jsme s vedoucím vyhledli normu ISO 26262 – ve zkratce norma pro funkční bezpečnost pro automotive (tedy i programování ECU). Ale ukázalo se, že sehnat takovou (byť v elektronické podobě) je jednak náročné a druhak velmi drahé. Nakonec jsme našli normu ČSN EN 61508-3, která je obsahově velmi podobná.

Norma pro funkční bezpečnost

V celém znění: Funkční bezpečnost elektrických/elektronických/programovatelných elektronických systému související s bezpečností – Část 3: Požadavky na software. Já o ní zde budu mluvit jako o normě nebo normě bezpečnosti.
Norma byla vydána v roce 2011 a je platná dodnes. Z této normy se (krom jiných) derivovala i norma ISO 26262 a to specificky pro automotive. Mezi další patří adaptace pro železnici, zpracovatelský průmysl, elektrárny a strojírenství. Všechny víceméně kopírují obsah a jen upravují pro konk. užití v oblasti.
Norma je poměrně obsáhlá a obecná, ale její požadavky pro tuto úlohu bych tu zkrátil do pěti bodů a vysvětlil jak se dodržují zde.

- Safety requirement – krom dbání na bezpečnost obecně je potřeba např. vozidlo zastavit a to třeba i vzdáleně. Pokud je ECU připojená, lze ji přes mobilní aplikaci zastavit. o tom více v kapitole o nástrojích (viz. 2.4).
- Design – ECU by měla umět pracovat s chybami a rozlišovat je na fatální (zastavit vozidlo) a nefatální (signalizovat, ale nechat běžet algoritmus dál – s přeskočením subrutiny).
- Documentation – jak již název napovídá, mít kód zdokumentovaný do úrovně vysvětlení principů fungování (včetně bloků a funkcí samotných). Splňuji zde tím, že komentuji kód a samozřejmě velká část této práce se tomu věnuje v kapitole 2.3.4.
- Verify and validate – v podstatě rozdělení kódu do bloků, které se dají testovat. To znamená, že je důležité mít kód stavový (testování jednotlivých stavů), mít try-catch blok (ve zkratce zachytávat chyby) a všechny bloky otestovat.

- Tracing – je bodem, který říká, že jakákoliv chyba musí být replikovatelná a dohledatelná. Jednou možností je mít tzv. blackbox (zapisovat do nevolatilní paměti událost), ale debugging po sériové lince by měl vyhovovat. Bez něj by např. nešlo vytrasovat, jestli blackbox nemá poruchu.

2.3.4 Základy kódu

Do začátku bych rád uvedl, že jsem kód psal procedurálně – tedy neobjektově. Není to tak, že by to objektově taky nešlo (tím naopak povzbuzuju čtenáře, aby si základní kód přepsal do objektů, zdali mu to tak více vyhovuje), ale knihovna od Arduina je též psaná procedurálně s využitím C-like kódu s výjimkou užití šablon. Chtěl jsem tedy zachovat konzistenci a tradiční C-like appeal, který je v embedded standardem.
Dále je možná potřeba uvést, že veškerý zde zmíněný ukázkový kód je smyšlený. Vymyslel jsem si nějaký realistický scénář a ten pomocí syntaxe kódu realizoval (občas jsem volil až příliš konkrétní názvy, ale v dobrém duchu a v rámci jasnosti a srozumitelnosti).
Ukázkovým kódem je myšlen takový kód, který demonstruje příklad. Neplést si s reálným kódem použitým v aplikaci a základní struktuře.

Každý sketch znamená de facto samotný projekt. Lze tedy psát kód do jednotlivého souboru, např. do "slotcar.ino". Avšak to nedoporučuju jednak z hlediska přehlednosti, tak i udržitelnost kódu. Sám jsem zvolil jinou metodu (a pokud student zvolí jinou, má tu možnost, ale plně nevyužije potenciál mnou připravené architektury) granulovaní kódu do jednotlivých modulů. Arduino IDE totiž obsahuje kompilátor pro **C++** soubory a umí s nimi spolupracovat.
Jednotlivé moduly jsou tvořeny hlavičkovým a zdrojovým souborem jako standardní **C++**. Do ".h" si vložte "#pragma once", do ".cpp" pak kód pro linker. Nepoužívejte ".hpp" ani ".c". Všechny soubory mějte ve stejné složce jako máte sketch. Alespoň v mém případě IDE neumí pracovat s různými úrovněmi vnoření, což mě na jednu stranu mrzí z hlediska struktury a přehlednosti, ale nepřišel jsem na to, jak to obejít.

Syntaxe, metodika a programátorská hygiena

Zde bych rád popsal metodiku užití syntaxe a jaká pravidla jsem vytvořil a dodržuji. Každému vyhovuje vlastní syntaxe a styl, ale rád bych tu uvedl svůj, aby byl kód přehlednější a hlavně čitelnější.

Nejdříve si definujme/sjednoťme termíny pro formátování textu:

- Camel case – též znám jako UpperCamelCase nebo PascalCase má tvar složených slov a každé začíná velkým písmenem. Typický např. pro **C#**.
- Lower camel case – tedy lowerCamelCase je jako UpperCamelCase, ale jeho první písmeno je vždy malé. Např. U Javy nalezneme.

- Snake case – neboli lower snake_case má formátování jednotlivých slov bez užití kapitálky a jsou oddělená podtržítkem. Typický pro původní dialekt jazyka C, kde je prakticky vše ve snake case.
- Upper snake case – jedná se o snake case kde jsou namísto malých písmen použita pouze velká. Často vidíme u konstant.
- Kebab case – taktéž známý jako dash-case je typický svým formátem oddělující slova pomlčkou/spojovníkem – přesněji znaménkem mínus. Právě význam znaku mínus jako operátoru znemožňuje v mnoha jazycích použití jinak než v řetězci.
- Upper case – vytvoření názvu jen pomocí velkých písmen, bez mezer.
- Lower case – vytvoření názvu je pomocí malých písmen, bez mezer.
- Můj modul case – nazvěme ho module case pro pořádek, protože se de facto nepoužívá jinde než právě u modulů - přesněji pro názvy funkcí. Jedná se o kombinaci vícera stylů. V podstatě se jedná o spojení upper casu a camel casu přes snake case. Zde je uvedena definice: "UPPERCASE_CamelCase".

Nyní se můžeme podívat na jednotlivé případy a ukázky. Nutno podotknout, že z hlediska obecnosti používám výhradně angličtinu – jak pro názvy, tak i popisky a komentáře.

Komentáře a popisky

Komentáře jsou v mém kódu několika typů a zde je popíšu:

- Strukturový komentář – nebo též sekční komentář mi pomáhá oddělovat jednotlivé části kódu nebo definuje prostor pro určitý typ kódu. Používám v zásadě pouze 2 úrovně. Jedna je striktně upper case a druhá je camel case. Vždy jsou v celkové délce řádku 68 znaků. Číslo je odvozeno od 80 znaků, které jsou historickým maximálním počtem znaků na řádek – přidáním odrážky v podobě 12 znaků (při použití monospace editorů).
- Hlavičková komentář – u funkcí (resp. nad ní). Popisuje funkci a popř. I její parametry a návratové hodnoty s podmínkami.
- Blokový komentář – zejména ve funkcích, je vždy s počátečním malým písmenem a komentuje, co se v daném bloku děje.
- Řádkový komentář – informativní dodatek k proměnné nebo řádku obecně.
- Podmínkový komentář – popsán v samostatné sekci s ukázkou (viz 2.3.4).

```
/* FUNCTIONS
   ====================================================*/
/* MoodChangers
   ----------------------------------------------------*/
// This function makes user of given id happy
// id - id of user
// returns positive as strength of happiness
// - can return negative if failure
int MakeHappy(int id) {

    int res; // stores result

    // tries to change the mood to happiness
    res = ChangeMood(id, HAPPY);
}
```

Proměnné

Proměnné vytvářím za pomocí snake case. Snažím se se vyhýbat jednopísmenným názvům, pokud není zcela jasné o jakou proměnnou se jedná. Většinou píšu celé pochopitelné názvy, výjimkou jsou názvy proměnných u funkcí jako parametry, kde se často zkracují. Znaménko pro pointer vkládám v definici k typu proměnné, při použití samozřejmě k názvu proměnné. Název, pokud se jedná o jen o funkci pointeru – tedy ukazatele, nepracuje se s hodnotou na adrese kam ukazuje, používám suffix „_ptr“. Při definici polí používám konstanty. Konstanty jsou pouze upper snake case.

```
#define STR_END '\0'
#define BUF_LEN 512

const float PI = 3.149;

char c1 = 0x0d;
char c2 = 'W';
char c3 =  STR_END;
uint32_t my_integer;
char* c2_ptr;
char buf[BUF_LEN];
char* storage = buf;
```

Objekty a typy

Názvy pro objekty, struktury, enumerace tvořím camel casem. Respektuji základní typy s lowercasem. Některé typy jsou definovány v souboru “commons.h” – o tom je samostatná kapitola 2.3.5. U enumerací buď číselně definuji všechny stavy nebo alespoň počáteční. Pokud se jedná o konečný výčet a je zde vždy iterace o 1 od 0, přidávám koncový stav pro získání počtu hodnot. Vždy beru enum jako unsigned.

```
typedef enum {
    // first
    MYENM_VALUE = 0,

    // count of myenum members
    MYENM_Count,

} MyEnum;

typedef uint8_t u8; // unsigned 8 bit type
```

Funkce a prototypy

Názvy funkcí tvořím přes camel case a to v případě, že jsou privátní. Nepoužívám klíčová slova jako "public" a "private", privátní myslím takovou funkci nebo proměnnou, která není exportována mimo modul. Pro veřejnou proměnnou používám klíčové slovo extern. Pro veřejnou funkcí z modulu používám výše definovaný module case.

Hlavičkový soubor ukázky by mohl vypadat takto:

```
// in module.h file
void MODULE_Init();
```

Zdrojový soubor ukázky by mohl vypadat takto:

```
// in module.cpp file
extern bool ready = false;

// Inits entire module
void MODULE_Init() {

   // init some stuff\dots
   // some stuff


   // create log
   InitLogCreate();
}

// Creates init log
void InitLogCreate() {

   // some stuff
   ready = true;
}
```

Smyčky

Pro while nebo do while používám "1" namísto "true", obojí je syntakticky v mém projektu možné. For smyčka má klasický syntaktický předpis s užitím proměnné i uvnitř a iterací o jednu.

```
// check all registered structs
for(int i = 0; i != not_met_cond; i++) { // always using
    brackets

     // get object
    MyStruct* ms = structs[i];
}

while(1); // infinite loop

// code block with security
bool condition = false;
int security = 10;
do {

     // make something with condition
    condition = CallFunction();

     // security check
     security--;

} while(!condition || security);
```

Podmínky

Pokud lze použít switch, použiju switch s default casem (zejména u enumerací). Snažím se rovnou vracet návratovou hodnotu pokud to logika umožňuje. Pokud nelze použít switch, snažím se dělat pokaždé if-else statement. Při odchytávání chyb volím metodu "jestliže nastala chyba, odejdi".
Komentáře u switch-case statementu jsou v podstatě dvojího typu. Zastřešující komentáře pomyslné sekční úrovně 3, který je použit pokud se jedná o skupinu případů se stejným nosným znakem – takové případy lze pozorovat u enumerací. Vždy (opět z hlediska zobrazení u monospace editorů) 30 znaků. Druhým je obecné komentování jednotlivých stavů ve stylu komentáře hlavičky funkcí.

```
switch(myenum_val) {

     /* KNOWN STATE ------------*/
     // Good one
     case MYENUM_SUCCESS:
     return true;
```

```
7
8       // Not so good
9       case MYENUM_FAIL:
10      return false;
11
12      /* THIS IS BAD ------------*/
13      // Unknown state, programmer´s fault
14      default:
15      break;
16  }
17
18  error = "Encountered unknown "state;
19  return false;
20
21
22  // Checks if sensor is good shape
23  MyEnumState IsOk(int state) {
24
25      // is not positive number, skip
26      if(state < 1)
27          return STATE_UNKNOWN;
28
29      // check for diff states states
30      if(state < 0x1000) { // only error states
31          return STATE_ERROR;
32      } else if(state < 0x2000) { // only warnings
33          return STATE_WARN;
34      } else { // all ok
35          return STATE_INFO;
36      }
37  }
```

2.3.5 Moduly

Tady bych nejdříve krátce zmínil strukturalizaci jednotlivých souborů modulů. V hlavičkovém souboru dělám include-guard přes “pragmu” namísto klasicky C-like přes “define”, je to však zase na studentovi.
Poté pokaždé include 2 základních souborů (kapitola 2.3.5 a 2.3.5).
Následující definice a makra, které bych chtěl exportovat z modulu (v kompilační části ne v běhu programu). Po nich přichází definice užitých typů, které chci exportovat z modulu.
Soubor je poté zakončen definicemi prototypů.

Hlavičkový soubor může vypadat takto:

```
#pragma once
// Or use this
#ifndef HEADERFILENAME_H
#define HEADERFILENAME_H

#include "commons.h"
#include "config.h"

/* DEFINITIONS
   ================================================*/
#define MODULE_MEM_USAGE 512

/* TYPE DEFINITIONS
   ============================================*/
typedef signed single doublechar;

typedef enum {

    UNKNOWN = 0,
    ON,
    OFF,
    FAILURE
} MachineState;

/* PROTOTYPES
   ================================================*/
MachineState CheckMachine(void);

#endif // !HEADERFILENAME_H
```

Zdrojový soubor vždy začíná self-includem – tj. inkluzí vlastního hlavičkového souboru.
Následují další includy přičemž je řadím do 3 kategorií a sekvenčně řadím a odděluji:

- Systémové – ty, které jsou standardní pro **C**nebo **C++** knihovny a jsou v závorkách "<>"
- Modulové – mé vlastní hlavičkové soubory, jsou v uvozovkách
- Frameworkové – souvisejí s užitým frameworkem což je momentálně

Pod includy definují konstanty a makra (věci pro kompilaci či podmíněnou kompilaci).
Následují externy a globální proměnné včetně těch statických.

```
#include \uv{myheaderfilename.h}

```

```
3 #include <stdio.h>
4 #include <stdint.h>
5
6 #include "module_calc.h"
7 #include "myperipherals.h"
8 #include "uv{utilityfc.h"
9
10 #include <ArduinoBLE.h> // BLE module
11 #include <Arduino.h> // arduino main framework
12
13 #define ERROR_MAX_LEN 128
14
15 extern MyEnumState state = MYENUM_UNKNOWN;
16 static called_this = 0;
17 char error_buf[ERROR_MAX_LEN];
18 char* last_error;
19
20 // Some private function doing own stuff
21 void LocalFunction() {
22
23     // do stuff
24 }
25
26
27 // Some public function doing something
28 void MODULE_DoIt(int* it) {
29
30     // do something to *it
31 }
```

Konfigurace

Téže by šlo označit za konfigurační modul, je samostatný hlavičkový soubor „config.h“.
V podstatě je to konfigurační soubor, ve kterém si student nastaví vlastní parametrizaci řešení, může jej však ponechat beze změny, vše je nastaveno tak, že je ihned možné použít.
Jedná se o prakticky jediný soubor, u kterého je „vyžadován“ zásah studenta, ale to vzhledem k předchozí větě není úplně pravdou. Mohou vzniknout exotické případy, kdy vážně vyžadována úprava jednoho nebo dvou parametrů, ale tím jsem chtěl hlavně říct, že ostatní soubory by neměly být měněny a pokud, tak je upravovat/rozšiřovat. Je to na zvážení studenta, nechávám je otevřené.

Commons

Hlavičkový soubor „commons.h“ je další a poslední jednosouborový „modul“. Commons by měl být vidět v celém řešení neboť upravuje globální definice.
Pokud by student měl potřebu parametrizovat nekteré definice v rámci celého řešení, tento soubor je pro jeho kód ideální. Taktéž se hodí na globální includy.

Modul FSM

Je nejzákladnějším modulem a jádrem celého řešení. Rozhodl jsem se, že jádrem řešení by měl být stavový automat (odtud název – Finite State Machine). Kromě toho, že se v drtivé většině případů používá stavový automat (popř. task switcher – pokud je možné a výhodné využívat vlákna a přerušení), je to i výhodné z hlediska přehlednosti a rozšiřování.
Stavový automat lze vytvořit dvěma způsoby a to:

- if-else statement – méně přehledný, hůře rozšiřitelný
- switch-case statement – pracnější ale přehlednější, jednoduché rozšíření

Já si zvolil switch-case metodu s enumerační proměnnou, která označuje jednotlivé stavy.
Sketch v rámci nekonečné smyčky volá funkci „FSM_Exec“, která s každým zavoláním vykoná část aplikačního kódu daného stavu a přejde do (jiného) definovaného stavu.
Stavy jsou definovány jako hodnoty enumerace FsmState. V současném kódu je celkem 12 stavů, z toho je 10 „systémových“ a 2 „uživatelské“. Jsou rozděleny do skupin (pro přehlednost).

Systémové stavy Tímto názvem označuji stavy kostry – páteře chcete li. Jsou to ty nejzákladnější stavy, které zaručují běh ve smyčce.

Nejdříve je popíšu z hlediska zařazení do skupiny a až poté (v samostatné kapitole 2.3.5) jejich návaznost a obsah.
Skupiny jsou definovány rozpětím hodnot, které enumerační proměnné přiřazuje v podobě hexadecimální hodnoty:

- Chybové stavy (0x0000 – 0x000F) – zachycení běžné chyby a neimplementovaný stav
- Manažerské stavy (0x0010 – 0x00FF) – hřbitov, inicializace, začátek a konec smyčky
- Stavy spojení (0x0A0 – 0x0AFF) – kontrola spojení po BLE, zápis a čtení BLE
- Bluetooth specifické stavy (0x0B00 – 0x0BFF) – stav pro pozdrav zařízení (tzv. Aloha)

Uživatelské stavy To jsou ty stavy, které by si sám student měl vytvářet a spravovat. Já jsem pro ukázku přidal stavy 2, které demonstrují použití s vlastním algoritmem řízení.
Též jsou rozdělené do skupin a jako u předchozí kapitoli popíšu nejdříve skupiny a až pak stavy samotné a jejich návaznosti. Skupiny jsou taktéž definovány rozpětím přiřazené hodnoty:

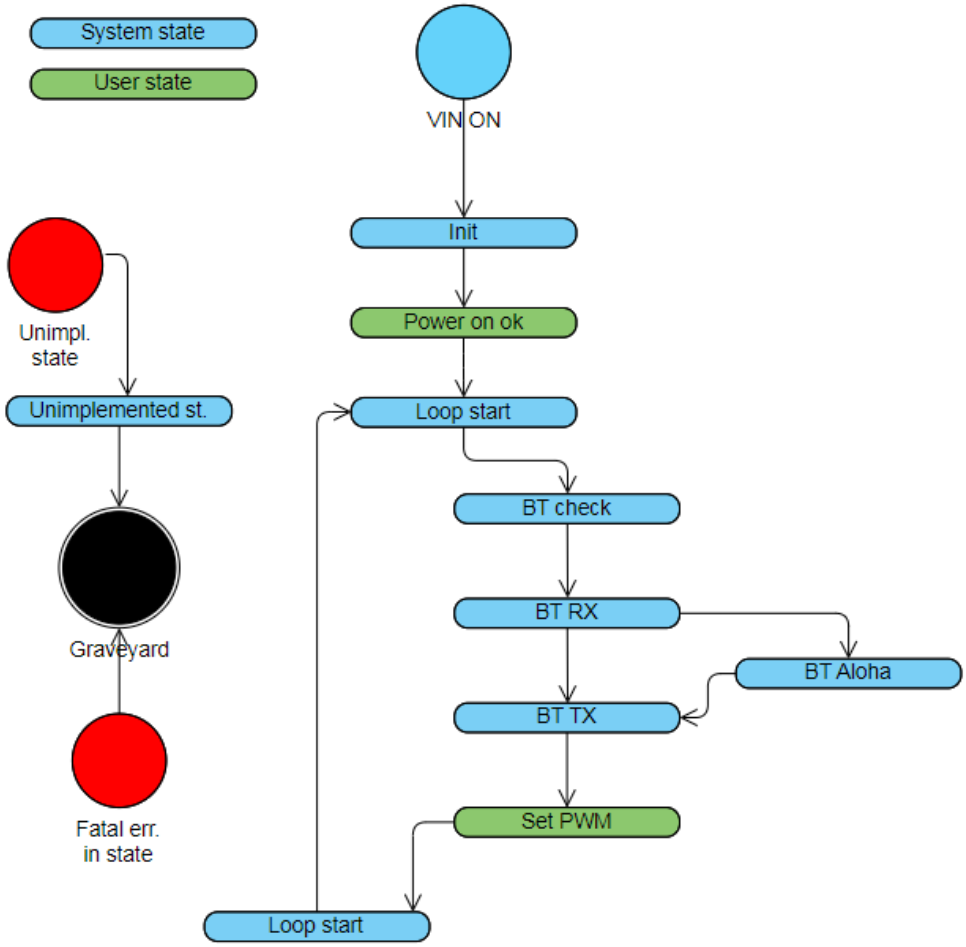
- Uživatelské stavy (0x0100 – 0x06FF) – např. stav signalizující, že vozidlo je připravené
- Uživatelské stavy smyčky (0x0700 – 0x09FF) – např. stav nastavující PWM

Popis stavů automatu Skupiny lze libovolně spojovat nebo rozdělovat v libovolném rozpětí, nynější struktura je pouze pomocná a lze na ni stavět.
Zde jsou definice všech stavů automatu:

```
1  typedef enum {
2
3      // Error states
4      FSMST_GENERR         = 0x0000, // general error handling
5      FSMST_STATEERR       = 0x0001, // unimplemented state
6
7      // Management states
8      FSMST_GRAVEY         = 0x0010, // graveyard / black hole
9      FSMST_INIT           = 0x0020, // inits user code
10     FSMST_START          = 0x0030, // start of a~FSM loop
11     FSMST_END            = 0x0040, // end of a~FSM loop
12
13     // User states
14     FSMST_PWRON          = 0x0100, // slot car is powered up
15
16     // User in-loop states
17     FSMST_GOSLOW         = 0x0700, // slot car drives at
           slow speed
18
19     // Connectivity state
20     FSMST_BTCHECK        = 0x0A00, // check if BT is
           connected
21     FSMST_RXCHECK        = 0x0A01, // check if there are
           data to receive
22     FSMST_TXCHECK        = 0x0A02, // check if there are
           data to send
23
24     // BT states
25     FSMST_ALOHA          = 0x0B00, // make aloha blinkin
26
```

```
27 } FsmState;
```

Zde si popíšeme stavy a přechod mezi nimi (můžete využít schéma přechodů na ilustraci 2.5. Hlubší popis funkci ve stavech bude popsán v modulu funkcí FSM.



Obrázek 2.5: Statový automat a přechod mezi stavy

Automat vždy začíná stavem FSMST_INIT, kde se provede resetování vozidla. Přechází v uživatelský stav FSMST_PWRON. ve stavu FSMST_PWRON se indikuje, že je vozidlo úspěšně inicializováno a přechází se na začátek hlavní smyčky-/rutiny automatu FSMST_STAR. Momentálně se na začátku kontroluje podmínky pro blikání LED diody. Dále se navštíví postupně stav FSMST_BTCHECK a jeli bluetooth připojené pak FSMST_RXCHECK a FSMST_TXCHECK, kde se zkouší číst a odesílat relevantní data.
FSMST_ALOHA aloha pro bluetooth je tu od toho, aby se dalo připojení signalizovat. Do tohoto stavu se přechází automaticky, pokud čtení bluetooth zprávy

obsahovalo požadavek na Alohu.
Po čtení a zápisu dat se přechází na uživatelský stav smyčky FSMST_GOSLOW, kde se nastaví PWM na určitou hodnotu (v době sepisování na poloviční, tj 127). To nastaví střídu PWM na adekvátní procentuální hodnotu. Došli jsme na konec, přesněji řečeno do konce vnitřní smyčky na stav FSMST_END, v dalším zavolání exekuce hlavní smyčky se přejde na FSMST_START, startovní pozice vnitřní smyčky.
Jsou zde stavy, které jsou volány mimo vnitřní smyčku a to je např. hřbitovní stav FSMST_GRAVEY. Jakmile se do něj dostanete, už v něm zůstanete, zaindikuje se červená dioda jako fatální chyba.
Také je zde stav FSMST_STATEERR, do kterého se dostanete vždy, pokud zapomenete implementovat do struktury switche svůj nový stav. Po debug zprávě skončíte na hřbitově.
Je tu také stav pro chybu, která není fatální (ve smyslu, že nekončí na hřbitově), ale je potřeba ji zashlásit např. po sériové lince. Tento stav je tu pod jménem FSMST_GENERR.

Modul funkcí FSM

Tento modul vznikl pro zpřehlednění FSM modulu. Zde jsou veškeré funkce, které jsou volány skrze stavy a nejsou přímou součástí nějakého modulu (např. pro sériovou komunikaci nebo Utils. Naopak v těchto funkcích zde, je voláno rozhraní ostatních a zejména periferních modulů.

Zde je popis jednotlivých funkcí:

- Void ReserCar() – funkce pro resetování vozidla do původního stavu, zhasne všechny LED diody, vygeneruje náhodný tisknutelný znak (ten si zapamatuje – viz. FsmStart()) a pošle debug hlášku po sériové lince, že je autíčko resetováno a připraveno.

- Void CarWorking() – moje uživatelská funkce, která rozsvítí zelenou LED, jako indikace, že je autíčko zresetováno. Ne vždy mám připojený terminál na odposlech debug zpráv.

- Void FsmStart() – je funkcí, která je volána na začátku vnitřní smyčky. Kontrolu je dle nastaveného intervalu v konfiguračním souboru, jestli mám zhasnout nebo rozsvítit zelenou LEDku. Tedy v podstatě se jedná o blikání. S každou periodou bliknutí zašle unikátní znak, který byl nastaven resetováním vozidla. To je pro čtenáře terminálu jasným znakem, že vozidlo bylo resetováno a vnitřní smyčka běží (ECU vozidla se nezasekla).

- Void Graveyard() – zaindikuje rozsvícení červené LED a zhasne ostatní. Jako indikátor, že je ECU v nekonečné smyčce (fatal error).

- Bool BluetoothCheck() – dotazuje se bluetooth modulu, jestli je spojení aktivní (a jestli nějaké existuje).

- Bool SignalAloha() – je debugovací funkce, která zabliká modrou LEDkou. Počet blikání a periodu lze nastavit ve zdrojovém souboru v horní části (ALOHA_BLINKS a ALOHA_BLINK_PERIOD).
- int BluetoothRx() – volá bluetooth modul a táže se, jestli nebyla přijata nějaká data. Ta následně zpracuje a vrací indikaci pro chování automatu.

Modul utilities

Je modulem pro knihovnu funkcí pomocných. Zde jsou umístěny funkce, které jsou použity napříč řešením, ale nejsou úzce spjaty s žádným jiným modulem.
Momentálně jsou zde funkce:

- UTILS_RandomPrintableChar – vybere náhodný tisknutelný znak a vrátí ho.
- UTILS_TimeElapsed – testuje uběhnutí času. Vstupním parametrem je celkový uběhnutý čas (nyní), čas testovaný a hranice. Pokud čas uběhl, je hodnota skutečného času zapsána do času testovaného – synchronizace bez kumulování chyby rozdílu.

Zde je ideální prostor proto, aby student rozšířil poskytnuté řešení o své vlastní funkce.

Modul sériové komunikace

Obsahuje funkce po komunikaci po USB s PC. Doporučuji např. použít Putty nebo odobný software. Klasicky virtualizace sériovou komunikaci na COMx portu, který je viditelný/vyhledatelný.
Funkce SERIAL_Init je pro inicializaci sériové komunikace. Změna baudrate je možná v souboru „config.h“, stejně tak v hlavičkovém souboru modulu je možné změnit maximální velikost zprávy.
Pak jsou zde 4 funkce pro zaslání debugovací zprávy. Jelikož je u Arduina nastaven kompilátor na striktní kontrolu typů, jsou zde 2 funkce pro zaslání debug zprávy s newline znaky a obě mají svoji mutaci pro const char* a char*. Obecně si dejte pozor, budete dostávat plno upozornění a chyb u přetypování.
Poslední funkce rozhraní modulu je na zaslání chybové hlášky, která je velmi podobná debugovací hlášce, ale je zvýrazněná.

Modul nastavení pinů

Zde v modulu jsou přítomny 2 funkce. Jedna je na inicializaci pinů dle nastavení v konfiguračním souboru a druhá je wrapper pro rozsvícení a zhasnutí diody.
Makra pro diody jsou přítomny v konfiguračním souboru.

Modul pro PWM

Modul pro PWM obsahuje 2 funkce. Inicializační funkci pro nastavení PWM pinu a pro nastavení střídy onoho pinu. Hodnota je v rozmezí bezznaménkové 8-bitu – čím vyšší číslo parametru tím vyšší procento střídy (max. 255). Dle dokumentace je možná frekvence max. 0,5 kHz.

Modul pro BLE

Tento modul je jeden velký wrapper okolo knihovny ArduinoBLE.
Jsou zde 4 funkce. Funkce inicializační – zařízení se začne chovat jako bluetooth server (centrála/central). Začne vysílat svoje ID. To je možné upravit v konfiguračním souboru stejně jako unikátní ID. ID může být libovolné (samozřejmě s omezením na počet znaků, což je cca 32 znaků), pod tímto řetězcem se objeví ve vašem mobilním zařízení. Co se týče UUID je zde potřeba dodržet jak formát tak délku. Musí to být řetězec který má 27 hexadecimálních znaků v dash case formě, jako je naznačen v souboru. Pro ukázku ho zmiňuji i zde i s ID:

```
1 #define BT_ID    "Nano33BLE-MyId"
2 #define UUID     "123e4567-e89b-12d3-a456-426655440000"
```

Nejdříve je potřeba desku a její BLE spárovat s mobilním zařízením. POZOR! Lze spárovat i s klasickým BT na mobilním zařízení, avšak nikdy nedojde k výměně informací (dat) a ve skutečnosti se nikdy nepropojí. Je to dáno nekompatibilitou BLE a BT. Je nutný buď mít dual mode nebo mít podporu BLE, vždy si to nejdříve ověřte.
Jakmile je zpárováno, automat by měl na základě výsledku stavu kontroly propojení přejít na čtení a zápis – výměně dat mezi deskou a zařízením. Jestli je zařízení přípojeno je indikováno rozsvícením modré LED diody.
Další funkcí je již trochu nastíněná funkce kontroly spojení. Pokud je zařízení spárované (ověřte v nastavení mobilního zařízení) a je aktivní spojení, modrá dioda je rozsvícená. U ztráty spojení nebo přerušením párování se modrá LEDka zhasne.
Poslední 2 funkce byly taktéž naznačeny a jedná se o zápis a čtení dat. Zařízení o sobě posílá v pravidelných intervalech data a kontroluje, jestli nějaká nepřišla. Tento proces je asynchronní a kontroluje se na pravidelné bázi synchronně (podobně jako DMA bez callbacku přerušení, zkrátka musíte to fyzicky v intervalech kontrolovat).

2.3.6 Poznatky a poznámky ke kódu

V kapitole o bezpečnostní normě (zde 2.3.3) jsem uvedl, že je potřeba try-catch blok. ve vývojovém prostředí Arduina je to trochu problematické.
Již dříve jsem se zmiňoval, že toto IDE má docela striktně nastavený překladač (např. na přetypování) a je tu i striktní nastavení, které zakazuje použití try-catch. Údajně je to proto, aby se šetřily prostředky a footprint. V návodu jsem se dočetl, že by se měl try-catch používat jen při vývoji a v produkci by měl být vypnut. Zde uvedu kód pro hlavní smyčku, která pracuje se zachytáváním výjimek.

```
1 try {
```

```
2
3       ExecFsm();
4
5   } catch(const std::runtime_error& re) {
6
7       Fatal(re.what());
8
9   } catch(const std::exception& ex) {
10      Fatal(ex.what());
11  }
12  catch(...) {
13
14      Fatal("Unknown exception");
15  }
```

Je ale nutné dodat, že je naprosto nezbytné použít parametr “-fexceptions” u překladače.

2.4 Nástroje pro usnadnění

Původně jsem byl rozhodnutý, že bych vytvořil Windows aplikaci, která by umožňovala zobrazení hodnot z vozidla. Po přečtení normy bezpečnosti jsem ale došel k závěru, že mohu využít komunikaci po BLE např. V mobilním zařízení. Složitostí vypracování se to liší jen mírně a vyhodnotil jsem to jako čerstvý a modernější přístup. Student by pak mohl získat údaje bez nutnosti připojení k notebooku – doslova do zařízení, které nosí v kapse.

2.4.1 Mobilní aplikace

Aplikace je psaná v Xamarinu (Microsoft, **C#**, Xaml) a řešení aplikace je vytvořeno ze 3 projektů:

- Základní – zde píšete kód
- Android – zde pouze úpravy pro Android
- iOS – zde pouze úpravy pro Apple

Pokud by student chtěl rozšiřovat řešení v Xamarinu, je nutná znalost vývoje aplikací v Xamarinu. Já zde budu hodně obecný, určitě to nenahradí kurz nebo návody na internetu.
Budu li mluvit o projektu – budu tím myslet základní projekt, do kterého se píše 95 % kódu a ostatní projekty z něj jen čerpají.
Student nemusí řešení ani projekt řešit, pokud nechce dále rozšiřovat nebo použít iOS. Bude dodána i vyexportovaná verze APK (v kapitole exportu popíšu případný postup – 2.4.1).

Nastavení projektu

Prvně zmíním balíčky, protože na tomto tématu se může při špatném nastavení a pochopení utrápit nejeden.
Používám v zásadě komunitní balíčky nebo přímo od microsoftu, zde je potřeba si jen pohlídat verze a dependencies, vzhledem ke stavu svého systému.
Pak je zde package s názvem Acr.User.Dialogs (specifická verze 7.0.0, ta se mi zdá přehledná a funkční). Skvělý balíček na práci s notifikacemi. Vřele doporučuju, pokud nechcete používat, můžete použít jiný nebo klasické intenty.
A posledním balíčkem je knihovna pro práci s BLE – Plugin.BLE (verze 2.1.3).
Ujistěte se, že korektně referencujete MonoAndroid.dll, máte správně nastavený soubor Manifestu pro Android.
Pro vytvoření a vývoj aplikace bylo použito:

- Visual Studio 2019 Community (16.11.26)
- Xamarin (16.11.000.204)
- Xamarin.Android.SDK (12.0.0.3)

- **C#** Tools (3.11.0)
- .Net Framework 4.8.03761
- Window 7 – x64

V případě technických problémů byste se co nejvíce měli přiblížit této konfiguraci. Otevřením projektu ve vyšší verzi Visual studia by mohlo způsobit problémy s kompatibilitou. Taktéž nedoporučuji aktualizovat kteroukoliv součást, pokud si nejste naprosto jistí, že víte, co děláte. To platí i (a zejména) u balíčků.
Ve visual studiu používám emulátor pro android zařízení. Mohu doporučit Pixel 2 Pie 9.0 – API 28. Je svižný a méně se zasekává než ostatní verze, co jsem zkoušel. Ujistěte se ale raději, že projekt znovu-sestavujete (Android). Taktéž doporučuju manuální odstranění složek "bin" a "obj". Pokud by nefungoval emulátor – ujistěte se, že jste přeplí do konfigurace Debug.
V případě jiných chyb sledujte výstupní konzoli.

Kód

V kódu je mnoho rutin vykonáváno asynchronně, to je potřeba mít na paměti.
Projekt je jedno stránková aplikace. Stránka MainPage má 2 sekce, z nichž je pouze jedna stále viditelná. A tou je sekce pro nastavení bluetooth. Druhou je samotné ovládání a ta je odkryta později.
Sekce pro nastavení bluetooth se opírá o vlastnosti balíčku Plugin.BLE, které jsou wrapovány v třídě BtManager. Zde jsou všechny základní vlastnosti a ovládání BLE, stačí jen vytvořit instanci a můžete volně použít.
Druhá sekce se odkryje po úspěšném připojení a odkazuje se na třídu Controls. Zde pro změnu nacházíme veškeré zasílání nebo zobrazování zpráv a hodnot.

Export

Budu zde popisovat jen postup pro operační systém Android.

1. Ujistěte se, že jste v konfiguraci Release (v Debug nebude fungovat)
2. Nastavte si potřebné parametry v manifestu nebo ve vlastnostech projektu pro Android
3. Ujistěte se, že minimální verzi Android verzi máte 5.0 (API 21), Cílová a užité konk. SDK je na Vás.
4. Vše uložte a udělejte clean build (znovu-sestavení)
5. Vyvolejte menu pravým kliknutím myši na projekt Android a vyberte možnost archivace
6. Zde bude seznam úspěšných sestav, nyní by seznam měl být prázdný – krom první pozice, kde se právě vytváří APK.

7. Pokud skončí chybou, zkuste některé z těchto kroků:
8. v případě úspěšného sestavení archivu klikněte na tlačítko distribuovat
9. Pravděpodobně budete mít na výběr jen jednu možnost a ta se názvem liší v závislosti na jazykové verzi. Ad-hoc možnost vyberte.
10. Pokud nemáte podpisovou identitu, nějakou si vytvořte nebo ji importujte. Následně ji zvolte a aplikaci podepište.
11. Vyberte výstupní složku (doporučuju kratší cestu a vyvarovat se speciálním znakům), pak vyplňte heslo, které jste zadávali u podpisové identity.
12. Výstupní APK dostaňte do mobilního zařízení s androidem.

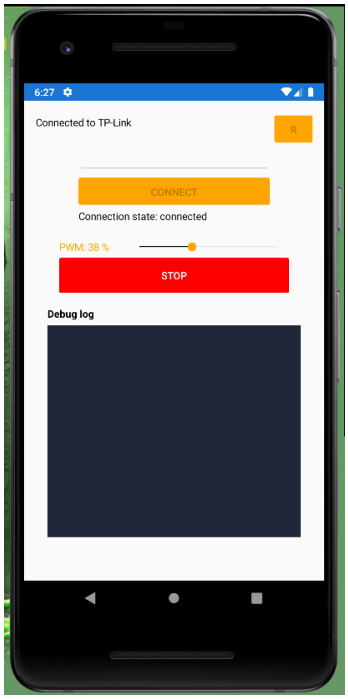
Prerekvizita a instalace

Nejdříve zmíním, že univerzální návod na instalaci není. Co model to samostatný návod. Mohu jen popsat z pohledu mého a zařízení, na kterém jsem aplikaci testoval.
Logickým předpokladem je mít zařízení se systémem Android. Já osobně zkoušel na zařízení Huawei Nova 3, který má Android verzi 9. Zařízení musí mít podporu BLE nebo Dual mode bluetooth. Což testované nemá (kapitola 2.4.3).
Pokud jste dostali do mobilního zařízení archív a operační systém ho třeba nezablokoval nebo nesmazal – pokuste se jej nainstalovat. Pravděpodobně archív identifikuje jako archív z neznámého zdroje.
Pro instalaci archivů z neznámých zdrojů musí telefon uživateli umožnit tuto ochranu ignorovat. Některá mobilní zařízení to neumožňují, např. některé modely od Samsungu. U jiných je to boj s nastavením. V mém případě to bylo na pár kliků.
Po instalaci aplikaci spusťte.
Výrazně doporučuji před každou instalací nejdříve aplikaci zavřít a odinstalovat. Ušetříte si tím spoustu bolesti hlav.

Návod na použití

Aplikace může už být spuštěná, ale nejdříve je potřeba mobilní zařízení spárovat s deskou ECU. To provedete v klasickém nastavení zařízení bluetooth. Ujistěte se, zelená LED u ECU bliká a dejte vyhledat změny. do 10 vteřin (timeout je různý) by se Vám mělo zobrazit zařízení s ID, které jste mu nastavili v projektu ECU. Pokuste se spárovat, je možné, že to napoprvé selže – opakujte, dokud se nepovede. Pozor, lze spárovat i s klasickým Bluetooth! Toto upozornění píši proto, že by někdo mohl nabýt mylného dojmu, že tímto krokem ověřil, že jeho mobilní zařízení podporuje dual mode nebo má BLE.
Nyní zpět do aplikace. Jsou zde 2 sekce (jak je vidět na snímku obrazovky emulátoru na 2.6), ale jedna je prozatím skrytá.
V sekci s nastavením Bluetooth naleznete 2 tlačítka a box s možností selekce. Dále jsou zde informativní texty, které by Vám v případě nejasností měli napovědět, kde

je chyba nebo co nyní probíhá.
Tlačítkem nadepsaným R obnovíte seznam spárovaných zařízení. Ujistěte se, že máte bluetooth zapnuté. Vyberte ECU s ID jaké jste ji nastavili a připojte se pomocí druhého tlačítka.
Pokud zařízení v seznamu nenaleznete, není zařízení spárované – celý proces párování opakujte (ideálně aplikaci zavřete nebo přeinstalujte).
Pokud se úspěšně připojíte k ECU, bude to signalizováno Alohou a na Vaší straně pak zpřístupněním druhé části stránky.
Zde máte veškeré ovládání a zobrazovadla. Je zde vidět aktuální hodnota PWM v procentech, možnost upravit a odeslat zpět do ECU. Jde zde tlačítko na nouzové zastavení vozidla. Po jeho klepnutí/stisknutí skončí vozidlo ve stavu hřbitova – signalizuje rozsvícením červené LED diody. V dolní části pak uvidíte zprávy, které vozidlo odesílá po sériové lince – nyní i do mobilního zařízení.



Obrázek 2.6: Mobilní aplikace pro komunikace po BLE

2.4.2 Komunikační protokol

Vymyslel jsem komunikační protokol, které není šifrovaný, ale je kódovaný. Někdo by mohl namítnout, že bezpečnostní norma to přikazuje. Není tomu však tak.
Komunikace může být nejen nešifrovaná, ale může být i zcela čitelná, co se zprávy a obsahu týče. Je to pouze na vůli výrobce a často je sáhnuto k šifrování (nejčastěji AES kombinace s výrobním číslem uložených v EEPROM), aby nebylo zařízení zkopírováno jistou zemí, která je levnými a nefunkčními kopiemi proslulá.
Kromě jiného norma vůbec komunikaci po BT nikterak neupravuje. Je to tedy zcela v mé režii. Proč jsem se rozhodl data nešifrovat? Mám proto 2 zcela logické důvody:

1. Svým charakterem je mobilní aplikace pouze pomocný nástroj na školní projekt. Z mého pohledu odposlech dat nehrozí a pokud ano, zařízení musí být spárováno a připojeno.

2. Komplikuje to úlohu a čitelnost, což je naopak jeden z požadavků normy bezpečnosti – trasování chyb.

Komunikační protokol používá různě velké pakety ne však delší než 256 bytů (verze 1.0). Paket je rozdělen do bloků, každý blok velikosti násobku 8 (bloky sestavené z byte-sub-bloků). Zde stručný popis jednotlivých bloků:

- Start byte [1] – speciální znak.
- Verze protokolu [1] – o jakou verzi protokolu se jedná, není zpětně kompatibilní (horní nibble Major, dolní nibble Minor – verze x.y).
- Id [4] – id zprávy, slouží k identifikaci a k odpovědím
- Typ [1] – o jaký typ zprávy se jedná (chyba, informace, potvrzení), pokud začíná bitově na 1 vyžaduje tzv. ACK).
- Subtyp [2] – konk. druh zprávy ("udělej bluetooth alohu"). Horní byte je reserved.
- Délka [2] – jak bude následující blok dlouhý.
- Obsah [0-X] – kde X je maximální podpora velikost zmenšená o velikost
- těla paketu pro danou verzi (v bytech)
- CRC16 [2] – kontrolní součet
- Stop byte [1] – speciální znak pro ukončení zprávy

Pro lepší vizualizaci je možné si prohlédnout schéma na ilustraci 2.7.

Start	Version		Id	Type	Subtpe		Length	Data		CRC16	Stop
1 byte	1 byte		4 bytes	1 byte	2 bytes		2 byte	X bytes	.....	2 byte	1 byte
Char	M	m	Int	Int	Reserve	Int	Int	Bytes	.....	Int	Char

Obrázek 2.7: Komunikační paket po BLE

Typy zpráv

Typy zpráv jsou jak v ECU tak v mobilní aplikaci reprezentovány jako enumerace. Je to tak nejjednodušší a nejpřehlednější, navíc lehce rozložitelné o další typy. ve skutečnosti některé z nich vůbec nevyužívám, ale umím si představit, že jej student ve svém řešení použije.
Velikost bloku 1 byte je podle mě dostatečným prostorem pro definici 128 různých typů zpráv. 128 proto, protože první bit je pro indikaci, že je potřeba ACK.

Zde jsou nastíněny a popsány:

- Message – obsahem je text
- Aloha – pozdrav, vynucení si zpětného ACK (Aloha má vždy ACK)
- Set – příkaz k nastavení hodnoty proměnné
- Get – čtení stavu určité proměnné
- Update – vynucený update informací
- Ack – potvrzení o přijetí zprávy/požadavku
- Em. – Pohotovostní příkaz

Subtypy zpráv

Subtypy jsou konk. příkazy, konk. požadavky a jako typ zprávy určuje kategorii pro zpracování, subtyp popisuje určitý postup. Zvolil jsem velikost 2 byty s tím, že horní byte je rezervovaný. 256 konk. zpráv nyní dohromady nevymyslím, je tedy dostatečně velký prostor. V budoucnu by ale nějaké zajímavější řešení mohlo hranici překročit – byť ne počtem ale např. vymezením rozsahů (0x0100+). Zde jsou uvedeny, které v současnosti používám:

- Debug message – zašle běžnou textovou zprávu
- Error message – zašle zprávu označenou jako chybová
- Fatal message – zašle zprávu označenou jako fatální chyba
- BtAloha – zasílá příkaz na Alohu
- General ACK – obecný ACK
- Emergency ACK – ACK na emergency
- Emergency stop – příkaz zastavení
- Set PWM – nastavení PWM na hodnotu
- Get PWM – žádost o zaslání hodnoty PWM
- Heartbeat – vynucení si ACK, ujištění se, že komunikace běží

2.4.3 Testování mobilní aplikace

Jelikož se mi nepodařilo sehnat zařízení, které by mělo operační systém android a umělo komunikovat s BLE, rozhodl jsem se jít delší, ale levnější cestou.
Pořídil jsem si Nano USB adaptér, který umí komunikovat po BLE a Bluetooth 4.0. Je to zařízení TP-link UB400. Zapojuje se do klasického USB portu v PC. Po instalaci ovladačů je zařízení připraveno k použití.
Rozkliknul jsem ikonu v tray liště a vyberal přidat zařízení – tím je pak dotyčné zařízení spárované.
Nyní bylo potřeba si napsat 2 testovací aplikace. Ty nebudu přikládat jako součást řešení, protože jsou pouze testovacího charakteru. Jedna se chovala jako server a druhá jako klient a v podstatě v daném scénáři buď simulovala připojený telefon k ECU, nebo předstírala ECU a poslala voleně náhodná data.
Tím jsem ověřil jak funkčnost komunikace ECU (server), tak i mobilního zařízení.

2.5 Test na dráze

Tento bod jsem bohužel nemohl plnohodnotně splnit. Otestoval jsem své teze a nástroje alespoň teoreticky. V následujícím textu přiblížím, proč se tomu tak stalo.
V podstatě a ve zkratce je to pouze nedostupností komponent na trhu. Ke zprovoznění a vyzkoušení návrhu řešení jsou potřeba tyto komponenty:

- Díly dráhy
- Model vozidla
- ECU
- H-můstek
- Mobilní aplikace
- Program ECU

V následujích kapitolách popíšu teoretické otestování.

2.5.1 Díly dráhy

Jeden ze základních HW požadavků, je mít sestavenou testovací dráhu, na které lze řešení ozkoušet. V útrobách budovy a na TUL, byla jedna taková dostupná. Vozítko se starou deskou lze vložit do drážky a pak regulovat vstupní napětí drážek pomocí joysticku nebo jít alternativní cestou a nastavit vidlicí konstatní napětí na maximu a přenechat řízení vozidla autonomnímu regulátoru, který student navrhl.
Tento požadavek je splněn.

2.5.2 Model vozidla

V předchozích kapitolách bylo zmíněno, že mi byly dodány dva modely vozidel. Jak bylo předestřeno, lze je považovat za totožná. Z hlediska HW je nutné ověřit několik požadavků:

- Elektromotor je funkční – ověřeno na laboratorních generátorech napětí (naměřen odběr 150 mA)
- Nápravy – nevniká zde velké nebo nadbytečné tření, zadní náprava a soukolí není poničené.
- Konektory – jak konenktory z elektromotoru, tak vodiče z napájení od dráhy nejsou zkorodované, zničené či přerušené. Ověřeno multimetrem a laboratorním generátorem el. proudu.
- Pogumování – u jednoho vozidla došlo k rozpadu jeho “pneumatik” na jedné z náprav. Po více než 10 letech došlo k jejich znehodnocení vlivem času a tření. Vyměněny z jiného modelu.

- Šasí vozidla – ujistil jsem se, že vše drží pohromadě a elektronika je dostatečně izolována od napájení dráhy.

2.5.3 ECU

Nově vybraná deska byla zasazena do útrob vozidla. Je možné ji dodatečně izolovat od potenciální okolní elektroniky pomocí extra plastu, či speciální nevodivé hmoty, se kterou byla deska dodána.
Deska je přítomna. Ověřeno, že funguje, naprogramoval jsem základní obsluhu periférií, přes které jsem prověřil funkčnost. Tento bod je plně splněn neboť je dráha funkční.

2.5.4 H-můstek

K současnému řešení ECU je zapotřebí H-můstek, tedy mezikus, který by byl napájený přímo z kolejnice, ale řízený PWM od ECU. Na výběr bylo několik řešení, které jsem si vyhlédl. Bohužel i které postupně selhávaly na požadavcích.

Po dlouhé odmlce z hlediska dostupnosti desek s H-můstkem, jsem nakonec zvoli a objednal "Motor driver l293D" na webu dratek.cz. Bohužel mi nestačila deska přijít při sepisování diplomové práce, tedy nemohu jí otestovat a udělat tak patřičné závěry. Nicméně věřím, že se mi ji podaří otestovat až po odevzdání DP a předvést její funkčnost (včetně celého řešení) u obhajoby u SZZ.

2.5.5 Mobilní aplikace

Mobilní aplikace byla jednou ze součástí bodu usnadňující studentovi vyvíjet a ladit jeho regulační autonomní algoritmus.
I když fyzicky nebylo možné přímo vyzkoušet funkcionalitu mezi ECU a mobilním zařízením (kvůli nepřítomnosti mobilního zařízení podporující BLE), byla funkcionalita vyzkoušena přes 2 interní testovací aplikace, které simulovali roli jednoho nebo druhého zařízení při komunikaci.

Zde je seznam zařízení, které ChatGPT doporučil v dubnu 2023 jako vhodná zařízení pro testování mobilní aplikace. Parametry byla dostupnost a nízká cena na bazaru (relevanci nechám čistě na čtenáři):

- Xiaomi Redmi 9A – Released in June 2020, cca 2,500 Kč
- Samsung Galaxy A02s – Released in January 2021, cca 3,500 Kč
- Motorola Moto G9 Play – Released in August 2020, cca 4,000 Kč
- Nokia 2.4 – Released in December 2020, cca 3,500 Kč
- Oppo A15 – Released in October 2020, cca 3,500 Kč

- LG K42 – Released in January 2021, cca 4,000 Kč
- Realme C11 – Released in July 2020, cca 3,500 Kč
- HTC Wildfire E2 – Released in August 2020, cca 4,500 Kč
- Sony Xperia L4 – Released in March 2020, cca 6,000 Kč
- Lenovo K12 Note – Released in September 2020, cca 4,500 Kč

2.5.6 Program ECU

Vytvořil jsem lehce rozšiřitelný základ kódu pro oblsuhu nové platformy. Je jednoduše zkompilovatelný a lze jej nahrát do do dané desky. Základem kódu je FSM s oblushou hlavínch stavů. Druhou částí je vlastní jednoduchá ukázka, jak stavy rozšířit o vlastní řešení.
Veškerou funkcionalitu jsem ověřil na debugovacích nástrojích (sériová linka, komunikace po BLE) a komunikaci s protikusem.

Závěr

Tato diplomová práce vznikla záměrně za účelem renovace ročníkového projektu na ústavu MTI.
Primární motivací bylo projekt nejen obnovit, ale rozšířit o užitečné nástroje, aby se student, který by si projekt vybral, mohl soustředit na seberealizaci v rámci návrhu vlastního regulačního algoritmu pro autonomní řízení modelu vozidla.
V první řadě bylo potřeba vyzkoumat, čím stávající ECU nahradit. Našel jsem nové řešení, které vyhovuje omezením v podobě desky Arduino Nano 33 BLE. Splňuje většinu požadavků, je výkonnější a dostupná. Navíc je rozšířená o modul BLE, který umožnil rozšíření o další možnosti použití. I přes několik nedostatků je deska rozšiřitelná o další HW a moduly – další možnost seberealizace studenta.
Desku jsem oživil a vytvořil základní projekt a architekturu, kterou student může využít a sám si ji upravit. Vše je okomentované a zdokumentované v práci a kódu. Kód je psaný v souladu s bepečnostní normou ČSN EN 61508-3.
Vytvořil jsem mobilní aplikaci, jako nástroj pro vzdálené ovládání. Je zde univerzální komunikační protokol po BLE, který student může rozšířit o vlastní zprávy. Je k dispozici i zdrojový kód – je zde další možnost rozšíření zobrazování a ovládání. Návod na práci s aplikací je uveden v práci.
V teoretických testech a na laboratorním zařízení jsem otestoval, že navrhované řešení funguje. BLE komunikaci, sériovou komunikaci, stavový automat jsem otestoval v domacích podmínkách pomocí testovacích scénářů, pomocí TP-link UB400, vlastních WinForms test aplikací a pomocí Android device emulátorů. A tím splnil i poslední bod zadání.
Při realizaci jsem se významně opíral o vlastní zkušenosti a to zejména z pracovního života (programátor od roku 2014).

Použitá literatura

[1] WIKIPEDIA. *Freescale buyout* [online]. 2023. [cit. 2022-05-12]. Dostupné z: https://en.wikipedia.org/wiki/Freescale_Semiconductor#Buyout.

[2] VOŠOUST, Petr. *Elektronická řídicí jednotka samořízeného modelu vozu: Electronic control unit for self-driving car model*. 2022. Magisterský projekt. Technická univerzita v Liberci, Liberec. Vedoucí práce Jan KOPRNICKÝ.

[3] HONS, Petr. *Elektronická řídicí jednotka pro účely samořízeného modelu auta: Electronical control unit for driveless car model*. 2017. Dostupné také z: https://dspace.tul.cz/handle/15240/49723. Diplomová práce. Technická univerzita v Liberci, Liberec. Vedoucí práce Petr MRÁZEK.

[4] ARDUINO. *Arduino Nano 33 BLE* [online]. 2022. [cit. 2022-05-30]. Dostupné z: https://store.arduino.cc/products/arduino-nano-33-ble.

[5] ARDUINO. *Arduino Nano 33 BLE - datasheet* [online]. 2022. [cit. 2022-05-27]. Dostupné z: https://docs.arduino.cc/resources/datasheets/ABX00030-datasheet.pdf.

[6] WIKIPEDIA. *BLE kompatibilita* [online]. 2023. [cit. 2023-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy#Compatibility.

[7] DRATEK.CZ. *L298N dual specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://dratek.cz/arduino/877-arduino-h-mustek-pro-krokovy-motor-l298n-dual-h-most-dc.html.

[8] POLOLU. *DRV8833 specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://www.pololu.com/product/2130.

[9] DRATEK.CZ. *L293D specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://dratek.cz/arduino/916-motor-driver-l293d.htm.

[10] HADEX. *BTS7960B specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://hadex.cz/m536-h-mustek-bts7960b-pro-stejnosmerne-motory/.

[11] POLOLU. *VNH2SP30 specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://www.pololu.com/product/706.

[12] INSTRUMENTS, Texas. *SN754410NE specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://www.ti.com/product/SN754410/part-details/SN754410NE.

[13] DIGIKEY. *MC33886 specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://www.digikey.at/htmldatasheets/production/68198/0/0/1/mc33886pvw.html.

[14] BRIV. *IBT-2 specifics* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://www.briv.cz/p/5665/ibt-2-h-bridge-pwm-6-27v-43a-arduino.

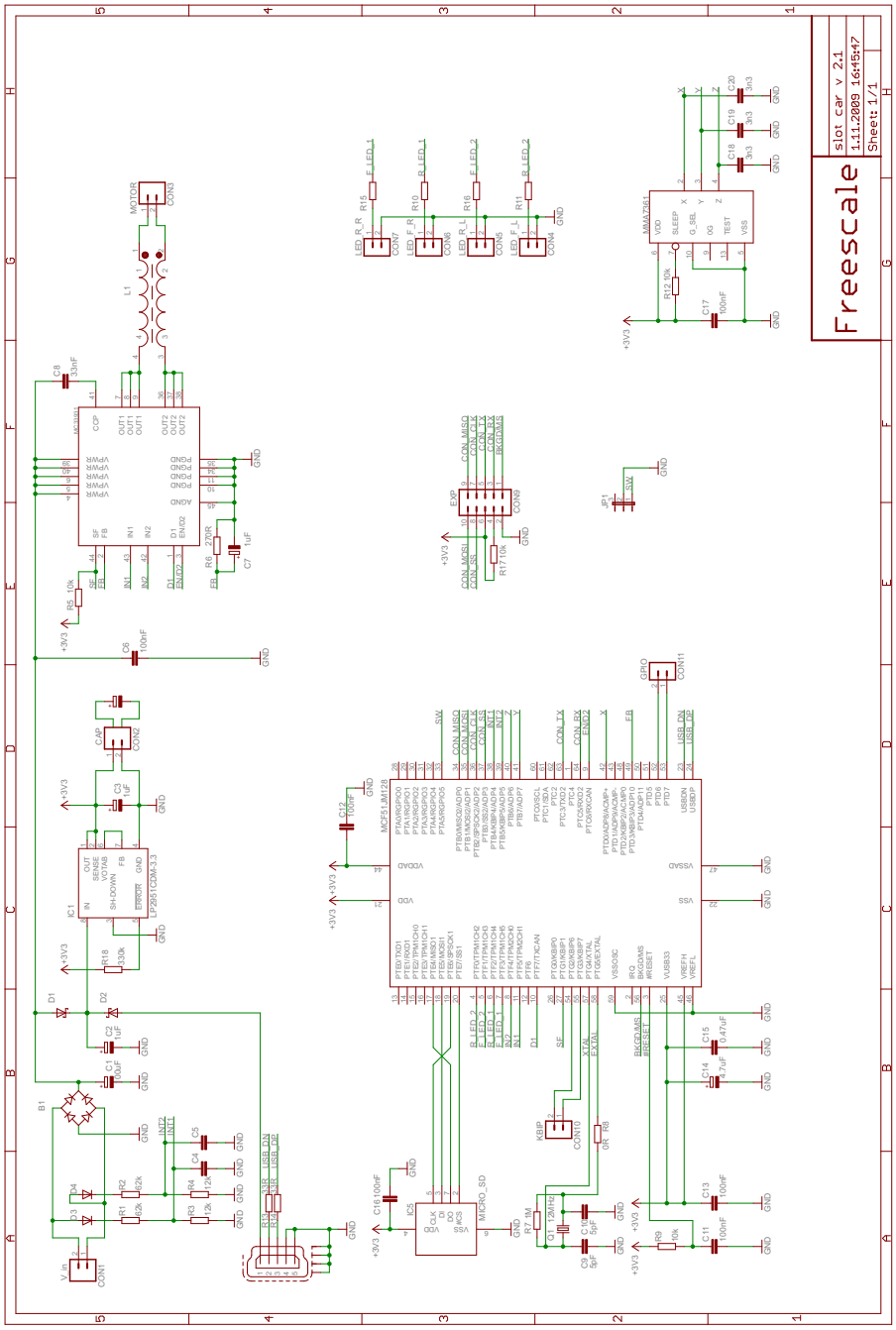
A Přílohy

A.1 Obsah vloženého balíku do IS/STAG TUL

- Schéma staré desky ECU
- Schéma Arduino Nano 33 BLE
- Pinout pro Arduino Nano 33 BLE

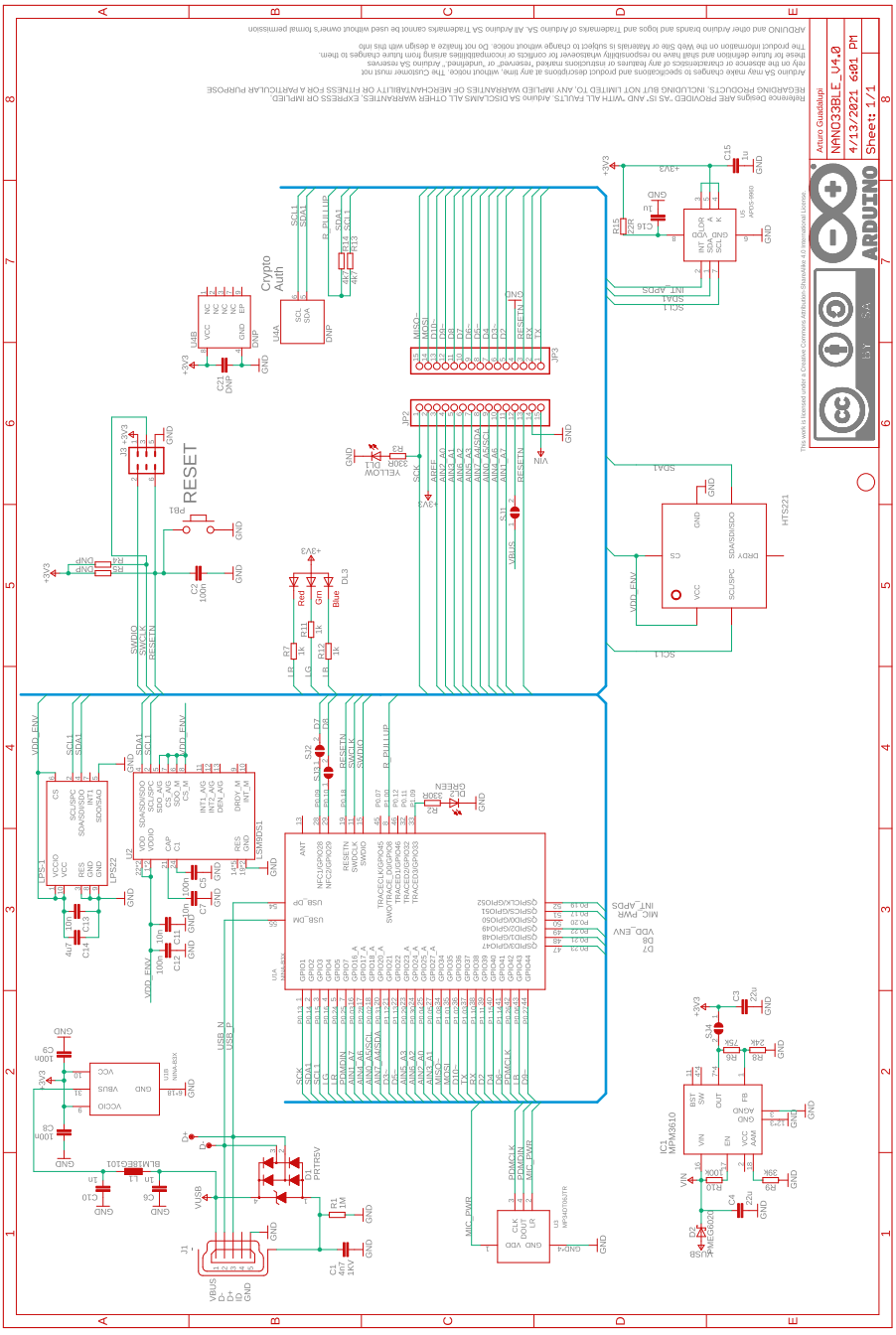
A.2 Schémata

A.2.1 Schéma staré ECU (Slot car v2.1)



1.11.2009 16:48:03 D:\berem\zaloha PC\work\autickо v2.0\slot car v 2.1\slot car v 2.1.sch (Sheet: 1/1)

A.2.2 Nano 33 BLE (Arduino, v. 4)



A.3 Velké tabulky a obrázky

A.3.1 Skutečný model vozu Audi R8 LMS GT3, 2022

A.3.2 Skutečný model vozu BMW Z4 M Coupé E86 GTE, 2012)

A.4 Média

Odevzdáno CD se zdrojovým kódem pro Arduino a Xamarin. Odevzdán archív se zabalenými projekty.

A.5 Zdrojové kódy

Na vyžádání u dok. Koprnického.