VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SOFTWAROVÝ PALUBNÍ POČÍTAČ DO AUTOMOBILU S ROZHRANÍM KWP-1281

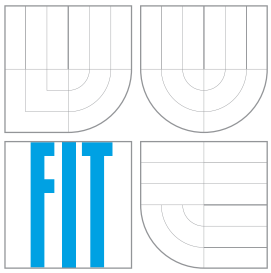
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATEJ MACHÁČ

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SOFTWAROVÝ PALUBNÍ POČÍTAČ DO AUTOMOBILU S ROZHRANÍM KWP-1281

SOFTWARE ONBOARD COMPUTER FOR AUTOMOBILE WITH KWP-1281 INTERFACE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATEJ MACHÁČ

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2013

Abstrakt

Cieľom tejto bakalárskej práce je vytvoriť systém, ktorý bude fungovať ako palubný počítač pre automobily s diagnostickým konektorom OBD2 a komunikačným protokolom KWP-1281. Hlavným cieľom je získať údaje o spotrebe automobilu a vhodne ich zobraziť na telefóne s OS Android. Implementácia systému prebehla na platforme FITkit a telefóne s OS Android. Pri vývoji aplikácie pre Android bol použitý jazyk Java a pri vývoji programu pre FITkit bol použitý jazyk C.

Abstract

The goal of this bachelor's thesis is to develop a system that will work as on-board computer for cars with diagnostic connector OBD2 and communication protocol KWP-1281. The main goal is to get information about the fuel consumption and show it properly on smartphone with OS Android. The system is implemented on platform FITkit and smartphone with OS Android. Java language was used to implement the application for Android, C language was used to implement the program for FITkit.

Klíčová slova

Palubný počítač, KWP-1281, Android, FITkit, Bluetooth, GPS

Keywords

Onboard computer, KWP-1281, Android, FITkit, Bluetooth, GPS

Citace

Matej Macháč: Softwarový palubní počítač do automobilu s rozhraním KWP-1281, bakalářská práce, Brno, FIT VUT v Brně, 2013

Softwarový palubní počítač do automobilu s rozhraním KWP-1281

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Doc. Ing. Adama Herouta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Matej Macháč
14. května 2013

Poděkování

Ďakujem svojmu vedúcemu Doc. Ing. Adamovi Heroutovi, Ph.D. za odborné vedenie a podnety, ktoré mi poskytol.

Obsah

Kapitola 1

Úvod

Palubný počítač v automobile je v súčasnosti uz štandardnou výbavou každého nového auta a ľudia si už zvykli, že vždy majú po ruke informácie o spotrebe palive. Len pár rokov dozadu, výrobcovia automobilov nižšej triedy nedávali do áut palubný počítač.

Ja sám som vlastníkom takéhoto automobilu - Škoda Felícia rok výroby 2000. Ako študenta ma samozrejme zaujíma, koľko benzínu spálim. Spotreba automobilu totiž nie je zaujímavá iba z finančného hľadiska, ale neobvykle zvýšená spotreba prezradzuje, že s autom nie je niečo v poriadku. Navyše, začal som sa zaujímať o diagnostiku motorov a kúpil som si diagnostický USB kábel s diagnostickým softvérom, s ktorým som bol schopný čítať dáta o behu motora z riadiacej jednotky v reálnom čase. Zamyslel som sa nad tým, či by som bol schopný tieto dáta získať z riadiacej jednotky a z nich vyčítať spotrebu paliva.

Ponúka sa otázka, prečo si nekúpiť originálny palubný počítač, ale miesto toho vymýšľať akýsi softwarový palubný počítač? Originálny palubný počítač sa už totiž nevyrába. A dokonca je ťažké ho zohnať aj použitý z druhej ruky a jeho cena sa pohybuje okolo 3000kč, čo je dosť veľa za minimálne 10 rokov používaný prístroj. Navyše jeho montáž je dosť náročná.

Preto som sa rozhodol, že vytvorím vlastný softwarový palubný počítač. Mojou prvou myšlienkou bolo použiť notebook pripojený USB káblom k diagnostickému konektoru. Musel som si naštudovať diagnostický protokol KWP-1281 a komunikáciu po sériovom porte. Podarilo sa mi vytvoriť funkčnú aplikáciu, ale toto riešenie bolo veľmi nemotorné.

Bolo by vhodné nahradiť notebook niečím menším a skladnejším – tabletom, alebo telefónom s OS Android. O tom, čo z toho vzišlo sa môžete dočítať v tejto bakalárskej práci.

Zadanie tejto bakalárskej práce bolo vytvorené v čase, keď ešte nebolo jasné, aká bude výsledná forma celého systému. Riešenie tak nekopíruje celkom presne zadanie, pretože po konzultácii s vedúcim bakalárskej práce sme uvážili za vhodné obmeniť bod 5. a 6. tak, že dáta zozbierané za jazdy sa nebudú analyzovať na stolovom počítači, ale priamo v telefóne alebo tablete s OS Android. Rozhodli sme sa tak preto, lebo sme sa zhodli na tom, že pre užívateľa bude praktickejšie a intuitívnejšie prezerať si tieto dáta na tom istom zariadení, ktoré mu ich zobrazuje za jazdy.

Druhou odchýlkou od zadania bola nutnosť vložiť do systému FITkit a navrhnúť a implementovať softvér pre toto zariadenie. Bez tejto zmeny by bol systém nefunkčný.

Kapitola 2

Komunikačný protokol KWP-1281

Komunikačný protokol KWP-1281 je prvý diagnostický protokol vyvinutý skupinou Volkswagen Audi Group (VAG), do ktorej patrí okrem iných aj Škoda Auto. Je definovaný v štandarde ISO 9141-2. Protokol bol predstavený v roku 1990 a bol široko využívaný viac ako 10 rokov. V niektorých autách bol dokonca naďalej používaný aj po predstavení nového protokolu KWP-2000. Použitý je napríklad v aute Audi A4 od roku 1995, VW Transporter od roku 1992, ale aj Škoda Felícia, ktorú vlastním ja. KWP-1281 poskytuje viac ako 10 diagnostických funkcií, medzi ktoré patrí napríklad čítanie závad motora, mazanie závad motora, nastavenie škrtiacej klapky, čítanie dát zo senzorov v reálnom čase. Protokol KWP-1281 bol po roku 2000 nahradený protokolom KWP-2000. V súčasnosti je v nových autách koncernu Volkswagen na potreby diagnostiky používaný protokol CAN.

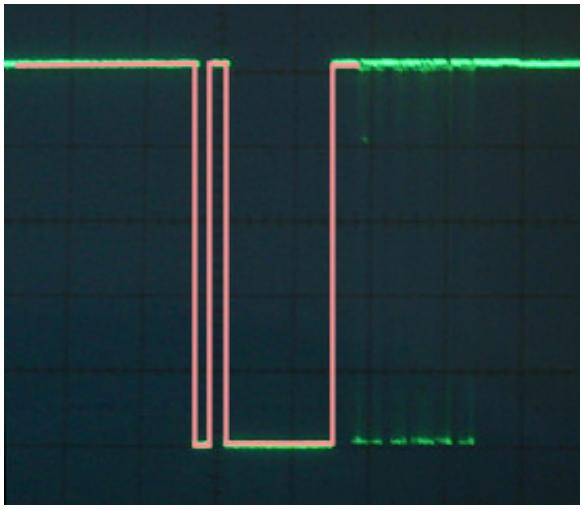
2.1 ISO 1941 (K-Line)

Norma ISO 1941 popisuje komunikáciu na fyzickej vrstve, okrem iných aj komunikačný kanál K-Line, ktorý je použitý v aute Škoda Felícia na diagnostiku [1]. Napäťové úrovne sú odvodené od palubného napätia a kostry vozidla. Keď je napäťová úroveň na K-Line väčšia ako 60% palubného napätia, je vyhodnotená ako log. 1. Ak je táto úroveň nižšia ako 30%, je vyhodnotená ako log. 0.

K-Line je založený na asynchrónnom prenose, je teda potrebné prenášať start bit, ktorého zostupná hrana je použitá na synchronizáciu. Po ňom nasleduje 1 byte (8 bitov), ktorý je nositeľom dát. Ako posledný je stop bit, ktorý nastaví úroveň do pôvodného stavu. Pozície bitov v slove sú určené podľa LSB - ako prvý je najmenej významný bit.

2.2 Inicializácia komunikácie

Aby bolo vôbec možné komunikovať s riadiacou jednotkou, je potrebné ju najskôr zobudiť. Túto inicializáciu vykonáva diagnostické zariadenie - v našom prípade FITkit. Ten musí poslať po K-Line adresu riadiacej jednotky - 0x01 extrémne pomalou rýchlosťou 5 Baud. K tomuto bajtu sa ešte pridáva start bit a stop bit, takže čas potrebný na poslanie inicializačnej adresy je 2 sekundy [4].



Obrázek 2.1: Inicializačná sekvencia, ktorú je potrebné zaslať riadiacej jednotke extrémne nízkou rýchlosťou 5 Baud po kanále K-Line, aby začala komunikovať.

Ihneď po poslaní sa musí prepnúť rýchlosť prenosu na 9600 Bd a očakávať prijatie 3 bajtov z riadiacej jednotky - 0x55, 0x01 a 0x8A. Odpoveďou mikrokontroléru je potom doplnok k 0x8A a teda 0x75.

Do RJ	0x01				0x75
Z RJ		0x55	0x01	0x8A	
Rýchlosť	5Bd	9600Bd			

Tabulka 2.1: Tabuľka zobrazujúca prvých 5 bajtov komunikácie medzi diagnostickým zariadením a riadiacou jednotkou. Komunikáciu iniciuje diagnostické zariadenie zaslaním bajtu 0x01 rýchlosťou 5 Baud

Tieto dva bajty musia byť poslané ihneď, pretože po približne 500 ms sa stane riadiaca jednotka neaktívnou a bolo by potrebné ju znova zobudiť inicializačnou sekvenciou. Poslaním tohto bajtu sa ukončí inicializácia a riadiaca jednotka pošle svoje ID. Na každý prijatý bajt je treba odpovedať jeho doplnkom.

2.3 Dáta zo senzorov

Z riadiacej jednotky je možné počas jazdy získať množstvo zaujímavých informácií v reálnom čase zo senzorov umiestnených priamo v motore, v prevodovke, alebo napríklad aj vo výfukovom systéme. Týmito informáciami sú teplota oleja, poloha škrtiacej klapky, rýchlosť otáčania motora, dĺžka vstrekovania paliva a veľa ďalších. Niektoré autá (napr. Audi A4) poskytujú dokonca aj údaj o rýchlosti vozidla, no Škoda Felícia touto informáciou v digitálnej forme nedisponuje, pretože má mechanický tachometer.

Vyššie spomenuté dáta neposkytuje riadiaca jednotka po jednom, ale v skupinách po štyroch. Diagnostické zariadenie musí požiadať o takúto skupinu zaslaním bloku, ktorý obsahuje požiadavku na čítanie konkrétnej skupiny (každá skupina má svoje číslo). Riadiaca

jednotka na to reaguje zaslaním bloku obsahujúceho nami požadované dáta. Opäť treba na každý prijatý bajt odpovedať jeho doplnkom a takisto riadiaca jednotka odpovedá na každý ňou prijatý bajt doplnkom.

Tieto dáta je však treba ešte správne interpretovať, pretože sú upravené v riadiacej jednotke na celé číslo v rozsahu 0-255 (1 Bajt). Pre každú veličinu potom existuje konštanta, ktorou treba vynásobiť prijatú hodnotu. Ak chceme zistiť napríklad rýchlosť otáčania motora, musíme prijatú hodnotu vynásobiť číslom 37. Rozsah otáčok motora môže teda byť teoreticky 0–255*37, čo je rovné 0–9435.

Ak je potrebné zistiť nejaké desatinné číslo, napríklad napätie na akumulátore, je v tomto prípade treba prijaté číslo vynásobiť konštantou 0,1. Získame tak teoretický rozsah 0–25,5V. Reálne sa však toto napätie pohybuje medzi 12–14,4V [5].

Kapitola 3

Návrh systému na meranie spotreby

3.1 Potrebné údaje

Pre zistenie aktuálnej spotreby automobilu potrebujeme dve základné hodnoty, a to objem paliva, ktoré sa spáli za jednotku času v l/hod a rýchlosť vozidla v km/hod. Spotreba vozidla v l/100km sa priamo úmerne zvyšuje so stúpajúcim prietokom paliva a naopak znižuje so stúpajúcou rýchlosťou (samozrejme, pri konštantnom prietoku paliva).

3.1.1 Prietok paliva

Prietok, alebo objem spáleného paliva za jednotku času je možné získať dvoma spôsobmi:

1. Pomocou prietokového senzoru umiestneného medzi palivovú nádrž a motor. Túto možnosť využíva továrenský palubný počítač tc-6, ktorý bol dostupný ako doplnková výbava.
2. Pomocou komunikácie s riadiacou jednotkou motoru. Z riadiacej jednotky sa síce nedá získať priamo požadovaná hodnota, avšak je možné ju vypočítať z hodnôt, ktoré nám riadiaca jednotka poskytuje:
   (a) Dĺžka vstreku
   (b) Otáčky motora
   (c) Akcelerácia/Decelerácia

Ja som sa rozhodol ísť cestou komunikácie s riadiacou jednotkou. Táto možnosť je síce náročnejšia na programovanie, ale je lacnejšia, dostupnejšia, jednoduchšia na inštaláciu do auta a bezpečnejšia, pretože nie je potrebné narúšať palivový systém auta.

3.1.2 Rýchlosť automobilu

Tachometer na Škode Felícia je mechanický a riadiaca jednotka nemá žiadne informácie o rýchlosti auta. Údaj o rýchlosti teda musíme získať nejakým iným spôsobom.

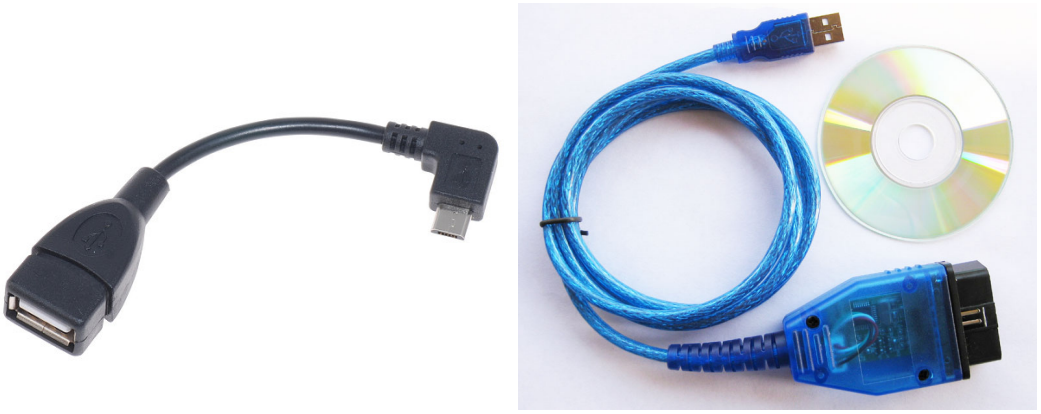
Továrenský palubný počítač tc-6 získava údaj o rýchlosti pomocou elektromagnetického snímača, ktorý je umiestnený za tachometrom na prístrojovej doske. Tento snímač však

nemeria reálnu rýchlosť, ale iba počet otáčok kolies. Pri montáži je teda nutné rozobrať celú prístrojovú dosku.

Toto riešenie je pre mňa nevhodné, pretože nechcem nijako zasahovať do konštrukcie automobilu. Preto som sa rozhodol, že pre zisťovanie aktuálnej rýchlosti použijem GPS prijímač zabudovaný vo väčšine smartfónov s Androidom. Zisťovanie rýchlosti pomocou GPS je síce menej presné pri prudšom zrýchľovaní a spomaľovaní, no väčšinu času je presnejšie ako mechanické meranie rýchlosti. Mechanické meranie rýchlosti je totiž ovplyvňované rozmerom pneumatík, ako aj ich nahustením (rozdiel 1cm v polomere pneumatiky spôsobí odchýlku v rýchlosti až 6%).

3.2 Prvotné riešenie

Prvotným zámerom bolo použiť štandardný USB OBD2 kábel určený na diagnostiku porúch motora pripojený k telefónu s OS Android cez USB OTG. Tento kábel je bežne dostupný a jeho cena sa pohybuje okolo 200Kč.



Obrázek 3.1: USB OTG a OBD2 USB diagnostický kábel použitý na komunikáciu PC s riadiacou jednotkou automobilu

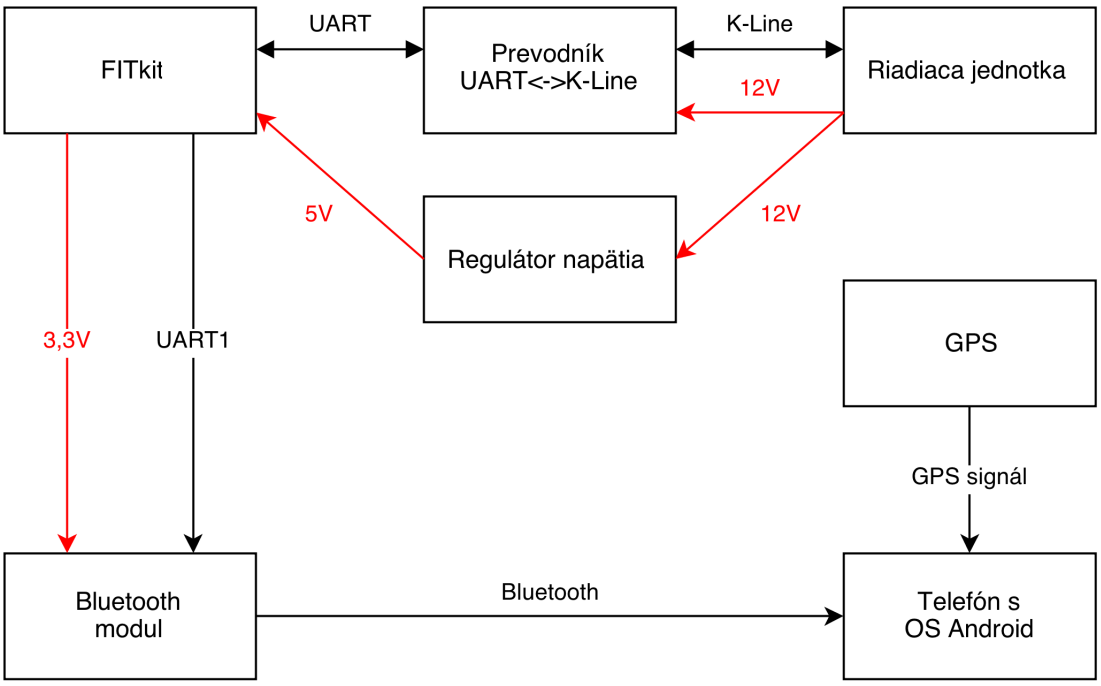
Tento kábel obsahuje FTDI prevodník USB – RS232 a následne RS232 – K-Line. Pomocou tohoto kábla zapojeného do PC sa mi podarilo inicializovať komunikáciu a následne aj plnohodnotne komunikovať s riadiacou jednotkou. Problém nastal po pripojení k Android zariadeniu cez USB host.

3.2.1 Ovládače pre FTDI prevodník v OS Android

Po pripojení OBD2 USB káblu k telefónu s OS Android je tento kábel rozpoznaný, no Android s ním nedokáže nijako pracovať - je potrebné nainštalovať ovládač. Zdrojové kódy tohoto ovládača sú dostupné, no je nutné ich pred použitím skompilovať s jadrom systému. To znamená nájsť presne tú verziu jadra, aká je v telefóne, stiahnuť ju, preložiť ovládač s týmto jadrom a až potom nahrať ovládač do zariadenia. Tento postup však nie je vždy úspešný. Mne sa to nepodarilo a tak isto sa to nepodarilo veľa ľuďom z komunity, ktorá sa zaoberá vývojom aplikácii pre Android. Dokonca niektoré zariadenia sú známe tým, že to na nich jednoducho nefunguje.

3.3 FITkit ako medzičlánok

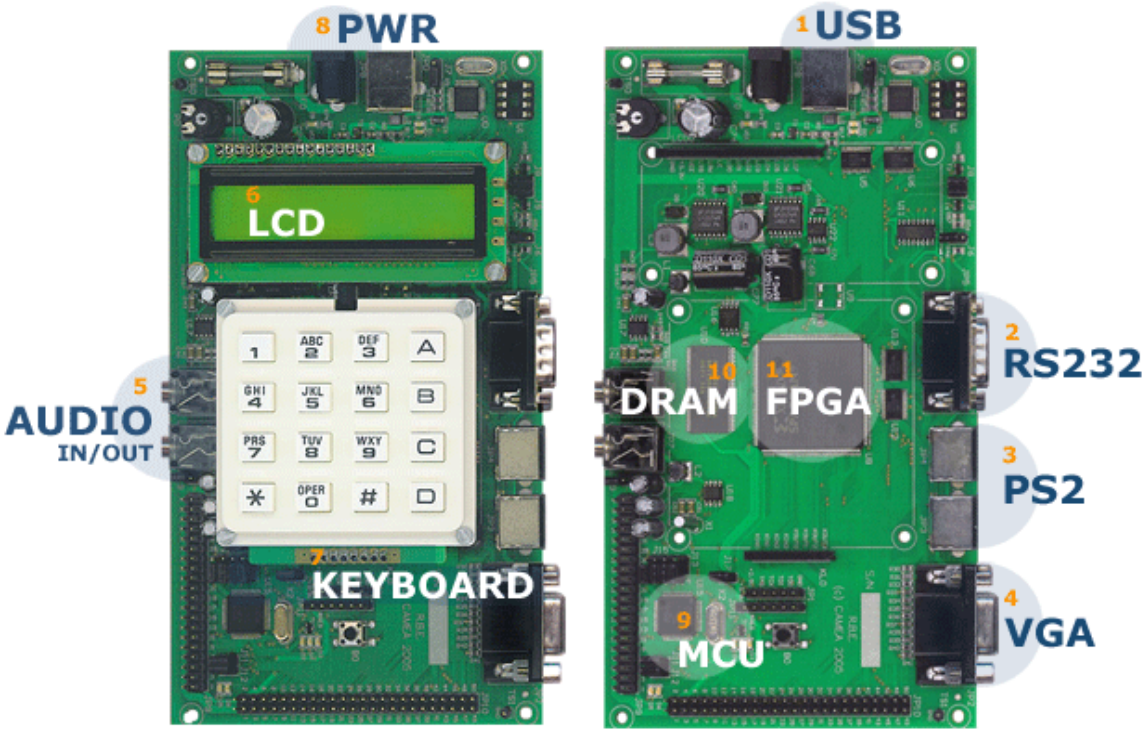
Je teda potrebné použiť akýsi medzičlánok, prostredníka, ktorý by zobudil riadiacu jednotku, získal z nej dáta a poslal ich telefónu. Vybral som si FITkit, pretože obsahuje mikroprocesor MSP430F2617TPM, ktorý má dve sériové rozhrania UART0 a UART1. UART0 budeme potrebovať na komunikáciu s riadiacou jednotkou a UART1 na komunikáciu s telefónom.



Obrázek 3.2: Diagram popisujúci prepojenie všetkých prvkov systému

3.3.1 FITkit

FITkit je samostatný hardware, ktorý obsahuje výkonný mikrokontrolér s nízkym príkonom, hradlové pole FPGA (anglicky Field Programmable Gate Array) a radu periférií. Software pre mikrokontrolér sa tvorí v jazyku C a do spustiteľnej formy sa prekladá pomocou GNU prekladača, ktorý je možné používať zdarma.

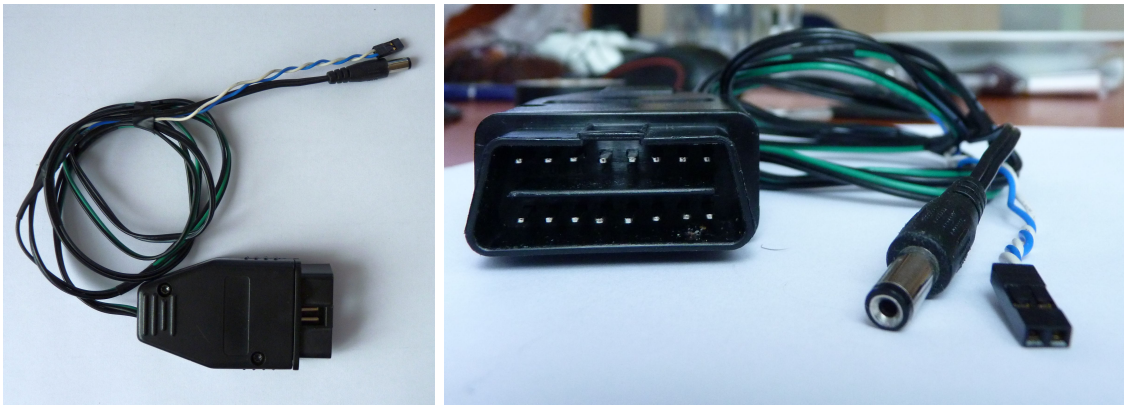


Obrázek 3.3: Výuková platforma FITkit použitá ako medzičlánok pri komunikácii telefónu s OS Android s riadiacou jednotkou automobilu Škoda Felícia

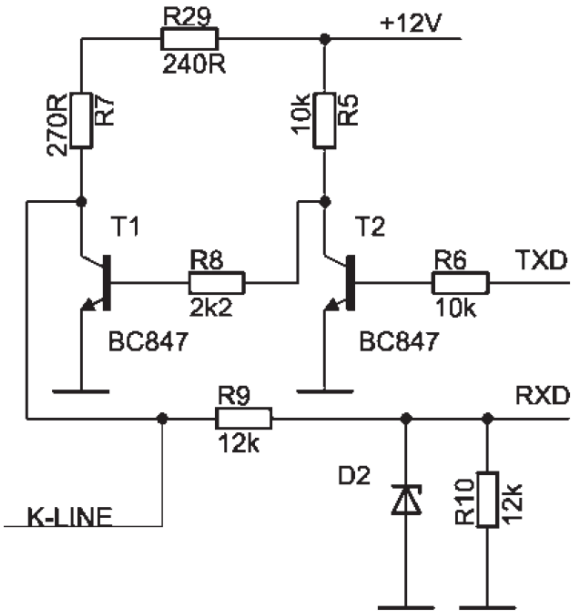
Platforma FITkit umožňuje obsiahnuť značnú časť spektra znalostí a schopností, ktoré musí dnešný inžinier – informatik poznať, aby bol schopný obstáť na globálnom trhu práce. Typickým príkladom využitia informatiky v praxi sú tzv. vstavané systémy (anglicky Embedded Systems), ktoré sa v dnešnej dobe dominantne uplatňujú v bežnom živote a ich význam ešte výrazne porastie. Jednoducho povedané, jedná sa o všetky zariadenia, ktoré v sebe majú nejakým spôsobom vstavaný počítač (automobil, mobilný telefón, MP3 prehrávač, televízor,...). Typické vstavané systémy sa skladajú z procesorov, špecializovaného hardwaru (napr. MP3 kodér/dekodér) a aplikačného softwaru. To znamená, že je treba vedieť a prakticky využívať znalosti nielen z oblasti čisto softwarových odborov, ale tiež z oblasti hardwaru.

3.3.2 Napájací a dátový kábel

Keďže FITkit nemá USB rozhranie, musel som použiť vlastný kábel na komunikáciu s riadiacou jednotkou. Tento kábel slúži tiež na napájanie FITkitu z palubného napätia automobilu. Do konektoru OBD2 som zabudoval prevodník UART – K-Line. Na prevod napätia z 12V(palubné napätie auta) na 5V (napájacie napätie FITkitu) je použitý jednoduchý regulátor napätia založený na integrovanom obvode L7805CV.



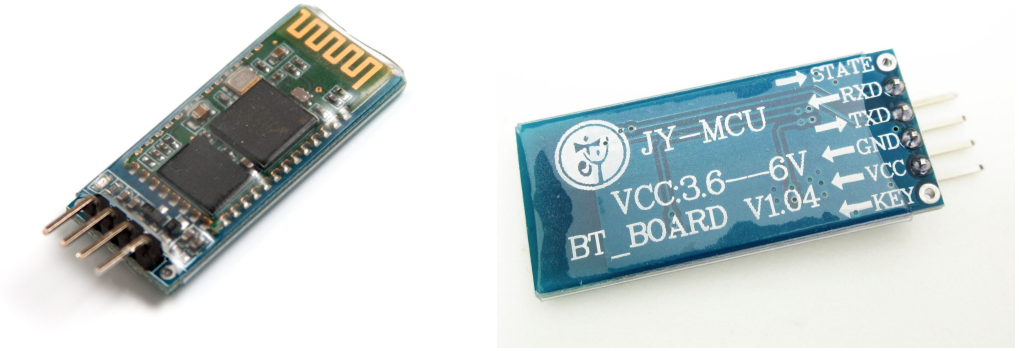
Obrázek 3.4: OBD2 kábel s integrovaným prevodníkom a regulátorom napätia vytvorený pre potreby napájania FITkitu a komunikácie FITkitu s riadiacou jednotkou automobilu.



Obrázek 3.5: Schéma zapojenia prevodníku UART – K-line, ktorý som integroval do konektoru OBD2. Tento prevodník je použitý na komunikáciu medzi FITkitom a riadiacou jednotkou. Toto zapojenie je prevzaté z časopisu Amatérské rádio - Praktická elektronika [9]

3.4 Komunikácia telefónu s FITkitom

Na komunikáciu FITkitu s telefónom je použitá bezdrôtová technológia Bluetooth. Túto technológiu som si vybral, pretože je veľmi rozšírená medzi mobilnými telefónmi a takmer každé zariadenie so systémom Android je schopné komunikovať prostredníctvom tejto technológie. FITkit však nepodporuje túto technológiu a tak som použil Serial Bluetooth Modul.



Obrázek 3.6: Serial Bluetooth modul použitý na bezdrôtovú komunikáciu medzi FITkitom a mobilným telefónom s OS Android.

K FITkitu je pripojený cez sériové rozhranie UART1 a napájaný je priamo z FITkitu. Tento modul sa dá kúpiť zo zahraničia za cenu približne 200Kč vrátane poštovného.

3.4.1 Komunikačný protokol

Pre potreby sériovej komunikácie medzi FITkitom a telefónom som navrhol jednoduchý komunikačný protokol. Protokol je zložený zo synchronizačného bajtu 0x10, ktorým vždy začína paket. Po ňom nasledujú hodnoty získané z riadiacej jednotky. Tieto hodnoty sú v takom istom tvare, v akom boli prijaté z riadiacej jednotky - každú hodnotu reprezentuje jeden bajt. Poradie, v akom sa hodnoty posielajú je pevne dané a to nasledovne:

1. 0x10 - synchronizačný bajt, podľa neho sa určí, že začína nový paket
2. dĺžka vstreku
3. otáčky motora
4. teplota motora
5. napätie na autobatérii

3.5 Vlastný výpočet

Z riadiacej jednotky motora je možné získať hodnoty: dĺžka vstreku v ms, počet otáčok za hodinu a informáciu o tom, či je motor v režimu akcelerácie alebo decelerácie - brzdenia motorom. V režime decelerácie motor nespotrebúva žiadne palivo. Z GPS v telefóne získame údaj o aktuálnej rýchlosti vozidla v km/hod. K vypočítaniu aktuálnej spotreby vozidla v l/100km však potrebujeme ešte vedieť, koľko paliva pretečie vstrekovačmi do motora za jednotku času v l/s. Táto hodnota je konštantná pre každé vozidlo a je určená konštrukciou vstrekovačov. Meraním som zistil, že v mojom aute je táto konštanta rovná 0.0062.

$$S = (O \times 60) \times \frac{D}{1000} \times \vartheta \times \frac{100}{V}$$

S je spotreba v litroch za hodinu,
O je počet otáčok za minútu (je potrebné previezť na ot/hodinu, preto je táto hodnota násobená 60),
D je dĺžka vstreku v milisekundách,
ϑ je konštanta určujúca objem paliva, ktorý pretečie vstrekovačmi za jednotku času v litroch za sekundu.
V je potom rýchlosť automobilu. Čím je vyššia rýchlosť, tým je nižšia spotreba (ak by sa ostatné hodnoty nemenili). Preto je rýchlosť v menovateli. Hodnota 100 je v čitateli prítomná preto, lebo štandardne sa udáva spotreba automobilu v litroch na 100km – bez nej by nám vyšla spotreba v litroch na kilometer.

Kapitola 4

Návrh užívateľského rozhrania aplikácie pre telefón

Užívateľské rozhranie sa dá považovať za jednu z najdôležitejších častí aplikácie, pretože práve s ním prichádza užívateľ do styku. Užívateľské rozhranie by malo byť jednoduché a pohodlné, aby aplikácia slúžila užívateľovi a nie naopak. V prípade aplikácie, ktorá sa bude používať v osobnom automobile to platí dvojnásobne. Aplikácia by mala zobrazovať informácie, ktoré užívateľa zaujímajú, no nepožadovať od neho takmer žiadne úkony, aby sa mohol čo najviac venovať šoférovaniu. Mala by mať teda iba akýsi informatívny charakter.

Preto som sa rozhodol rozdeliť užívateľské rozhranie na tri obrazovky, medzi ktorými sa môže užívateľ pohybovať veľmi jednoduchým pohybom jediného prstu - potiahnutím. V nasledujúcich sekciách predstavím jednotlivé obrazovky a metódu prepínanie medzi nimi. Základnou vlastnosťou užívateľského rozhrania je použiteľnosť počas jazdy v aute.

4.1 Číselné zobrazenie aktuálnej a priemernej spotreby

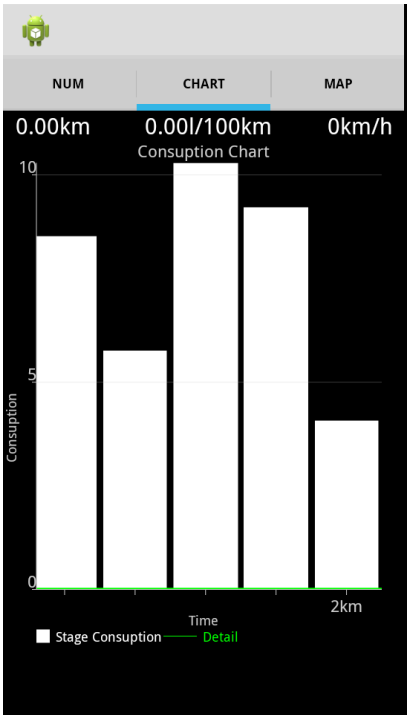
Po spustení aplikácie a kliknutí na položku v Menu – Start a pripojení k GPS a riadiacej jednotke sa užívateľovi poskytne pohľad na aktuálnu a priemernú spotrebu jeho automobilu. Táto obrazovka bude pravdepodobne aj najpoužívanejšou, pretože práve tieto dva údaje najviac zaujímajú užívateľov.

Spočiatku boli tieto dva údaje zobrazované pomocou dvoch analógovo vyzerajúcich ukazateľov s ručičkou ukazujúcou požadovaný údaj. Myslel som si, že takto lepšie zapadne medzi analógové ukazatele, ktoré sú súčasťou palubnej dosky automobilu. No po skúšobnej jazde (približne 100km) som zistil, že sa musím veľmi sústrediť na to, aby som vedel odčítať hodnotu z takéhoto zobrazenia a odvádzalo to moju pozornosť od šoférovania. Preto som sa rozhodol, že toto zobrazenie zmením na číselné a zvolil som písmo v retro štýle, ktoré je veľmi dobre čitateľné a súčasne pekne zapadá do obdobia, kedy bolo vozidlo vyrobené.

Obrázek 4.1: Vľavo je pohľad ihneď po spustení, kedy sa aplikácia pripája k GPS a riadiacej jednotke auta. Vpravo už je aplikácia pripojená a zobrazujú sa údaje o spotrebe.

4.2 Zobrazenie spotreby v grafe

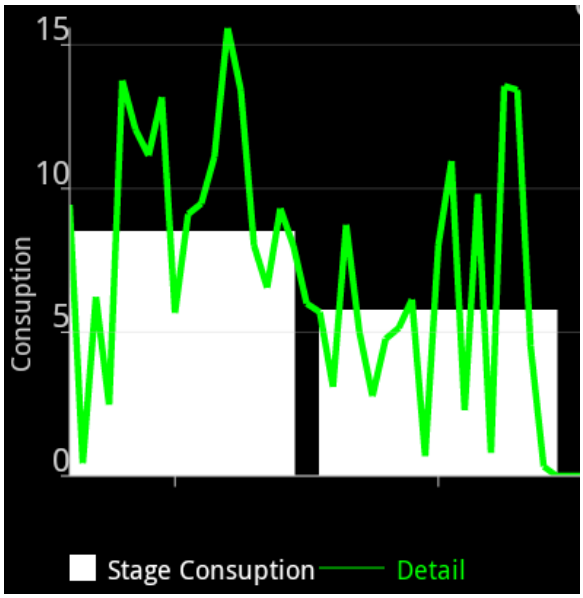
Okrem aktuálnych údajov z prvej obrazovky by mohlo užívateľa zaujímať aj to, ako sa menila jeho spotreba s prejdenou vzdialenosťou. Takáto situácia môže nastať napríklad pri jazde do tiahleho kopca a potom zasa dolu tiahlym kopcom, kedy je zaujímavé vidieť rozdiel medzi spotrebou pri jazde do kopca a z kopca. Údaj o spotrebe je zobrazovaný v stĺpcovom grafe, kde jeden stĺpec reprezentuje prejdený úsek 2 kilometrov. Na osi X je zobrazovaná prejdená vzdialenosť a na ose Y spotreba.



Obrázek 4.2: Zobrazenie spotreby na stĺpcovom grafe, kde jeden stĺpec reprezentuje dvojkilometrový úsek.

4.2.1 Detail úseku

Zaujímavou informáciou je aj to, aký bol priebeh spotreby v rámci jedného stĺpca – úseku 2 km. Túto informáciu poskytuje čiarový graf, ktorý sa kreslí počas jazdy a je veľmi zaujímavé pozorovať, ako sa mení spotreba počas úseku.



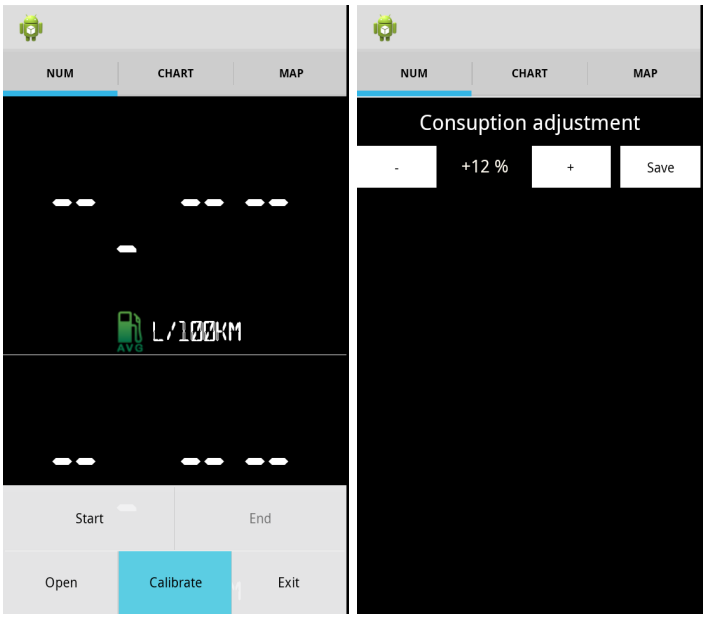
Obrázek 4.3: Pre každý dvojkilometrový úsek sa vykresľuje v reálnom čase aj jeho detail - čiarový graf, zobrazujúci spotrebu počas týchto dvoch kilometrov.

4.2.2 Porovnanie s históriou

Ak vodič prejde tým istým úsekom viac krát, aplikácia to rozpozná a zobrazí vodičovi, či mal na tomto úseku lepšiu alebo horšiu spotrebu ako je priemer na tomto úseku.

4.2.3 Úprava konštanty ϑ

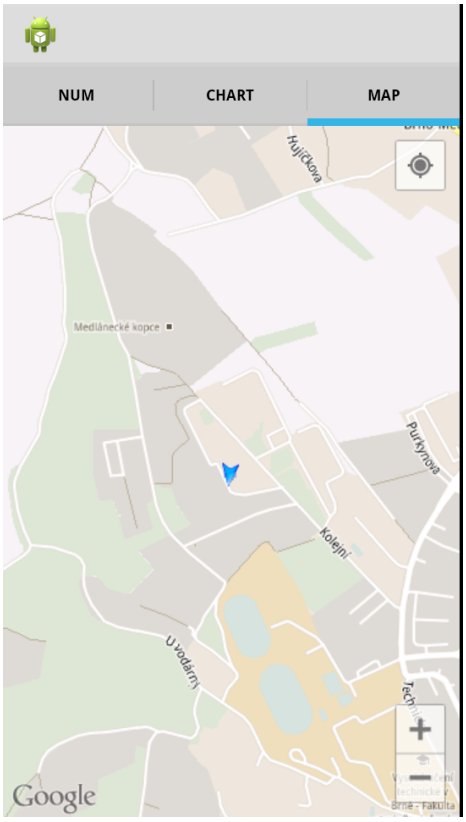
Aby bolo možné prispôsobiť vypočítanú spotrebu tak, aby sa čo najviac približovala k reálnej hodnote (spočítanej napríklad pri tankovaní), má užívateľ možnosť po stlačení Menu → Calibrate pridať, alebo ubrať z hodnoty ϑ, ktorá sa používa pri výpočte spotreby. Po stlačení tlačidla Save sa nová hodnota uloží, a od toho momentu sa počíta spotreba s novou hodnotou ϑ.



Obrázek 4.4: Na obrázkoch je vidieť užívateľské rozhranie pre úpravu hodnoty ϑ, ktorá sa používa pri výpočte spotreby

4.3 Zobrazenie na mape

Na poslednej obrazovke je zobrazená mapa, na nej aktuálna poloha a smer jazdy vozidla. Táto funkcia je vhodná pre rýchle zorientovanie sa pri jazde. Pri používaní tejto funkcie je potrebné mať zapnutý internet v mobile alebo mať dopredu uložené mapy v aplikácii Maps.



Obrázek 4.5: Mapa zobrazujúca aktuálnu polohu a smer jazdy. Táto karta umožňuje vodičovi rýchle zorientovanie sa v priestore v prípade, že sa stratil a nevie kam má ísť.

4.4 Tipy pre ekonomickú jazdu

Aplikácia navyše kontroluje, či vodič zbytočne nemíňa palivo, keď to nieje treba. Užívateľ je upozornený, že zbytočne míňa palivo pri týchto situáciách:

1. Jazda dolu kopcom: Pri dlhšej jazde dolu dlhým kopcom niektorí vodiči zaradia neutrál a tým zbytočne plytvajú palivom, ktoré sa spaľuje pri behu motora na voľnobeh. Preto by mal vodič vždy pri jazde dolu dlhým kopcom mať zaradenú rýchlosť. Na toto je upozorňovaný vyskakovacím oknom s upozornením.

2. Vypnutie motora pri dlhšom státí: Pri dlhšom státí nie je vhodné nechávať zapnutý motor a tým zbytočne míňať palivo. Na toto je užívateľ upozorňovaný tak isto vyskakovacím oknom.

Kapitola 5

Implementácia programu pre FITkit

Na prípravku FITkit sa nachádzajú dva programovateľné obvody - mikrokontrolér `MSP430F2617` a obvod `FPGA XC3S50`. Ja som sa rozhodol použiť mikrokontrolér `MSP430F2617`, pretože mám s ním väčšie skúsenosti a je vhodnejší pre potreby tohoto projektu. Program pre mikrokontrolér bol napísaný v jazyku C a skompilovaný prekladačom `MSPGCC`. K programovaniu mikrokontroléru bol použitý program QDevKit.

Celý program pre FITkit sa skladá z troch hlavných častí:

1. Zobudenie riadiacej jednotky
2. Komunikácia medzi FITkitom a riadiacou jednotkou
3. Komunikácia medzi FITkitom a telefónom prostredníctvom bluetooth modulu

Aby bolo možné implementovať tieto časti, musel som upraviť knižnicu **libfitkit**, ktorá sa bežne používa pri programovaní aplikácii pre FITkit.

5.1 Úpravy knižnice libfitkit

Knižnicu **libfitkit** bolo potrebné upraviť z viacerých dôvodov. Prvým problémom, na ktorý som pri implementácii narazil, bolo zobudenie riadiacej jednotky extrémne pomalou rýchlosťou prenosu `5 Baud`. Najnižšia rýchlosť, ktorú bolo možné nastaviť použitím knižnice **libfitkit** bolo `1200 Baud`. Preto som musel upraviť súbory `uart.c` a `uart.h`, a v nich vytvoriť novú funkciu, ktorá inicializuje sériový port na rýchlosť `5 Baud`. Toto som dosiahol tak, že som použil iný zdroj hodinového signálu, ako používa **libfitkit**, a síce `UCSSEL_1` s frekvenciou `32.768kHz` namiesto pôvodného `UCSSEL_2` s frekvenciou `7.3728MHz`. Rýchlosť prenosu sa totiž určuje ako zdroj hodinového signálu vydelený deličkou zloženou z registrov `USCA0BR0` a `USCA0BR1`. Ak by som ponechal pôvodný zdroj hodinového signálu, nebolo by možné komunikovať rýchlosťou `5 Baud`, ale najmenej 70 Baud - pri maximálnej možnej hodnote deličky `0xFFFF`. Preto som použil zdroj hodinového signálu `UCSSEL_1` a hodnotu deličky `0x199A` a tým som získal požadovanú rýchlosť `5 Baud` [6].

Túto úpravu som robil počas vianočných sviatkov a bolo veľmi obtiažne testovať, či je rýchlosť prenosu naozaj taká, ako chcem, pretože som nemal doma osciloskop. Musel som

improvizovať a tak som meral rýchlosť prenosu tak, že som posielal rôzne hodnoty na sériový port, na ktorý som pripojil LED diódu a so stopkami v ruke som meral ako bliká. Keď sa zdalo, že ide všetko ako má, zobral som FITkit do auta a overil som funkčnosť.

Ďalším krokom bolo implementovať komunikáciu podľa protokolu KWP-1281. To znamenalo, že ihneď po prebudení riadiacej jednotky som musel zavrieť sériový port a znova ho nainicializovať, no teraz už s klasickou rýchlosťou `9600 Baud`. Ďalej som postupoval presne podľa popisu protokolu KWP-1281 – vytvoril som funkcie pre prečítanie úvodnej frázy, ktorú posiela riadiaca jednotka a funkcie pre vyžiadanie a prečítanie skupiny.

Keď už dokáže FITkit získať dáta z riadiacej jednotky, zostáva implementovať poslednú časť – odosielanie týchto dát cez bluetooth modul do telefónu. A opäť bolo potrebné upravovať knižnicu **libfitkit**. Tá totiž obsahuje funkcie pre obsluhu iba jedného sériového portu a ja predsa potrebujem komunikovať po oboch naraz. Preto som musel vytvoriť nové funkcie pre inicializáciu a odosielanie dát cez druhý sériový port.

Kapitola 6

Implementácia aplikácie pre telefón

Implementácia aplikácie pre telefón s OS Android nebola nakoniec až taká náročná, ako sa spočiatku zdalo, pretože veľká časť projektu - implementácia komunikácia podľa KWP-1281 bola presunutá na FITkit. Bolo teda ešte potrebné naprogramovať komunikáciu s FITkitom, získavanie údajov o polohe a rýchlosti z GPS, intuitívne zobrazenie hodnôt, ktoré sú prijímané z FITkitu a ich uloženie a následné znovu-zobrazenie.

Jadrom celej aplikácie je trieda `MainActivity`, ktorá implementuje hlavnú aktivitu s rovnakým názvom. Z tejto aktivity sa potom vytvárajú ďalšie potrebné objekty. Hlavným ovládacím prvkom aplikácie je `ActionBarSherlock` a menu, ktoré sa vyvolá po stlačení tlačidla Menu. Každá z troch kariet je implementovaná ako fragment:

Karta Num je implementovaná v triede `FragmentOne`, `RouteListFragment` a `CalibrateFragment`. Posledné dve sú použité na výber údajov a úpravu konštanty určujúcej spotrebu.

Karta Chart je implementovaná v triede `FragmentTwo` – odtiaľ sa ďalej využíva trieda `Graph` implementujúca vykresľovanie grafov.

Karta Map – je implementovaná v triede `FragmentThree` – toto však na rozdiel od predchádzajúcich nieje obyčajný fragment, ale SupportMapFragment – špeciálny fragment na vykresľovanie máp

6.1 Android SDK

Android SDK je paleta nástrojov určených pre vývojárov aplikácií na operačný systém Android. Jej obsahom sú nástroje na ladenie aplikácie (debugger), AVD manažér - správca virtuálnych zariadení, na ktorých si môže vývojár vyskúšať, ako funguje jeho aplikácia, dokumentáciu, ukážky kódov a tutoriály. Oficiálnym vývojovým prostredím je Eclipse s použitím pluginu Android Development Tools - ADT. V tomto prostredí som aj ja vyvíjal túto aplikáciu [7].

6.1.1 Google Maps Android API v2

Knižnicu Google Maps Android API v2 [2] som použil na zobrazovanie aktuálnej polohy automobilu na mape. Aby bolo možné túto knižnicu použiť, bolo potrebné zaregistrovať aplikáciu a získať API kľúč, ktorý je potrebné vložiť do Manifestu aplikácie.

6.2 Zistenie polohy a rýchlosti pomocou GPS

Pre potreby získavania údajov o rýchlosti a polohe som vytvoril triedu `MyLocationListener`, ktorá implementuje triedu LocationListener z balíčku android.location. Pri každej zmene polohy sa vyvolá metóda `onLocationChanged`, v ktorej sa aktualizujú premenné určujúce rýchlosti (`speed, avgSpeed, totalAvgSpeed`), počítadlá `i` (pre potreby výpočtu priemernej rýchlosti v úseku, nuluje sa pri každom novom úseku), `totalI` (pre potreby výpočtu priemernej rýchlosti za celú jazdu) a premenné, v ktorých je uložená nadmorská výška a poloha GPS - `alt, lat, lng`. Pridal som metódy:

getActSpeed() – vráti aktuálnu rýchlosť v km/hod

getAvgSpeed() – vráti priemernú rýchlosť v rámci úseku

resetAvgSpeed() – vynuluje priemernú rýchlosť (používa sa pri začatí nového úseku)

getTotalAvgSpeed() – vráti priemernú rýchlosť za celú cestu

6.3 Komunikácia cez Bluetooth

Komunikácia telefónu s FITkitom je implementovaná v triede `BT`, kde sa v koštruktore vyhľadá bluetooth modul pripojený k FITkitu metódou `findBT()` a prebehne pripojenie k tomuto zariadeniu v metóde `openBT()`. Pri vytváraní týchto dvoch metód som sa inšpiroval vzorovým projektom **BluetoothChat**, ktorý je súčasťou balíčku **Android SDK**.

Jadrom celej tejto triedy je metóda `beginListenForData()` v ktorej sa vytvára vlákno `workerThread`, ktorá prijíma dáta po bajtoch poslané FITkitom a roztrieďuje ich. Protokol je navrhnutý tak, že dáta sú posielané stále dookola a synchronizujú podľa oddeľovača 0x10. Po ňom nasledujú 4 bajty určujúce postupne: dĺžku vstreku, otáčky, teplotu motora a napätie na autobatérii. Z týchto hodnôt sa potom spočítajú údaje o spotrebe.

Ďalej sú implementované nasledujúce metódy:

getActConsuptLH() – vráti aktuálnu spotrebu v litroch za hodinu

getAvgConsuptLH() – vráti priemernú spotrebu v litroch za hodinu v rámci úseku

resetAvgConsuptLH() – vynuluje priemernú spotrebu v litroch za hodinu (používa sa pri začatí nového úseku)

getTotalAvgConsuptLH() – vráti priemernú spotrebu v litroch za hodinu za celú cestu

6.4 ActionBarShelock

Aby som si uľahčil implementáciu pohybu medzi oknami potiahnutím prstom, použil som knižnicu **ActionBarSherlock**. Pomocou nej bolo možné vytvoriť okná a pohybovať sa medzi nimi podľa návrhu. **ActionBarSherlock** je rozšírením oficiálnej podpornej knižnice **SupportActionBar**. Ja som implementroval triedu `TabSwipeActivity`, ktorú som vytvoril podľa príkladov, ktoré zverejnili autori tejto knižnice [8].

6.5 AChartEngine

Na zobrazovanie údajov o spotrebe na jednotlivých úsekoch vo forme grafu som použil knižnicu **AChartEngine** [3]. Za pomoci tejto knižnice trieda `Graph` implementuje ako stĺpcový, tak aj čiarový graf. Zobrazovanie údajov o spotrebe na grafe prebieha v dvoch prípadoch:

1. Počas jazdy a zaznamenávania sa zobrazujú dáta v reálnom čase - na stĺpcovom grafe dvojkilometrové úseky, a na čiarovom grafe detaily týchto úsekov.

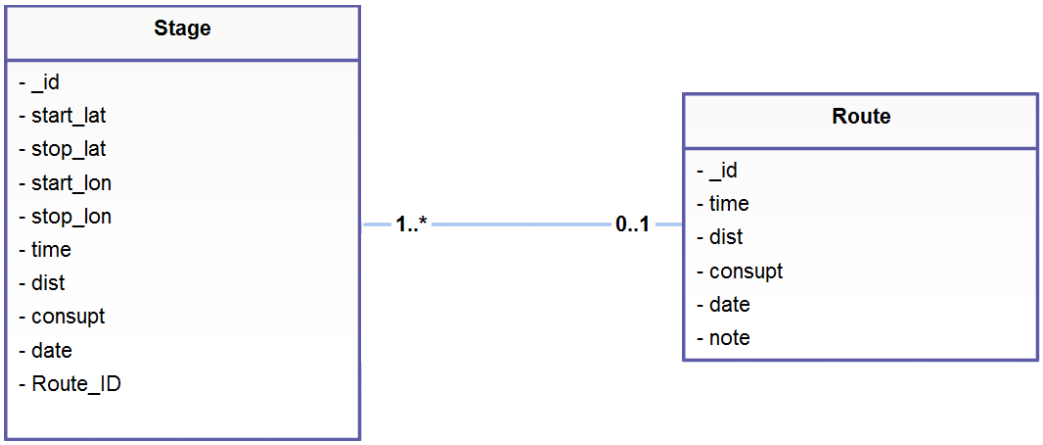
2. Počas prezerania už zaznamenaných dát sa na stĺpcovom grafe zobrazujú zaznamenané dvojkilometrové úseky. Na na čiarovom grafe sa v tomto móde ale zobrazuje postupne sa vyvíjajúca priemerná spotreba.

6.6 Ukladanie údajov

Ako dátové úložisko som použil databázový systém SQLite. SQLite je softwarová knižnica, ktorá implementuje sebestačný, transakčný SQL databázový engine. SQLite je najrozšírenejší SQL databázový engine na svete. Pre prácu s týmto enginom som vytvoril triedy:

MyDatabaseHelper – v tejto triede je definovaná databáza a tabuľky

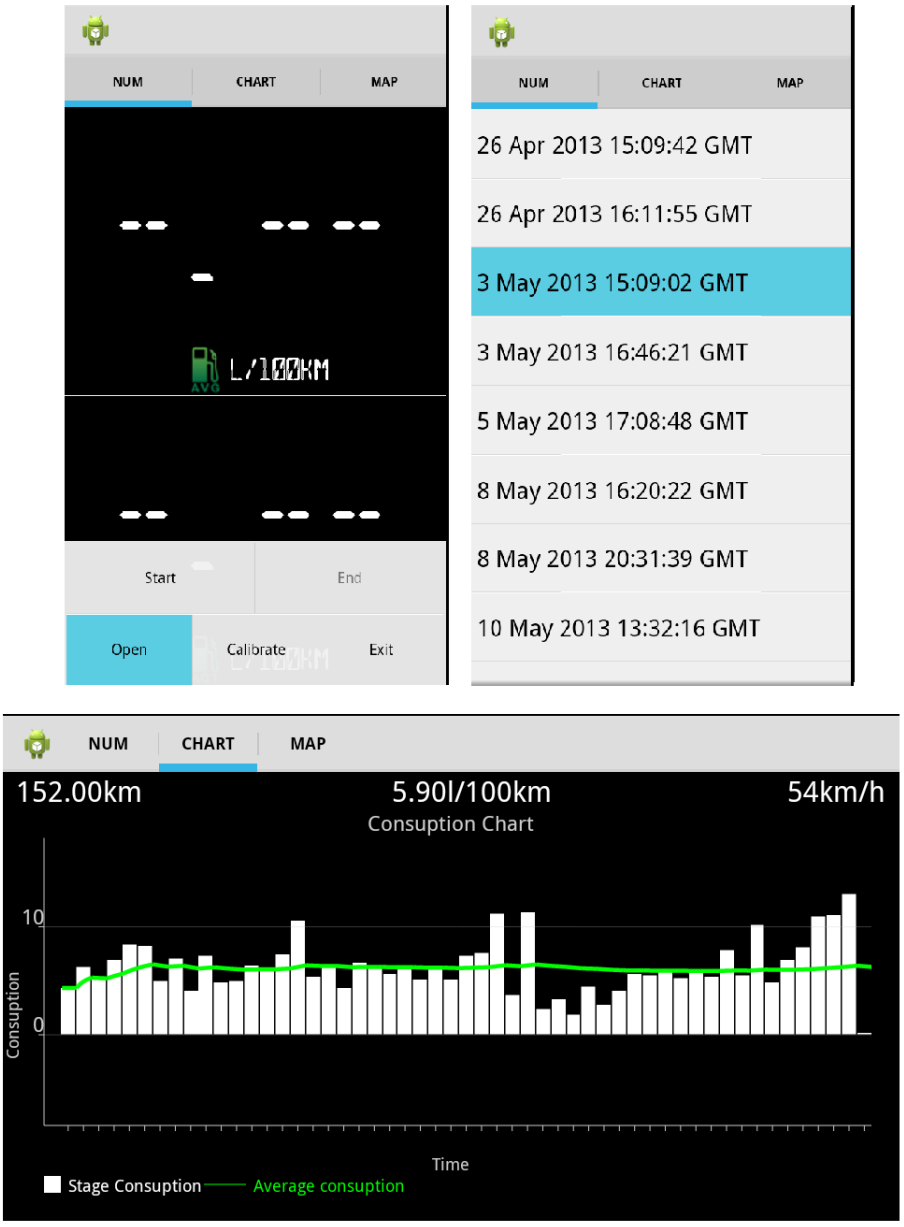
MyDB – táto trieda obsahuje metódy umožňujúce vkladanie, úpravu a vyhľadanie v tabuľkách `Route`, `Stage` a `ConsuptConstant`, pričom v tabuľke `Route` sú uložené jednotlivé cesty, v tabuľke `Stage` sú uložené úseky o dĺžke 2 km a v tabuľke `ConsuptConstant` je uložená konštanta ϑ potrebná pre výpočet spotreby. V tejto triede je tiež implementované rozpoznanie znovu prechádzaného úseku podľa súradníc GPS, a to v metóde `findSameStage`.



Obrázek 6.1: Diagram zobrazujúci jednoduchú databázu vytvorenú na ukladanie údajov o rýchlosti, spotrebe a polohe vozidla na jednotlivých úsekoch aj celej ceste.

6.7 Prezeranie zaznamenaných údajov

Aby bolo možné prezerať zaznamenané údaje, je nutné si najprv vybrať, ktorú jazdu chceme prezerať. Tento výber je imlementovaný ako samostatný fragment v triede `RouteListFragment`. Po stlačení menu-tlačidla **Open** sa vyvolá zoznam, v ktorom je uvedený dátum a čas začatia každej jazdy. Po kliknutí na požadovanú položku zoznamu sa načíta táto jazda do karty `Num` a `Chart`. V móde prezerania údajov je zakázané prechádzanie medzi kartami gestom potiahnutia prstom, aby bolo možné sa voľne pohybovať po grafe, posúvať ho, približovať, alebo vzďaľovať.



Obrázek 6.2: Na týchto troch obrázkoch je postupne vidieť menu, výber požadovanej jazdy zo zoznamu jázd a následné prezeranie údajov zaznamenaných počas tejto jazdy

Kapitola 7

Testovanie

Testovanie systému spočívalo v tom, že som so systémom jazdil dlhé úseky a na nich som pozoroval správnosť údajov zobrazovaných mojim systémom v porovnaní s palubnými ukazateľmi a množstvom natankovaného paliva. Urobil som 5 meraní na vzdialenosti vždy okolo 500 km. Aby som čo najviac zmiernil nepresnosti v množstve natankovaného paliva na benzínovej pumpe, tankoval som vždy plnú nádrž na tej istej benzínovej pumpe a z toho istého stojanu. V nasledujúcej tabuľke je vidieť detail tankovania a rozdiely oproti môjmu systému.

Číslo merania	1	2	3	4	5
Vzdialenosť v km	551	479	590	427	578
Množstvo paliva v litroch	34,69	30,35	36,00	26,53	37,74
Reálna spotreba v l/100km	6,296	6,336	6,102	6,213	6,529
Spotreba podľa systému l/100km	6,312	6,382	6,065	6,239	6,455
Rozdiel v l/100km	+0,016	+0,046	-0,037	+0,026	-0,074
Rozdiel v %	+0,25	+0,72	-0,61	+0,42	-1,14

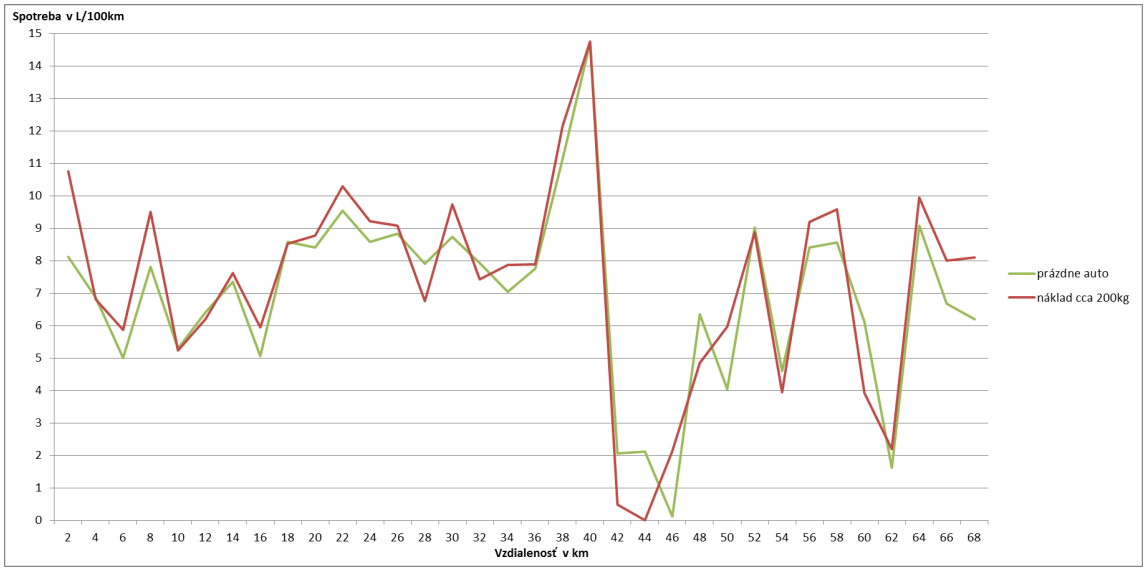
Z tohto merania vyplýva, že systém funguje správne a priemerná odchýlka od reálnej spotreby je 0,04 litra na 100 kilometrov. Takúto nízku odchýlku som dosiahol tým, že pred ostrým meraním som viac krát upravoval hodnotu ϑ, ktorá je konštantou určujúcou objem paliva, ktorý pretečie vstrekovačmi za jednotku času v litroch za sekundu. Takto som kalibroval môj systém, aby bolo meranie čo najpresnejšie.

K takým presným hodnotám určite prispel aj fakt, že som jazdil z veľkej časti stále rovnaké úseky. Rovnaké úseky som jazdil hlavne preto, lebo každý týždeň cestujem autom domov a späť na internát približne 150 kilometrov. To bola výborná príležitosť, ako dlhodobo testovať tento systém. Ak by som mal jazdiť toľko kilometrov iba za účelom testovania, stálo by ma to veľmi veľa peňazí za benzín.

Aplikácia je veľmi náročná na spotrebu elektrickej energie – výdrž na batériu je porovnateľná s výdržou pri zapnutej GPS navigácii. Je to spôsobené tým, že môj systém používa zabudované GPS a displej telefónu je stále rozsvietený. Navyše aplikácia v telefóne neustále používa Bluetooth na komunikáciu s FITkitom, čo je ďalší dôvod vysokej spotreby energie. Je teda vhodné mať pri používaní tohto systému telefón zapojený do nabíjačky. Telefón

na ktorom som testoval môj systém (Motorola Droid 3) vydržal bez pripojenia nabíjačky približne 1 hodinu a 30 minút. Pri použití nabíjačky je možné systém používať nepretržite.

Ďalej som testoval, aký je rozdiel spotreby medzi plne naloženým autom a tým, keď ide autom iba šofér. Meranie prebiehalo na 70 km dlhom kopcovitom teréne medzi Piešťanmi a Uherským Hradišťom. Na grafe je vidieť tieto dve jazdy. Na osi X je nanesená vzdialenosť, na osi Y spotreba v l/100km.



Obrázek 7.1: Graf zobrazujúci rozdiel medzi spotrebou automobilu s nákladom a bez nákladu pri prejazde tým istým úsekom.

Z vyššie uvedeného grafu vyplýva, že spotreba automobilu, ktorú som meral mojim systémom je pri naloženom aute vyššia ako jazde bez nákladu. Je všeobecne známe, že každých 100 kilogramov nákladu v aute navyše spôsobí nárast spotreby automobilu približne o 0,1 litru na 100 kilometrov. Pri meraní pomocou môjho systému som zaznamenal priemernú spotrebu na vyššie zobrazenom úseku pri naloženom aute (200kg) 7,28 litra na 100 kilometrov. Pri jazde bez nákladu som zaznamenal spotrebu 6,94 litra na sto kilometrov. Je teda jasne vidieť rozdiel v spotrebe medzi naloženým a prázdnym autom 0,34 litra na 100 kilometrov. Tento výsledok jasne ukazuje, že systém pracuje správne a je dostatočne citlivý aj na zmenu v zaťažení automobilu.

Kapitola 8

Záver

V rámci svojej práce som naštudoval komunikačný protokol KWP-1281, naučil som sa pracovať s diagnostickými nástrojmi pre automobily a preskúmal som ponuku aplikácií na trhu, ktoré zastávajú funkciu palubného počítača. Navrhol a implementoval som systém pozostávajúci z diagnostického káblu, programu pre FITkit s Bluetooth modulom a aplikácie pre mobilný telefón s OS Android. Riešenie sa po dohode s vedúcim práce mierne odchýlilo od zadania, no máme za to, že toto odchýlenie viedlo k lepšiemu a použiteľnejšiemu výsledku. Myslím si, že sa mi podarilo vyvinúť systém, ktorý je skutočne použiteľný a užitočný a dokáže nahradiť originálny palubný počítač do Škody Felície. Je mi ľúto, že nieje možné rozšírenie tohoto systému medzi širokú verejnosť, pretože zahŕňa FITkit, no je možné vytvoriť jednoduché zariadenie s mikroprocesorom, ktoré by zastávalo túto úlohu. Ja sám však tento systém používam pri každej dlhšej jazde a som s ním ako vodič veľmi spokojný.

Ďalším pokračovaním tohoto projektu bude rozšírenie podpory o modernejší protokol KWP-2000. Pri použití tohoto protokolu by odpadala nutnosť použitia FITkitu (alebo iného zariadenia, ktoré by zobúdzalo riadiacu jednotku), pretože tento protokol je možné kompletne implementovať na OS Android. Bolo by tiež možné použiť Bluetooth OBD2 adaptér, ktorý je bežne dostupný za cenu do 300 Kč.

Nasledovne by som rád sprístupnil aplikáciu v obchode Google Play a myslím si, že by si našla veľa užívateľov.

Literatura

[1] Keyword Protocol 1281. 2008, j2818.

[2] Google Maps Android API v2. 2013.
URL https://developers.google.com/maps/documentation/android/

[3] 4ViewSoft: AChartEngine. 2009.
URL http://www.achartengine.org/index.html

[4] Hinner, M.: Diagnostika automobilových řídicích jednotek. 2006. Bakalářská práce. FEKT VUT Brno.

[5] ŠKODA, automobilová a.s.: Dílenská příručka FELICIA: Vstřikovací a zapalovací zařízení Simos 2P. 1996.

[6] Litovsky, G.: Beginning Microcontrollers with the MSP430 Tutorial. online.
URL http://www.glitovsky.com/Tutorialv0_3.pdf

[7] Murphy, M.: *Android 2*. Computer Press, 2011.

[8] Wharton, J.: ActionBarSherlock. 2012.
URL http://actionbarsherlock.com/index.html

[9] Zdeněk, S.: Palubní počítač PP-KWP1281. *Praktická elektronika A Radio*, 2010, ISSN 1211-328X.

Příloha A

Obsah CD

Na priloženom CD sa nachádzajú:

1. Zdrojové kódy programu pre FITkit
2. Zdrojové kódy aplikácie pre telefón s OS Android
3. Plagát pre prezentovanie projektu
4. Video pre prezentovanie projektu

Příloha B

Plagát