Modifikacija komunikacijskog protokola upravljačkih tipki multimedijalnog sustava osobnog automobila

Cerin, Petar

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski studij elektrotehnike

Završni rad

Modifikacija komunikacijskog protokola upravljačkih tipki multimedijalnog sustava osobnog automobila

Rijeka, rujan 2023.

Petar Cerin
0069084198

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski studij elektrotehnike

Završni rad

Modifikacija komunikacijskog protokola upravljačkih tipki multimedijalnog sustava osobnog automobila

Mentor: doc.dr.sc. Ivan Volarić

Rijeka, rujan 2023.

Petar Cerin
0069084198

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 20. ožujka 2023.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Digitalna elektronika**
Grana: **2.03.03 elektronika**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Petar Cerin (0069084198)**
Studij: Sveučilišni prijediplomski studij elektrotehnike

Zadatak: **Modifikacija komunikacijskog protokola upravljačkih tipki multimedijalnog sustava osobnog automobila / Communication protocol modification of the car multimedia system control buttons**

Opis zadatka:

U sklopu završnog rada potrebno je analizirati komunikacijski protokol radio prijamnika tvornički ugrađenog u osobni automobil. Ugrađeni radio potrebno je zamjeniti s radio prijamnikom koji koristi NEC komunikacijski protokol. U svrhu ispravne funkcionalnosti novog radio prijamnika, potrebno je dizajnirati sustav temeljen na mikrokontroleru koji će postojeći komunikacijski protokol pretvoriti u NEC protokol, čime će se omogućiti upravljanje radio prijamnikom pomoću postojećih multimedijalnih tipki osobnog automobila.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

PCerin

Zadatak uručen pristupniku: 20. ožujka 2023.

Mentor:

Doc. dr. sc. Ivan Volarić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Dubravko Franković

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____

Petar Cerin

Sadržaj

Popis slika

Popis tablica

Predgovor

Ovaj završni rad je rezultat inženjerske znatiželje, te potrebe za stvaranjem jeftinijeg rješenja pristupačnosti kontrole za upravljanjem novim (zamjenskim) radio uređajem u automobilu. Ovakve solucije na tržištu koštaju i do stotinjak eura, te ih je sve teže nabaviti radi manje potrebe za zamjenom originalnog radio uređaja zbog dostatnih funkcionalnosti kao što su dovoljno dobra kvaliteta reprodukcije, te povezivanje pomoću Bluetooth-a ili žičnog aux kabela na mobilni uređaj.

Na fakultetskom obrazovanju se potrebne teme za izradu integriranih rješenja prolaze odviše površno, jer su sveučilišni studiji namijenjeni pripremanju studenata na akademiju, što rezultira nedostatku spremnosti studenata na tržište rada u industriji. Izradom ovakvih projekata se studentima olakšava integriranje nakon obrazovanja i izradu portfolija, te pokazuje dodatnu želju za učenjem i stvaranjem za vrijeme obrazovanja, što pridonosi izgradnju mreža sa kolegama i profesorima. Zaključno, ovakvi projekti omogućuju primjenu teorijskih znanja i komponiranje u jedan cjeloviti oblik demonstracije svojih mogućnosti.

Poglavlje 1

Uvod

U ovom završnom radu je obrađena i objašnjena sva problematika vezana uz proces obrnutog inženjerstva (eng. *reverse engineering*) komunikacije tipkala na upravljaču automobila s OEM (eng. *original equipment manufacturer*) radiom u Renault Megane-u II, 2004. godišta, te adaptacija takve komunikacije na zamjenski eng. *aftermarket*, radio uređaj, u ovom slučaju model Kenwood KMM-123Y, te izrada integriranog rješenja. Analiziranjem ovakvog "black box" zatvorenog modela, nije potrebno njegovo pretvaranje u otvoreni model, nego samo izmjena izlaza iz tog sustava i prilagođavanje voljenim izlazima.

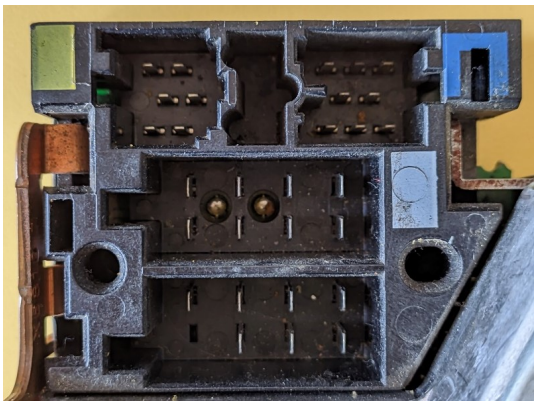
Poglavlje 2

Definiranje ožičenja

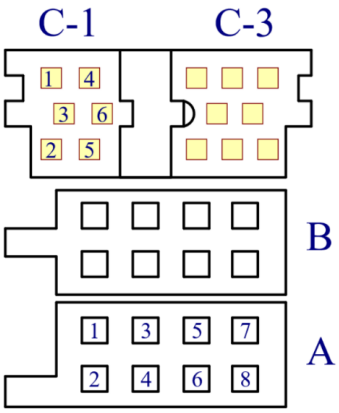
Tipkala za upravljanje multimedijskim sustavom nalaze se na desnoj strani iza upravljača automobila. Isprva, bilo bi logično da su komande spojene direktno na radio, te da ih radio prepoznaje pomoću ADC (eng. *analog to digital converter*), jer u većini slučajeva tipkala su sklopke koji prespajaju otpornike. S druge strane, konektor OEM radija koji je prikazan na slici 2.1 je podijeljen na tri dijela. "A" dio konektora služi za napajanje, te osnovne funkcije radija kao osvjetljenje, signal za brzinu vozila, itd. "B" dio je spojen na prednje i stražnje zvučnike, a "C" dio je opcionalan, te služi za dodatnu multimedijsku opremu kao što su ulaz zvuka i handsfree.

U ovom automobilu, postoji samo priključak C-1. Mjerenjem otpora multimetrom između bilo koje dvije žice, kao i žice i mase, ne donosi nikakve rezultate. Pronalazak mase na konektorima obavljen je multimetrom korištenjem zujala koje pišti kad je otpor manji od cca. 50-tak ohma, te spajanjem jedne testne sonde na šasiju automobila, a druge na svaki pojedini pin. Napon napajanja konektora A je također pronađen multimetrom. U tablici 2.1 su prikazani tako izmjereni rezultati, što na konektoru A odgovaraju nama potrebnim pinovima i standardu istog.

Napon na C-1-5 iznosi +12V samo kad je radio upaljen, što nam je kasnije važno. Pinovi 1,2 i 3 iznose 5V, te se napon ne mijenja kad su tipkala pritisnuta. Praćenjem žica ustanovljeno je da su one spojene na ekran u središnjoj konzoli, te od ekrana su također žice provučene do tipkala za upravljanje multimedijom. Kasnijim pronalaskom shema ožičenja, možemo potvrditi pretpostavku.[2]



(a) Fotografija konektora.



(b) Skica konektora.

Slika 2.1 ISO 10487 konektor.

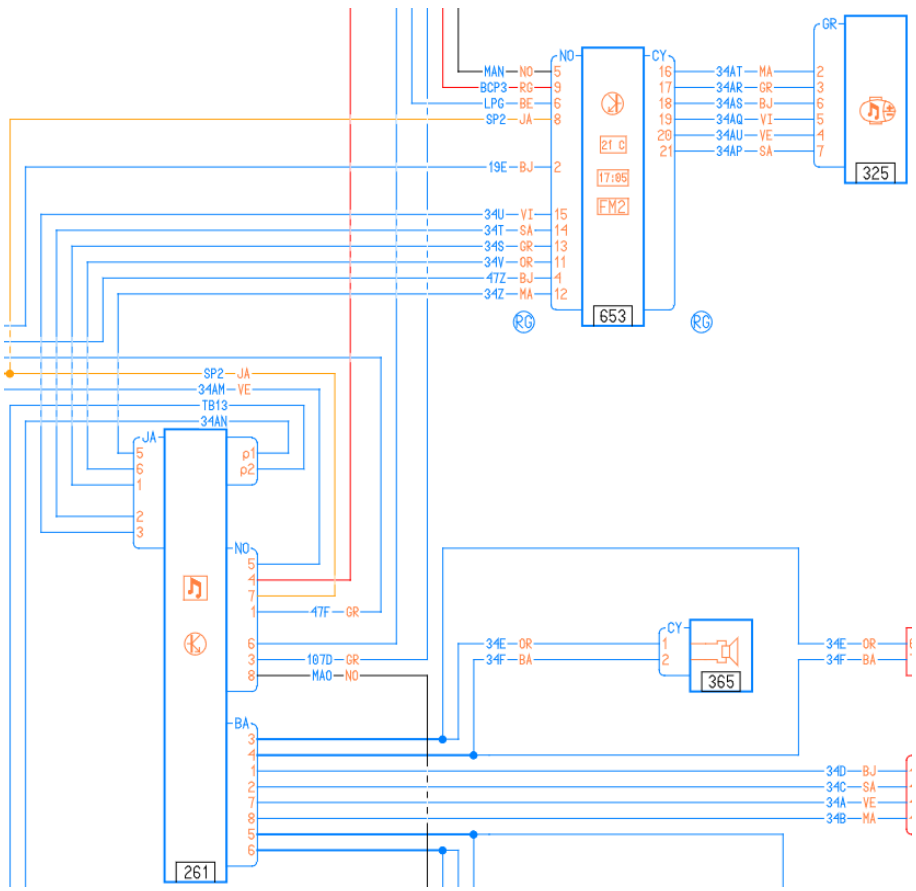
Tablica 2.1 Rezultati mjerenja multimetrom.

Konektor "A"		Konektor "C-1"	
4	+12V	1	+5V
5	Napajanje antene	2	+5V
7	+12V nakon kontakta	3	+5V
8	Masa	4	NC
		5	+12V
		6	Masa

Na slici 2.2 možemo vidjeti da je blok 261 radio uređaj, spojen direktno na ekran (blok 653), te su tipkala (blok 325) također spojeni na ekran. Standardna nomenklatura boja ove sheme prikazana je u tablici 2.2.

Tablica 2.2 Oznake boja žica.

BE	Plava	LG/GR	Siva	OR	Narančasta
JA	Žuta	BJ	Bež	VE	Zelena
MA	Smeđa	CY	Bijela		
RG	Crvena	VI	Ljubičasta		



Slika 2.2 Shema ožičenja ekrana.

Poglavlje 3

Analiza komunikacijskih protokola

Na slici 2.2, žice koje su nama bitne jesu one koje izlaze iz žutog konektora na radiju, te je njihova funkcija opisana u tablici 3.1

Tablica 3.1 Fukncije pinova.

1	34S	Siva	I2C RADIO DATA BUS SIGNAL
2	34T	Roza	I2C RADIO CLOCK SIGNAL
3	34U	Ljubičasta	I2C RADIO MARKING SIGNAL
5	34Z	Smeđa	RADIO OPERATING SIGNAL
6	34V	Narančasta	I2C RADIO EARTH SCREENING SIGNAL

Dakle iz navedenih podataka možemo zaključiti da radio komunicira sa ekranom pomoću I2C protokola. S druge strane, zamjenski radio ima jednu žicu koja je namijenjena spajanju na uređaj za kontroliranje multimedije. Pretraživanjem po internetu, može se naći da Kenwood koristi NEC protokol[1], koji se inače upotrebljava za prijenos podataka preko infracrvenih dioda u daljinskim upravljačima, no u ovakvom žičnom sustavu, koristi se 3.3V open-drain komunikacija bez modulacije noseće frekvencije od 38.222kHz.

3.1 Komunikacijski protokoli

Na Arduino platformi, komunikacijski protokoli su ključni za omogućavanje komunikacije između Arduino mikrokontrolera i drugih uređaja, senzora ili računalnih sustava. Postoje različite vrste protokola koji se koriste u različite svrhe, no njihove glavne razlike jesu brzina, maksimalna duljina sabirnice i broj uređaja spojenih na sabirnicu.

3.1.1 I2C protokol

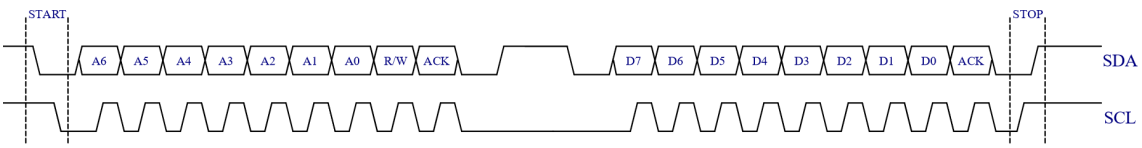
I2C komunikacijski protokol je serijski komunikacijski protokol za razmjenu podataka između različitih komponenti istog sustava razvijen od strane Phillipsa, te se najčešće koristi za komunikaciju između različitih integriranih čipova u jednom uređaju kao što su npr. tipkovnica na laptopu i IO (eng. *Input/Output*) kontroler. Za komunikaciju potrebna su dva vodiča u sabirnici, te uzemljenje.

1. SDA (eng. *serial data*) - Ovaj vodič prenosi podatke u sabirnici
2. SCL (eng. *serial clock*) - Ovaj vodič prenosi taktni signal
3. GND (eng. *ground*) - Uzemljenje

Glavna karakteristika sabirnice jest da ima samo jednog mastera, što je glavni uređaj koji može pozivati adrese ostalih uređaja (slave). U sabirnici nije dozvoljeno imati više mastera, radi nepodržavanja njihove arbitracije. Također, SDA i SCL linije su open-drain, što znači da u stanju mirovanja, iznose 5V, a prilikom slanja podataka one se spajaju na masu kako bi nastala logička nula. U stvarnosti ove vrijednosti mogu iznositi između 0V i 0.8V za nulu i 2V i 5V za jedinicu. Ograničenje brzine ovakve sabirnice određuje njen kapacitet, i vrijednost pullup otpornika. U svakoj I2C sabirnici, može biti priključeno 127 slave uređaja što odgovara $2^7 - 1$ adresa (moguće je korištenje i proširene 10-bitne adrese).

1. Na slici 3.1, START je označen isprekidanim linijama. Kako bi master probudio sve slaveove na sabirnici i započeo njihovo "slušanje", SDA mora postati 0 prije SCL linije.

2. Master kontrolira SDA liniju i za svaku nulu ili jedinicu koju pošalje generira impuls na SCL liniji, jer slave čita SDA na svakom rastućem bridu SCL linije. R/W bit govori slaveu da li želi pisati ili čitati sa slavea. Ako je slave s tom adresom na liniji, i može odgovoriti tj. nije zauzet nekim drugim zadatkom, on spušta ACK (eng. *acknowledged*) bit, te ga master čita. Ako je ACK dignut, drugim nazivom NACK (eng. *not acknowledged*), tj ako nitko ne odgovara, master čalje stop uvjet i linija se oslobađa.
3. Ako je slave prisutan i odgovori sa ACK, master oslobađa SDA liniju.
4. Kad slave preuzme kontrolu nad SDA linijom i spusti je, master počinje slati taktove na SCL liniji, a slave mu odgovara bit po bit do ACK-a, gdje se proces nastavlja bez slanja STOP uvjeta. Ako master odgovori s NACK, to znači da nema više čitanja, te se šalje stop uvjet na sabirnicu.



Slika 3.1 Slanje jednog bajta I2C protokola.

Brzina takta ovakve sabirnice je standardno 100kHz, ili 400kHz u brzom načinu rada. Postoje i brže konfiguracije uz korištenje push-pull umjesto open-drain kontrole.

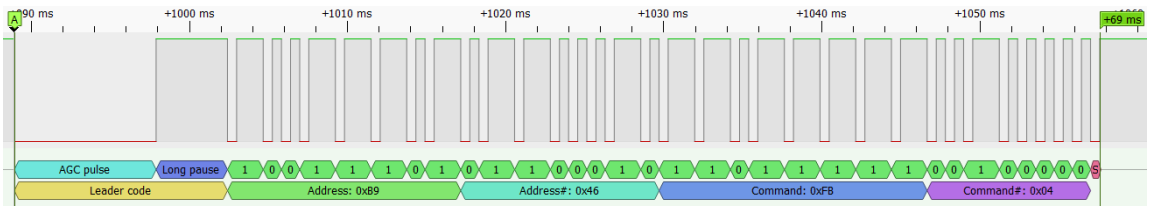
3.1.2 NEC protokol

U NEC protokolu, podatci se kodiraju kontroliranjem širine pulsa, s modulacijom noseće frekvencije od 38kHz. Logička nula odgovara visokom stanju sabirnice na 562.5μs, te niskom stanju jednakog vremenskog razdoblja. Logička jedinica odgovara visokom stanju sabirnice na 562.5μs, te niskom na 1.6875ms.[3] Takva se komunikacija koristi za slanje infracrvenog signala na daljinskim upravljačima. Na našoj sabirnici se koristi open-drain bez modulacije noseće frekvencije, stoga su naponske razine logičke nule i jedinice invertirane. Kako se ne bi mješali izrazi logičke nule

NEC protokola, i logičke nule stanja linije, koristit će se engleski izrazi HIGH za napon od 5V na liniji, te LOW za napon od 0V na liniji. Podatkovni paket NEC protokola sastoji se od :

1. Start uvjet (9ms LOW, i 4.5ms HIGH)
2. 8-bitna adresa uređaja s kojim komuniciramo
3. 8-bitni prvi komplement adrese uređaja s kojim komuniciramo
4. 8-bitni podatak
5. 8-bitni prvi komplement podatka
6. 562.5μs LOW puls kao stop uvjet

Svaki bajt se šalje u LSB redosljedu. Na slici 3.2 vidljiva je struktura tog paketa. Ukupno vrijeme trajanja poruke je 69ms.



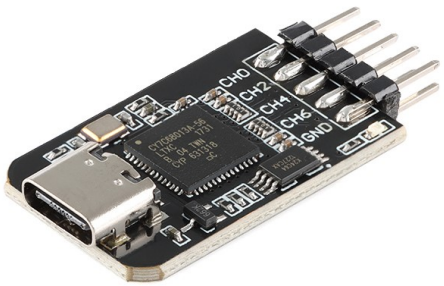
Slika 3.2 Slanje jednog bajta NEC protokola.

3.2 Prikupljanje podataka logičkim analizatorom

Logički analizator je instrument koji se koristi za analizu digitalnih signala. On omogućava inženjerima da detaljno pregledaju i dijagnosticiraju ponašanje digitalnih signala unutar sustava.

3.3 Hardverska podrška

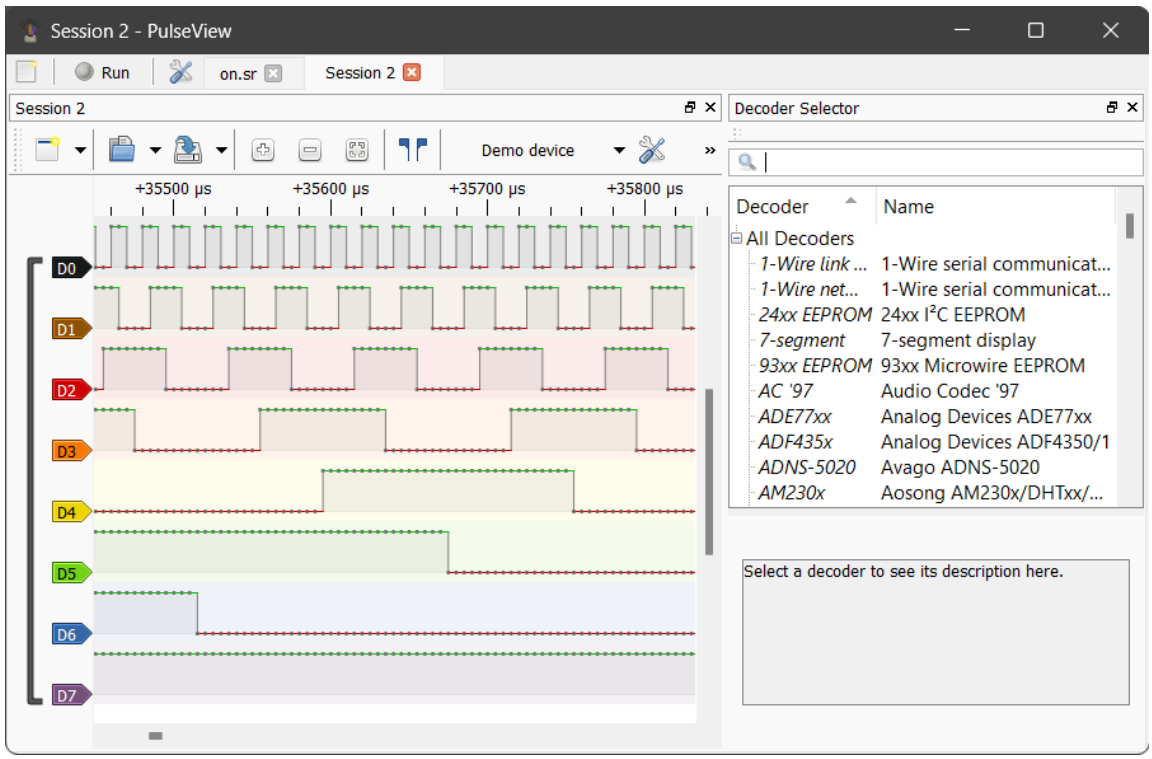
Za prikupljanje logičkih podataka, koristit ćemo logički analizer prikazan na slici 3.3 baziran na Infineon FX2 mikrokontroleru. Ovaj logički analizator ima 8 kanala, te podržava frekvenciju uzorkovanja do 24MHz. Frekvencija uzorkovanja digitalnih signala bi trebala biti barem 4 puta veća od frekvencije istog, no uzorkovanje većom frekvencijom nema nikakvih nedostataka, te smanjuje podrhtavanje (eng. *jitter*). Ovaj uređaj može primiti signale najvećeg mogućeg raspona napona od -0.5V do 5.25V, što je dovoljno za analizu komunikacije u automobilu, koja funkcionira od 0V do 5V. Također, ima prag zabranjenog područja od 0.8V do 2V, što nam omogućava čitanje i signala od 3.3V.



Slika 3.3 Logički analizator.

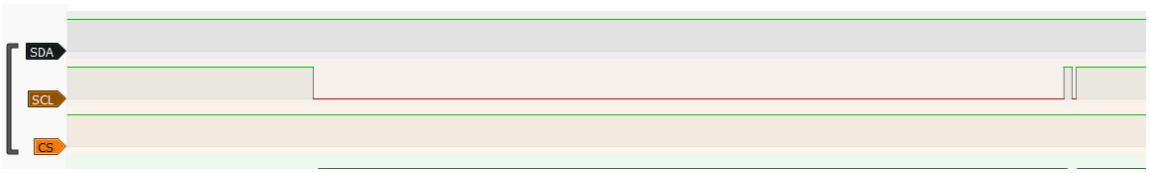
3.4 Programska podrška

Za upravljanje i iščitavanje podataka, koristi se open-source softver Sigrok Pulseview čije je sučelje prikazano na slici 3.4. On podržava dekodiranje 131 protokola, između kojih su I2C i NEC protokoli, što nam drastično olakšava stvari preskakanjem ručnog brojanja bitova. Za upotrebu ovog programa, potrebno je instalirati upravljački program pomoću alata koji dolazi u paketu sa instalacijom Pulseviewa zvan Zadig. Na njemu je potrebno odabrati spojeni logički analizer i kliknuti na gumb "Install driver". Pinove CH0, CH1, CH2 logičkog analizatora spajamo na pinove 1, 2 i 3

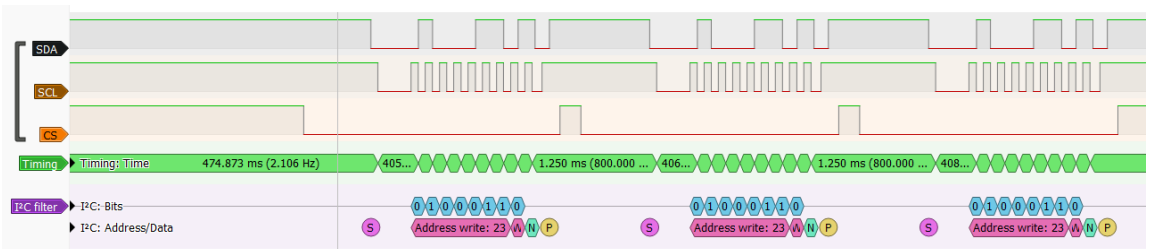


Slika 3.4 Pulseview program.

žutog priključka radija, te GND pin na masu. U programu je odabran korišteni "fx2lafw" uređaj, s frekvencijom uzorkovanja od 6MHz i brojem uzoraka od 1T što je maksimum koji korišteni program podržava. Ovakvim postavkama može se snimiti do 1T/6MHz = 166.67s. U programu se stisne gumb za početak snimanja, te se u automobilu upali radio. Stiskanjem gumba za uključivanje, pin 4 se dignuo na +12V što pretpostavljamo da daje signal ekranu da je radio upaljen. Prva komunikacija preko preostale 3 linije prikazana na slici 3.5 je spuštanje SCL-a, što je moguće da je samo kvar pri paljenju mikrokontrolera. Sljedeći prijenos su tri bajta prikazana na slici 3.6 šaljući adrese na sabirnicu, bajta za pisanje, te primanje NACK bita. Pošto se odspajanjem ekrana s radija i mjerenjem istog samo na radiju dobije jednak odziv, znamo da je u ovom slučaju radio naš master. Dodavanjem Timing kanala, možemo odrediti frekvenciju takta, koji između svakog impulsa iznosi 7kHz. Ova frekvencija je puno sporija od standardne frekvencije I2C protokola od 100kHz. Primjećujemo



Slika 3.5 Kvar pri paljenju radija.



Slika 3.6 Prva tri paketa.

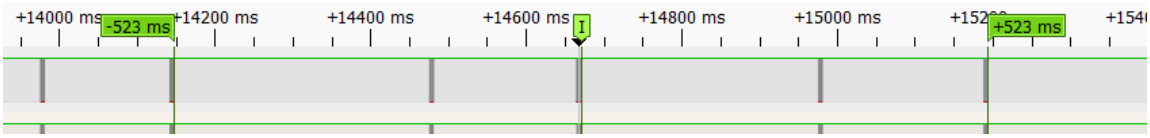
kako je pri svakom slanju treći kanal spušten, te se diže nakon slanja. Ovakvo ponašanje je nepostojeće u I2C komunikaciji, te poznavanjem SPI (eng. *Serial Peripheral Interface*) protokola podsjeća na CS (eng. *chip select*) liniju, te zbog toga je taj kanal tako i nazvan. Struktura sljedeća 4 paketa je prikazana u tablici 3.2. Nakon toga,

Tablica 3.2 Paketi poslani na I2C sabirnici.

Adresa (R/W) [ACK]	Data 1 [ACK]	Data 2 [ACK]
0x23 (W) [ACK]	0x01 [ACK]	0x10 [ACK]
0x23 (R) [ACK]	0x01 [ACK]	0x00 [ACK]
0x23 (W) [ACK]	0x01 [ACK]	0x10 [ACK]
0x23 (W) [ACK]	0x01 [ACK]	0x11 [ACK]

radio ima 500ms razmaka između pisanja bajtova 0x01 0x11, i razmak od 523ms od čitanja 0x01 0x01 koji su izmjereni na slici 3.7.

Kod čitanja, CS se spušta 660μs prije početnog uvjeta, a kod pisanja, 670μs prije. Možemo pretpostaviti da ekran spusti CS kad želi da se s njega čita, dok radio spusti CS ako želi pisati na ekran.



Slika 3.7 Razmaci između svakog stop uvjeta čitanja.

Prilikom stiskanja gumba za podizanje glasnoće, čita se sljedeća poruka: 0x04 0x82 0x91 0x00 0x03. Nakon kratke pauze, piše se: 0x0C 0x90 0x76 0x60 0x01 0x20 0x56 0x4F 0x4C 0x20 0x31 0x30 0x20. Za prvu poruku možemo pretpostaviti da prenosi informaciju koje je tipkalo pritisnuto, dok druga poruka sadrži informacije koje će se prikazati na ekranu. Pretvaranjem poruke iz heksadekadskog zapisa u ASCII (eng. *American Standard Code for Information Interchange*) format, dobivaju se rezultati prikazani na slici 3.8. Inženjeri koji su implementirali ovaj komunikacijski protokol su bili dovoljno srdačni da koriste lako prepoznatljivu abecedu u ASCII pri zapisivanju slova na ekran. Prijašnji bajtovi najvjerojatnije određuju adrese registra na koji se piše, ili direktno govore procesoru željenu naredbu. Stiskanjem svakog tipkala na komandi volana, možemo odrediti njihove podatke pri čitanju s ekrana, koji su zapisani u tablicama 3.3 i 3.4. Ovakvu nestandardnu i sporu I2C komunikaciju možemo objasniti zbog nepostojeće hardverske podrške na Renesas H8/3834 mikrokontroleru koji se nalazi u ekranu. Svaki protokol koji nema hardversku podršku potrebno je „bitbangat“ što zauzima resurse glavnog djela mikrokontrolera koji vjerojatno nije zaslužan samo za radiosustav, već i praćenje vremena kao i temperature unutrašnjosti vozila prikazanoj na ekranu. Provjeravanjem datasheeta najbliži protokol koji H8 mikrokontroler podržava je SCI (eng. *Serial communication Interface*).[4] Na njemu su 3 SCI kanala, od kojeg drugi jedini objašnjava naš CS pin. Sva 3 kanala koriste odvojene ulazne i izlazne pinove stoga ovakav način prijenosa

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
0C 90 76 60 01 20 56 4F 4C 20 31 30 20           ..v`. VOL 10
```

Slika 3.8 Paket prikazan crvenim slovima u programu HxD.

Tablica 3.3 Paketi poslani na I2C sabirnici pritiskom tipkala.

VOL+	0x04	0x82	0x91	0x00	0x03
VOL-	0x04	0x82	0x91	0x00	0x04
Source desni	0x04	0x82	0x91	0x00	0x01
Source lijevi	0x04	0x82	0x91	0x00	0x02
Pauza	0x04	0x82	0x91	0x00	0x05
OK	0x04	0x82	0x91	0x00	0x00
Enkoder gore	0x04	0x82	0x91	0x01	0x41
Enkoder dolje	0x04	0x82	0x91	0x01	0x01

Tablica 3.4 Paketi poslani na I2C sabirnici držanjem tipkala.

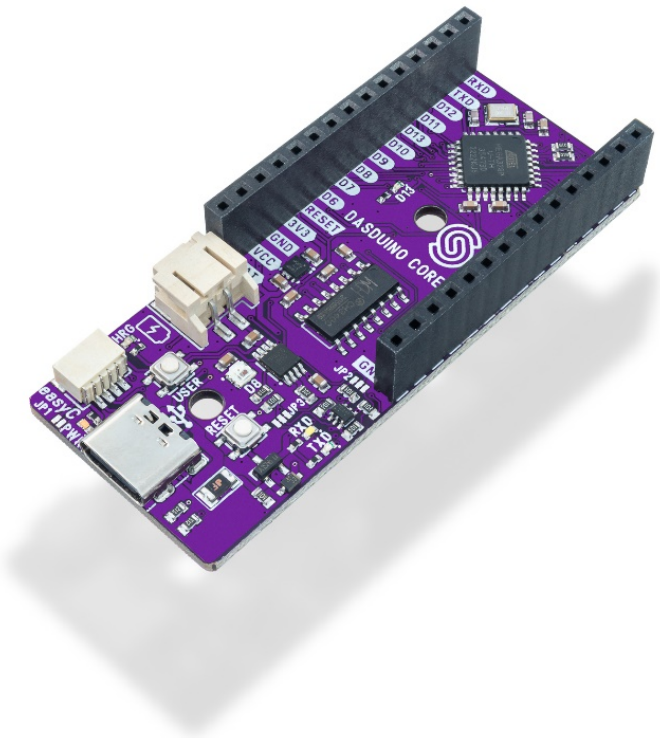
VOL+	0x04	0x82	0x91	0x00	0x43
VOL-	0x04	0x82	0x91	0x00	0x44
Source desni	0x04	0x82	0x91	0x00	0x41
Source lijevi	0x04	0x82	0x91	0x00	0x42
OK	0x04	0x82	0x91	0x00	0x40

je vanjskom promatraču nejasan bez izvornog koda. S ovako izmjerenim podatcima, možemo zaključiti da je u nastavku potrebno realizirati sustav koji će inicijalizirati ekran, čitanjem sljedećih podataka odrediti koji je gumb pritisnut, isti prevesti u NEC protokol, te poslati zamjenskom radiju.

Poglavlje 4

Posredni mikrokontroler

Za prethodno opisanu zadaću odabrana ja Dasduino CORE razvojna pločica (slika 4.1) bazirana na ATmega328p mikrokontroleru, te podržava programiranje na Arduino platformi koja ima ekstenzivnu korisničku podršku. ATmega328p koristi 5V kao TTL (eng. *Transistor-Transistor Logic*) logiku stoga nije potrebna upotreba konvertera razine logičkog napona. Jednostavna je za programirati, te koristi modificiranu verziju C++ programskog jezika. Radi na 16MHz što je više nego dovoljno, s obzirom da Renesasov čip ima maksimalnu moguću brzinu više od 3 puta sporiju. Manjih je dimenzija od Arduino Uno ploče, no zato nema linearni regulator kojim bi se pojednostavilo napajanje mikrokontrolera. Stoga, potrebno je osmisliti način napajanja pločice s 5V. Također, pojednostavljeno je prototipiranje radi headera koji su udaljenosti višekratnika 0.1 inča (2.54mm). Arduino Uno ima manu što su dva headera nejednako pomaknuta, stoga ga je nemoguće staviti na prototipnu tiskanu ploču. Za početak, prilikom testiranja koda napajanje će se izvoditi s USB (eng. *Universal Serial Bus*) porta. SDA i SCL pinovi su označeni s A4 i A5, te ih spajamo na ekran. CS možemo spojiti na D2 ili D3 pin, jer oni jedini podržavaju interruptove. Interrupt je događaj koji procesor može prepoznati usred izvršavanja neke druge radnje, te to može biti rastući brid, padajući brid, nisko stanje, ili promjena stanja (LOW u HIGH ili HIGH u LOW). Prepoznavanjem događaja, pokrene se ISR (eng. *interrupt service routine*) funkcija programa.

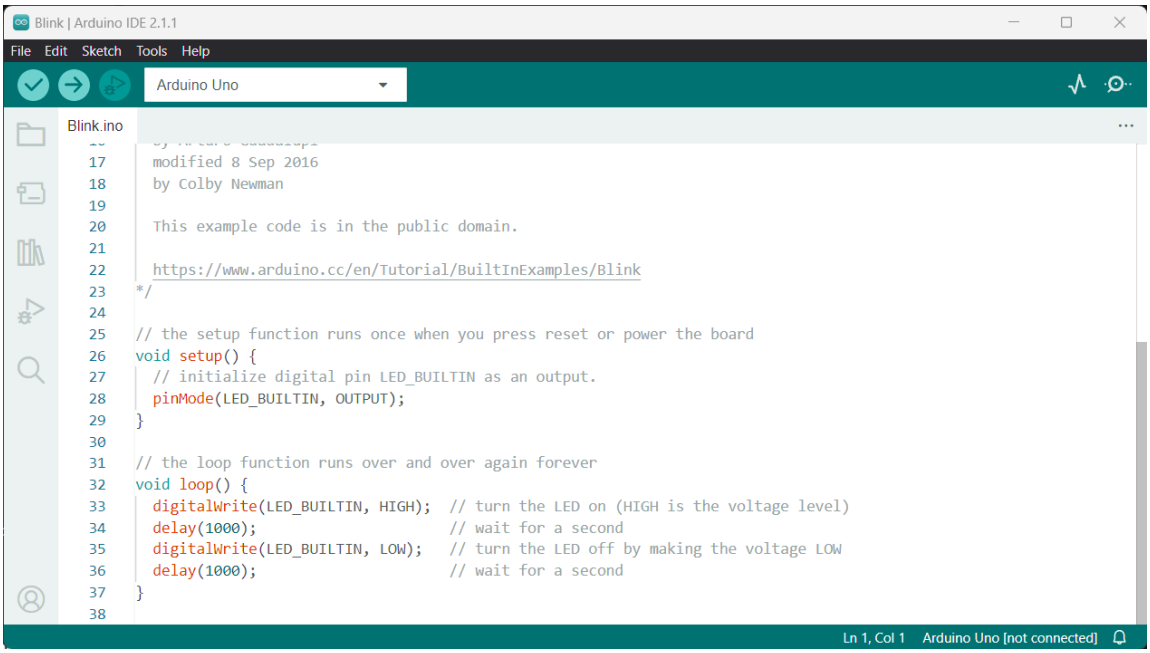


Slika 4.1 Dasduino CORE.

4.1 Priprema Arduino IDE

Klasični Arduino proizvodi koriste Atmega16u2 ili Atmega8u2 kao USB u Serial konverter, te s instalacijom Arduino IDE programskog paketa, automatski se instaliraju i upravljački programi. Za korištenu razvojnu pločicu ovo nije slučaj, jer je njegov USB u Serial konverter baziran na CH340C integriranom krugu, te je potrebno instalirati posebni upravljački program koji se može preuzeti na službenoj stranici proizvođača Dasduina. Instalacija Arduino IDE programa je jednostavna, te nakon što je instaliran može se isprobati funkcija spajanja programa s mikrokontrolerom, i programiranje testnog koda iz izbornika File → Examples → 1.Basics → Blink. Prilikom uploada obraća se pažnja na odabir točnog uređaja spojenog na COM (eng. *communication port*). Nije potrebno instalirati knjižnicu od Dasduina, već je mo-

guć odabir Arduino Uno razvojne pločice pošto koriste isti bootloader. Uspješnom kompilacijom i uploadanjem koda, LED (eng. *light-emmiting diode*) na razvojnoj pločici počne treperiti. Program se sastoji od dvije funkcije vidljive na slici 4.2, prva funkcija imenom `setup()` se izvršava, te služi većinom za početno konfiguriranje, dok se funkcija `loop()` izvodi u beskonačnoj petlji.



Slika 4.2 Arduino IDE sa Blink primjerom programa.

Poglavlje 5

Programiranje čitanja tipkala

Naredbom `#include` u program pozivamo knjižnicu Wire.h koja nam služi kontroliranju I2C hardvera, također zvanim „two-wire interface“ na jednostavan način. Predprocesorske naredbe `#define` nam omogućuju sužavanje više linija koda u jednu naredbu kojom smanjujemo nered i pojednostavljujemo čitanje. U ovom slučaju makro `L3_HIGH` poziva definiranje pina D2 kao ulaznog pina, tj. pin s visokom impedancijom, koji služi mikroprocesoru za čitanje stanja pina, te na njega vežemo prije objašnjeni interrupt s padajućim bridom koji poziva funkciju `L3low()`. Makro `L3_LOW` miče interrupt s pina kako ne bi pozvao sam sebe u nadolazećim linijama koje postavljaju pin kao izlazni nisko-impedantni, te ga spuštaju na logičku nulu. Predprocerske `#define` naredbe moramo definirati prije pozivanja knjižnice IRRemote kako bi se inicijalizirala s našim postavkama. To je knjižnica koja se koristi za slanje NEC protokola radiju. Definicija na prvoj liniji se koristi za smanjivanje veličine programa, druga se koristi za onemogućavanje slanja noseće frekvencije, i treća za korištenje open-drain načina rada.

```
#define DISABLE_CODE_FOR_RECEIVER                                    1
#define USE_NO_SEND_PWM                                              2
#define USE_OPEN_DRAIN_OUTPUT_FOR_SEND_PIN                           3
#include "PinDefinitionsAndMore.h"  // Define macros for input      4
   and output pin etc.
#include <IRremote.hpp>                                              5
#define IR_SEND_PIN 5                                                6
```

```
#define L3_HIGH \
pinMode(InterruptD2, INPUT_PULLUP); \
attachInterrupt(digitalPinToInterrupt(InterruptD2), L3low,
   FALLING);
#define L3_LOW \
detachInterrupt(digitalPinToInterrupt(InterruptD2)); \
pinMode(InterruptD2, OUTPUT); \
digitalWrite(InterruptD2, LOW);
```

Izvan funkcija definirane su globalne varijable koje vrijede unutar svih funkcija. Veličinu varijabli biramo ovisno o najvećoj mogućoj vrijednosti koje mogu doseći. Ključna riječ "const" definira konstantu, dakle prevoditelj (eng. *compiler*) štedi memoriju i vrijeme pristupa varijabli jer je ona konstantna. Volatile identifikator govori prevoditelju da se varijabla `isLow` može promjeniti u bilo kojem djelu programa, što je pogodno za ISR. Varijabla `button[]` je polje pokazivača, gdje svaki pokazivač pokazuje na jedan od stringova.

```
uint8_t buf[5] = { 0, 0, 0, 0, 0 };
uint8_t i;
const uint8_t InterruptD2 = 2;
uint32_t previousMillis = 0;
const uint16_t interval = 500;
uint32_t currentMillis;
volatile bool isLow = false;
char* button[] = { "VOLPLUSP",
     "VOLPLUSH",
     "VOLMIN P",
     "VOLMIN H",
     "SRC R P ",
     "SRC R H ",
     "SRC L P ",
     "SRC L H ",
     "PAUSE ",
     "OK PRESS",
```

```
    "OK HOLD ",
    "ROT UP ",
    "ROT DOWN" };
```

U funkciji `setup()` prvo je postavljen D2 kao ulazni pin. Potom se poziva objekt `Wire` koji inicijalizira hardware za I2C komunikaciju. Funkcija setWireTimeout postavlja vrijeme u milisekundama nakon kojih će se sabirnica kao i ugrađeni I2C hardware resetirati ukoliko je došlo do blokiranja sabirnice. Sljedeće dvije linije su jedine neprenosive linije koda u programu, značeći da ako bi se htjeli prebaciti na bilo koji drugi mikrokontroler kojim se želi obaviti ista stvar, parametri se moraju posebno namjestiti. Atmega328p koristi prescaler na registru TWSR kako bi generirao taktne impulse na I2C sabirnici. Registar TWSR ima raspored bitova prikazan tablicom 5.1, iz koje su nam zapravo jedino važna 2 LSB (eng. *least significant bit*). Ovisno o vrijednostima tih dvaju bita, u tablici 5.2 prikazana je vrijednost prescalera koji se koristi u jednadžbi (5.1) za izračun frekvencije generiranja taktnog signala.[5]

Tablica 5.1 Raspored bitova u TWSR registru.

TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0

Tablica 5.2 Vrijednost prescalera za vrijednosti TWPS bitova.

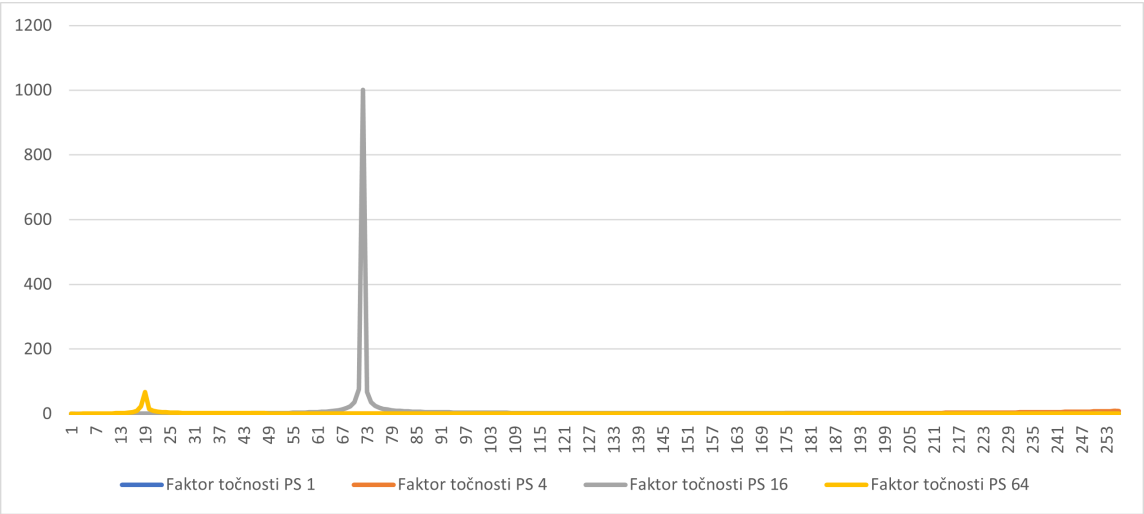
TWPS1	TWPS0	Vrijednost prescalera
0	0	1
0	1	4
1	0	16
1	1	64

$$f_{SCL} = \frac{f_{cpu}}{16 + 2 * TWBR * 4^{TWPS}} \tag{5.1}$$

U prethodnom poglavlju smo vidjeli da nam je potrebna frekvencija od 7kHz. Izračunavanjem dobivene frekvencije korištenjem formule (5.1) za svaku vrijednost

prescalera i bitratea, kao i NKMM (Nakon kratke matematičke manipulacije) korištenjem formule (5.2) za izračun faktora točnosti, dobivamo najveću točnost pri vrijednosti prescalera od 16, kao i vrijednosti TWBR registra od 71 koji nam rezultiraju frekvencijom od 6993Hz. Faktori točnosti su prikazani grafom na slici 5.1. Makro `L3_LOW` se poziva prije svakog pisanja ili čitanja, dok se makro `L3_HIGH` poziva završetkom, kako bi se oslobodila linija CS. Funkcije `delay()` su dodane za kompatibilnost jer se bitbangani kod u ekranu ne "pripremi" dovoljno brzo pa može doći do preskakanja čitanja prvog bita. Ova solucija nije idealna, no radi sinkronog redoslijeda izvođenja programa proces se izvađa s točnom vremenskom odgodom. Funkcija `beginTransmission()` određuje adresu na koju se šalju podatci, `write()` sprema te podatke u buffer, a pozivanjem `endTransmission()` se svi podaci iz buffera pišu na sabirnicu.

$$Faktor\,točnosti = \frac{f_{potrebna}}{f_{potrebna} \mid - f_{dobivena} \mid} \tag{5.2}$$



Slika 5.1 Faktori točnosti za različite vrijednosti TWSR i TWBR registra.

```
void setup() {                                                    1
    pinMode(InterruptD2, INPUT_PULLUP);                           2
```

```
    Wire.begin();
    IrSender.begin(DISABLE_LED_FEEDBACK);
    Wire.setWireTimeout(500000,true);
    TWSR = 0b10;
    TWBR = 71;
    L3_LOW;
    delayMicroseconds(600);
    for (i = 0; i <= 2; i++) {
        Wire.beginTransmission(0x23);
        Wire.endTransmission();
        delayMicroseconds(1000);
    }
    L3_HIGH;
    delay(1000);
    L3_LOW;
    Wire.beginTransmission(0x23);
    Wire.write(0x01);
    Wire.write(0x10);
    Wire.endTransmission();
    L3_HIGH;
}
```

Funkcija `text`L3low() je ISR funkcija koja se poziva padajućim bridom na pinu D2. Unutar te funkcije se varijabli `isLow` dodjeljuje vrijednost 1, kako bi mikrokontroler u izvršavanju glavnog programa znao da je došlo do interrupta. ISR funkcije bi trebale biti što kraće.

```
void L3low() {
    isLow = true;
}
```

U `loop()` funkciji se konstantno čita stanje varijable `isLow` kako bi detektirali je li došlo do interrupta, ako je u stanju logičke jedinice pozivamo funkciju

`readButtons()`, a ostatkom funkcije se izvršava prije spomenuto konstantno čitanje i pisanje za keep-alive.Funkcija `loop()` je karakteristična funkcija u Arduinu, jer je ona jednaka funkciji `while(true)`, to jest ponavlja se svakim krajem izvođenja programa. Funkcija `memset()` postavlja stanje svake varijable u nizu na heksadecimalnu vrijednost 0xFF jer je u petlji za čitanje programa preskačemo.

```
void loop() {
     if (isLow) {
          readButtons();
     }
     currentMillis = millis();
     if (currentMillis - previousMillis >= interval) {
          previousMillis = currentMillis;
          L3_LOW;
          Wire.beginTransmission(0x23);
          Wire.write(0x01);
          Wire.write(0x11);
          Wire.endTransmission();
          delay(40);
          if (Wire.requestFrom(0x23, 5)) {
               while (Wire.available()) {
                    for (i = 0; i <= 1; i++) {
                         buf[i] = Wire.read();
                    }
               }
          }
    memset(buf, 0xFF, sizeof(buf));
          L3_HIGH;
     }
}
```

Funkcija `readButtons()` je nešto dulja, te se nalazi u prilogu 1. Pozivanjem funkcije, ona spušta pin D2, te šalje zahtjev ekranu za čitanje 5 bajtova. U switch-case funkcijama uspoređuje pročitane bajtove zapisane u varijabli `buf[]` te poziva funkciju

`PressButton()` u kojoj su definirani bajtovi koji pripadaju ekvivalentu fukciji tipkala na autu koje Kenwood radio razumije. Funkcija `sendNEC()` objekta `IrSender` šalje određeni bajt na adresu 0xb9 koja je adresa radija. Naredbe od radija se nalaze u tablici 5.3.

```
void PressButton(byte batun) {
    /*
    1   Volume + press
    2   Volume + hold
    3   Volume - press
    4   Volume - hold
    5   Source right press
    6   Source right hold
    7   Source left press
    8   Source left hold
    9   Pause
    10  OK press
    11  OK hold
    12  Rotary up
    13  Rotary down
    */
    const byte batunNEC[] = {0x14,
         0x14,
         0x15
         0x15,
         0x0c,
         0x0c,
         0x0d,
         0x0d,
         0x0e
         0x07,
         0x07,
         0x0b,
         0x0c,
    }
```

```
    IrSender.sendNEC(0xb9, batunNEC[batun]);
    memset(buf, 0xFF, sizeof(buf));

}
```

Tablica 5.3 Funkcije s naredbama od Kenwood radio uređaja.[1]

Funkcija	NEC Bajt	Funkcija	NEC Bajt	Funkcija	NEC Bajt
0	0x00	8	0x08	volume +	0x14
1	0x01	9	0x09	volume -	0x15
2	0x02	track -	0x0a	mute	0x16
3	0x03	track +	0x0b	tuner	0x1c
4	0x04	reverse	0x0c	tape	0x1d
5	0x05	forward	cd	0x0d	0x1e
6	0x06	play/pause	0x0e	cd-md-ch	0x1f
7	0x07	source	0x13	dnpp	0x5e

Funkcija `WriteToDisplay()` je opcionalna, te služi pomaganju pri uklanjaju grešaka u kodu prikazom stisnutog gumba iz stringova `button[]`. Također, ona resetira varijablu `isLow` kako bi bila spremna za sljedeće okidanje.

```
void WriteToDisplay(char* msg) {
    return;
    delayMicroseconds(23360);
    DEBUG_SERIAL.println("WriteToDisplay");
    L3_LOW;
    delayMicroseconds(630);
    Wire.beginTransmission(0x23);
    Wire.write(0x0C);
    Wire.write(0x90);
    Wire.write(0x76);
    Wire.write(0x60);
    Wire.write(0x01);
    Wire.write(msg);
```

```
    Wire.endTransmission();                                    14
    DEBUG_SERIAL.println("written to disp");                  15
    L3_HIGH;                                                   16
    previousMillis = currentMillis;                            17
    isLow = false;                                             18
    memset(buf, 0xFF, sizeof(buf));                            19
}                                                              20
```

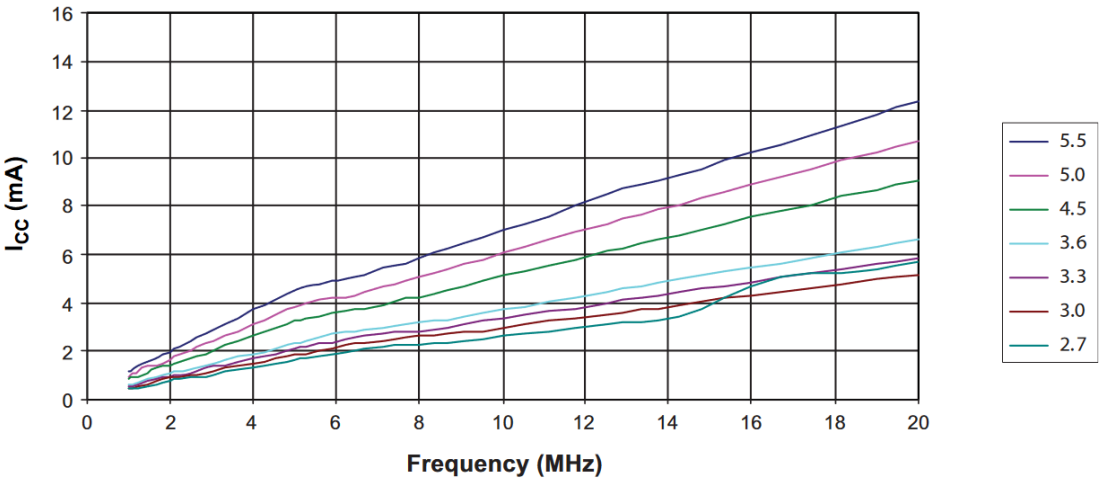
Poglavlje 6

Integriranje napajanja

Napon potpuno napunjene baterije automobila je 12.7V, no alternator može dati van napone i do 14.5V. Ako je automobil ugašen, naš uređaj ne smije trošiti previše struje, kako se akumulator ne bi ispraznio. Tipična vrijednost struje koju vuče ugašeni automobil iznosi do 40mA. Glavni potrošači u tom slučaju su centralno zaključavanje, alarm, čitač kartice, itd... Tipičan kapacitet olovnog akumulatora automobila je oko 80Ah. Pri gore navedenoj struji od 40mA, bateriji bi trebalo 2000h prema izrazu (6.1) da se potpuno isprazni.

$$t = \frac{80Ah}{40mA} = 2000h \tag{6.1}$$

Na slici 6.1 je prikazan graf koji pokazuje potrošnju korištenog mikrokontrolera, ovisno o frekvenciji radnog takta i napona napajanja u normalnom režimu rada. Korišteni mikrokontroler se napaja s 5V, te radi na frekvenciji od 16MHz, što po datasheetu daje aktivnu struju od 9mA ne uključujući bilo kakvu periferiju mikrokontrolera. Ovo nije loše, no svakako želimo minimizirati potrošnju razvojne pločice. Automobil ima žicu na kojoj daje 12V nakon kontakta, ili 0V prije kontakta. Ova žica nije baš najpogodnija za napajanje, jer ima jako malu dozvoljenu struju, pa ju ne smijemo opteretiti kako ne bi došlo do izgaranja osigurača ili u najgorem slučaju upravljačke jedinice putničkog prostora. Upotreba običnog releja se ne čini najboljim izborom, radi veličine i zvuka zavojnice kod sklapanja, kao i povećani stres pokretnih radi vibracija u automobilu. Poluvodički releji su skupi, te također dimenzionalno nepogodni.



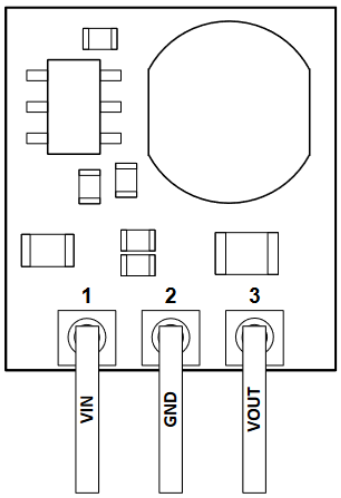
Slika 6.1 Graf potrošnje struje mikrokontrolera s obzirom na frekvenciju radnog takta.

6.1 Komponente

Za napajanje mikrokontrolera korišten je MCP1407 MOSFET driver, koji može podnijeti i do 6A struje, te s apsolutnim maksimumom ulaznog napona od 20V pogodan je za ovu svrhu. Za galvansku izolaciju napona s baterije od napona kontakta korišten je 4N35 optocoupler. Za inicijalizaciju ekrana, potrebno je dovesti 12V na pin 5 žutog priključka, te za to se koristi BS170 MOSFET, izabran zbog mogućnosti otvaranja vrata s naponom od 5V[6]. TPSM84 silazni pretvarač se koristi za spuštanje napona akumulatora na napon pogodan razvojnoj pločici.

6.1.1 TPSM84 silazni pretvarač

Za pretvaranje napona baterije automobila, korišten je silazni pretvarač prikazan na slici 6.2 s pripadajućim funkcijama pinova u tablici 6.1 kao i njegove blok sheme na slici 6.3[7]. On je bolja alternativa linearnom regulatoru, ponajviše radi učinkovitosti, koja je veliki nedostatak linearnog regulatora. Linearni regulatori razliku u ulaznom i izlaznom naponu pretvaraju u toplinu, te su zbog toga vrlo neučinkoviti. Silazni

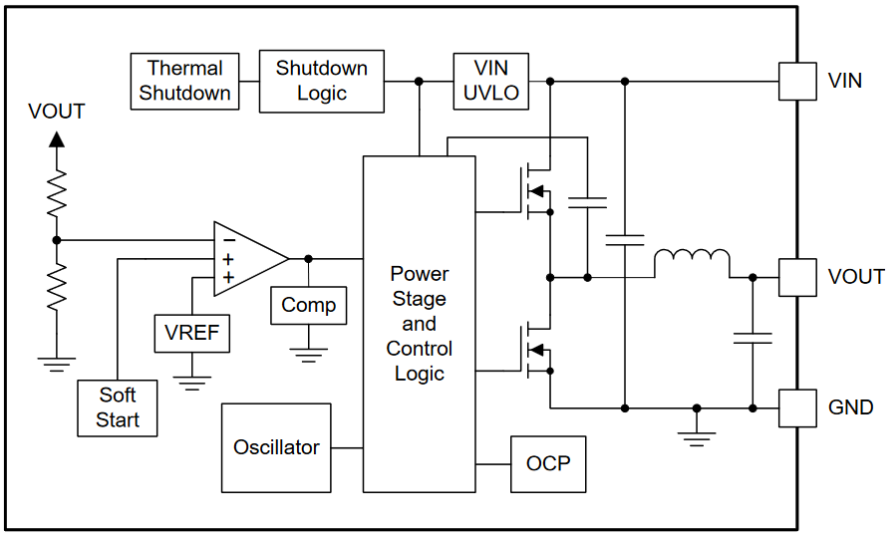


Slika 6.2 Skica TPSM84 silaznog pretvarača s numeriranim pinovima.

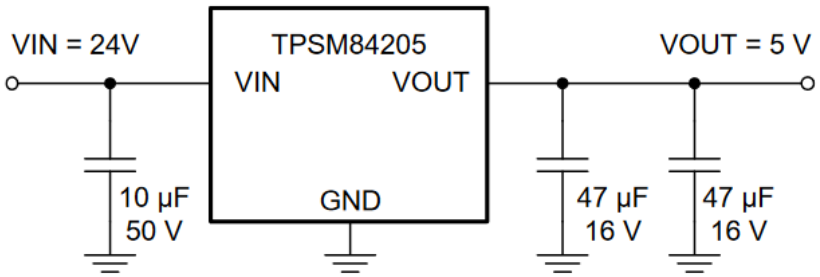
Tablica 6.1 Funkcije pinova TPSM84 silaznog pretvarača.

Broj pina	Oznaka pina	Funkcija pina
1	VIN	Ulazni napon
2	GND	Uzemljenje
3	VOUT	Izlazni napon

pretvarači pretvaraju napon PWM (eng. *pulse width modulation*) modulacijom, gdje je radni ciklus jednak kvocijentu izlaznog i ulaznog napona. U ovom slučaju, to je 5/12 *100 = 41.67%. Pretvarač ima fiksni izlazni napon od 5V, te samoregulaciju promjenom ulaznog napona, kao i zaštite od prenapona ili kratkog spoja. Na slici 6.4 je prikazana tipična primjena korištenog silaznog pretvarača. Ulazni i izlazni kapaciteti prikazani na slici 6.4 moraju biti keramičkog tipa. Minimalni kapacitet ulaznog kondenzatora je 10μF, dok na izlazu su dva keramička kondenzatora od 47μF u paraleli, što daje 94μF kapaciteta. Na izlazu se preporuča dodati još jedan nekeramički kondenzator za glađenje prijelaznih pojava. Vrijednost izlazne valovitosti napona je prikazana na grafu 6.5. Pri malim opterećenima, izlazna valovitost je zanemariva i iznosi 6mV za naš mikrokontroler, koji ima brown-out reset zaštitu tek na 3.7V.



Slika 6.3 Blok shema TPSM84 silaznog pretvarača.

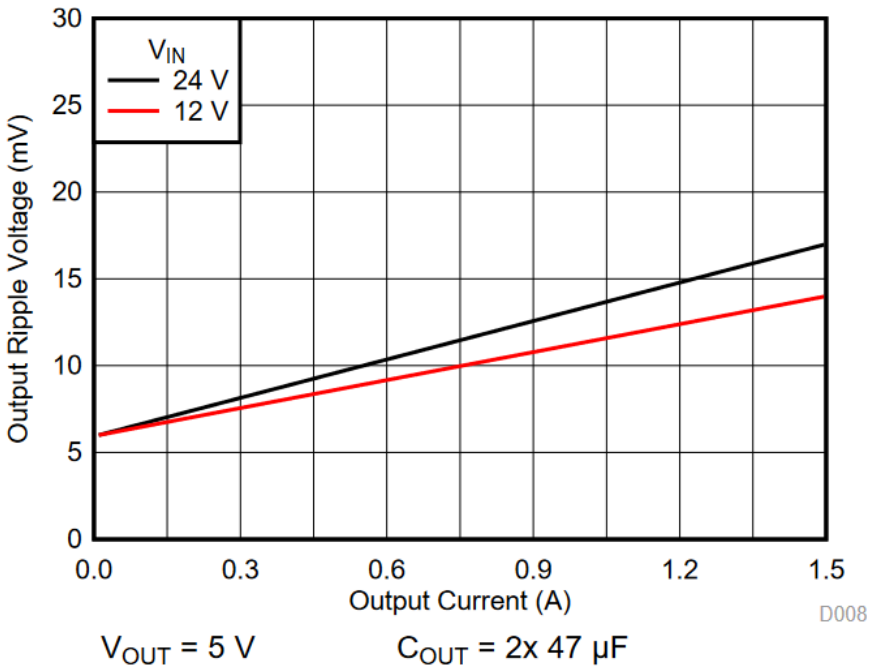


Slika 6.4 Tipičan spoj TPSM84 silaznog pretvarača.

6.1.2 MCP1407 MOSFET driver

MCP1407 je neinvertirajući MOSFET driver velike brzine. Njegov blok dijagram je prikazan na slici 6.6[8]. Vidljivo je da pri davanju upravljačkog napona na input pin, MOSFET-i na outputu daju napon s ulaza VDD. Logički visoki napon je minimalno 2.4V, stoga se može kontrolirati i TTL/CMOS (eng. *complementary metal-oxide semiconductor*) logikom. Najveći napon koji možemo dovesti na input drivera je VDD + 0.3V, što je odlično, jer bi napon baterije i napon nakon kontakta trebali

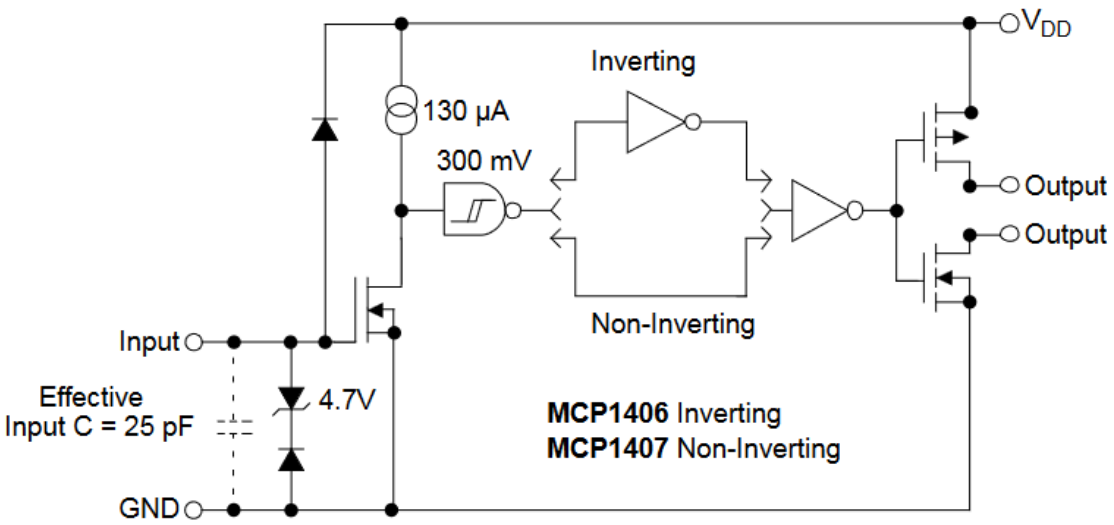
biti isti. U korištenom PDIP kućištu integriranog kruga prikazanog na slici 6.7 pridružena su objašnjenja pinova u tablici 6.2. VDD ima raspon od 4.5 do 18V, te mora biti spojen na masu s kondenzatorom. Kondenzatori pružaju lokalizirane niskoimpedantne puteve za struju koja je pružana trošilu. INPUT je ulaz visoke impedancije upravljan TTL/CMOS naponima. On također ima histerezu između visokog i niskog stanja kako bi se mogao pokretati i sporo rastućim signalom, te kako bi smanjio smetnje. GND mora imati spoj niske impedancije na uzemljenje izvora



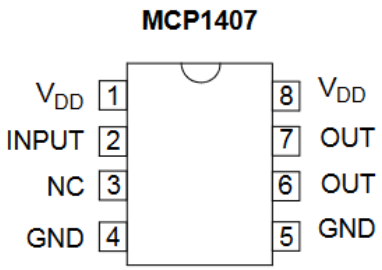
Slika 6.5 Izlazna valovitost napona TPSM84 silaznog pretvarača.

Tablica 6.2 Funkcije pinova MCP1407 MOSFET drivera.

Broj pina	Oznaka pina	Funkcija pina
1,8	VDD	Napon napajanja
2	INPUT	Upravljački ulaz
3	NC	Nije spojeno
4,5	GND	Uzemljenje
6,7	OUT	CMOS Push-Pull izlaz

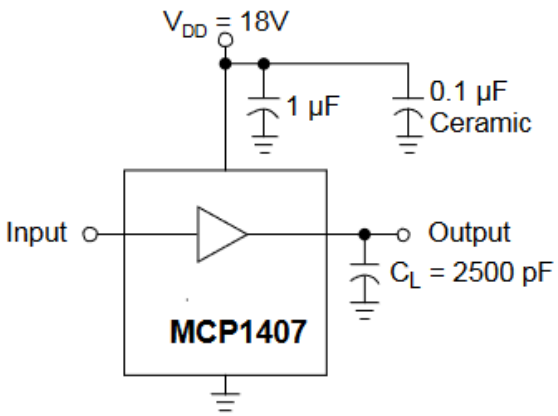


Slika 6.6 MCP1407 blok dijagram.



Slika 6.7 MCP1407 PDIP kućište s oznakama pinova.

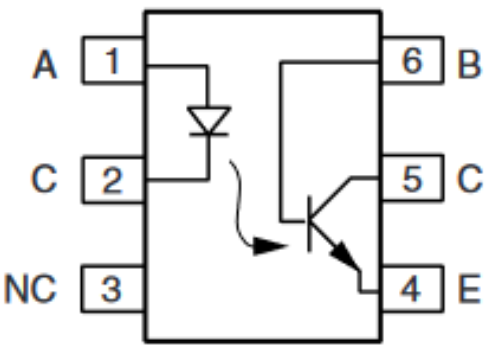
napona, jer visoke struje teku kroz njega prilikom pražnjenja kapacitivnog trošila. OUTPUT je CMOS push-pull izlaz koji može dati do 6A struje. Izlazni pinovi imaju do 1.5A sigurnost protiv reverzne struje. Minimalne i maksimalne vrijednosti kondenzatora korištenih kao primjer tipičnog spoja na slici 6.8 se mogu pronaći u njegovom datasheetu.



Slika 6.8 Tipičan spoj MCP1407 MOSFET drivera.

6.1.3 4N35 Optocoupler

Optocoupleri su bipolarni tranzistori kojima je baza kontrolirana LE diodom, pa je stoga galvanski izolirana. Slika 6.9 kao i tablica 6.3 prikazuju funkcije pinova korištenog optocouplera[9]. Njega se upotrebljava za galvansko odvajanje žice napajanja od 12V poslije kontakta od žice koja je spojena direktno na akumulator auta. Pomoću optocouplera se daje signal na upravljački ulaz MCP1407 MOSFET drivera za aktiviranje napajanja razvojne pločice. Optocoupleri inače nemaju poseban pin

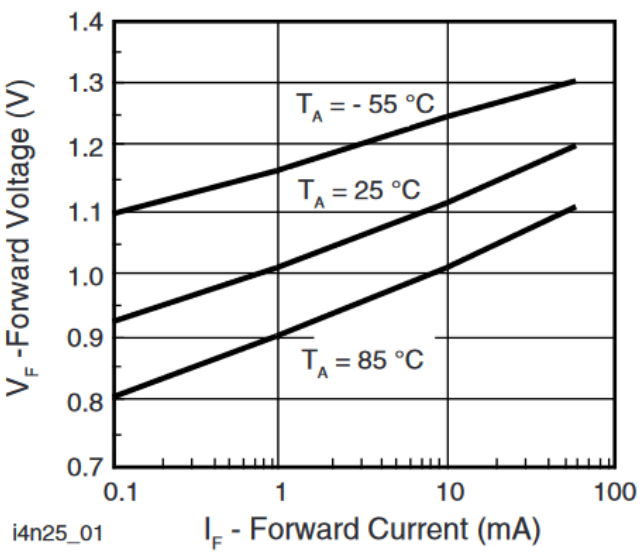


Slika 6.9 4N35 PDIP kućište s oznakama pinova.

za izvod baze, no u ovom slučaju će biti odspojen, jer se bipolarni tranzistor može kontrolirati samo LE diodom. Po grafu na slici 6.10 može se odrediti struja potrebna za otvaranje tranzistora. Na 25 stupnjeva Celzijevih propusna struja može biti 1mA. Uz tu struju ima 1mW disipacije snage u toplinu.

Tablica 6.3 Funkcije pinova 4N35 optocouplera.

Broj pina	Oznaka pina	Funkcija pina
1	A	Anoda
2	C	Katoda
3	NC	Nije spojeno
4	E	Emiter
5	C	Kolektor
6	B	Baza



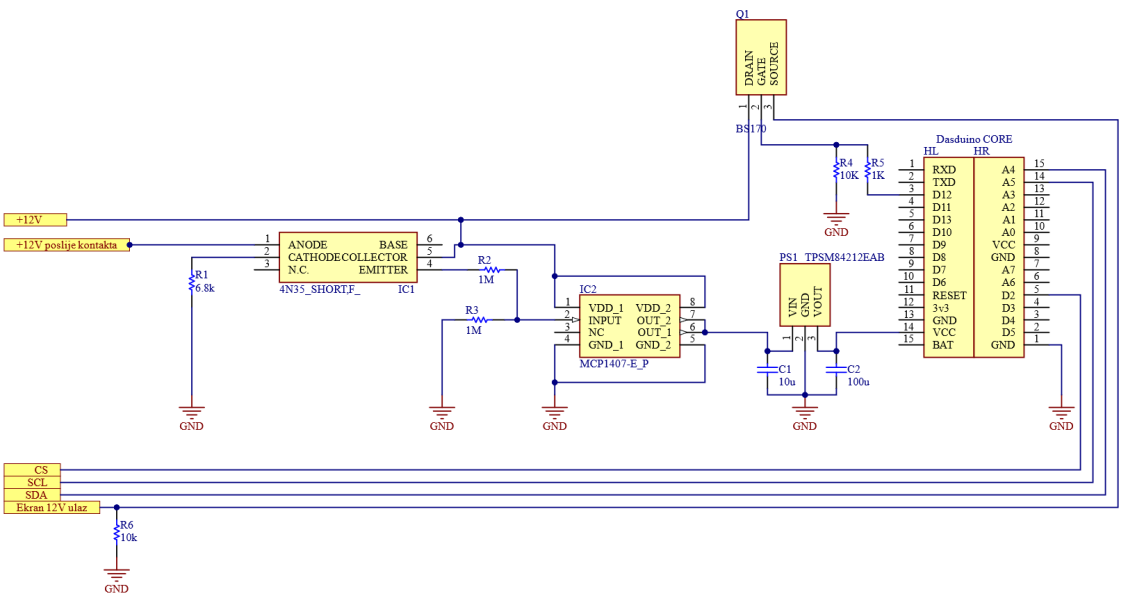
Slika 6.10 Odnos napona i struje propuštanja optocouplera.

6.2 Napajanje

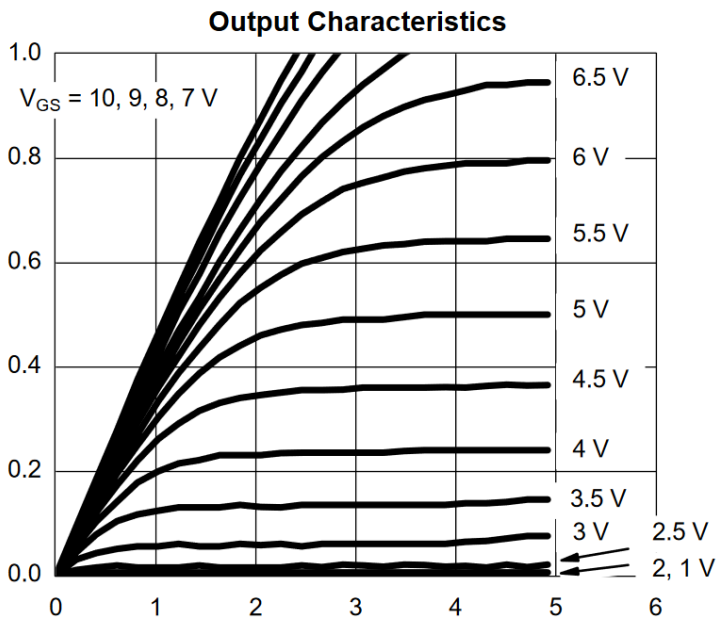
Shema gotovog napajanja prikazana je na slici 6.11. Otpornik R1 koristimo za ograničavanje struje kroz diodu optocouplera na 2mA. Otpornici R2 i R3 čine otporničko djelilo, ograničavaju struju punjenja kapaciteta na INPUT pinu MCP-a, te mu R3 omogućava i pražnjenje. Na izlazu MCP-a spajamo silazni pretvarač TPSM84 koji ima ulazni kapacitet C1 za smanjivanje visokofrekventnih šumova i izlazni C2 za glađenje po datasheetu. Njegov izlaz je spojen na V_{CC} pin mikrokontrolera. MOSFET BS170 ima R5 kao limitator struje punjenja gate-a i R4 kao pull-down otpornik. Skupa čine otporničko djelilo koji na vrata tranzistora daju napon od:

$$V_{iz} = \frac{R5}{R4 + R5} * V_{ul} = 4.45V \tag{6.2}$$

Pri upaljenom pinu D12 na slici 6.12 se može vidjeti izlazna karakteristika koja nam daje I_{DS} od 0.35mA. Ovo nam je više nego dovoljno za napajanje logike koja upravlja ekranom. Za svaki slučaj je dodan otpornik R6 koji pri isključenju prazni ovu liniju, no ni izostanak tog otpornika nije neophodan. Na slikama 6.13 vidimo napajačku logiku i mikrokontroler spojen na eksperimentalnu pločicu, napajanu s 12

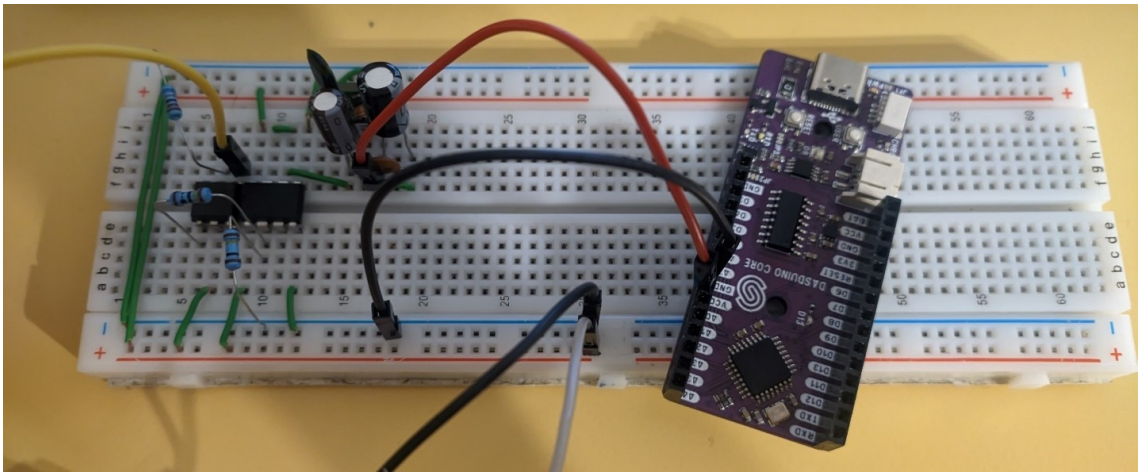


Slika 6.11 Shema napajanja.

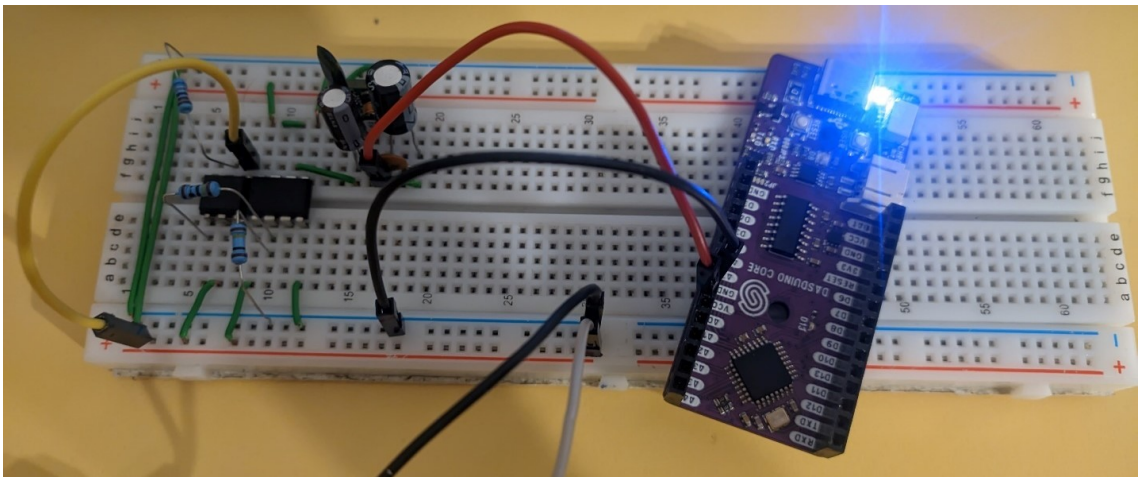


Slika 6.12 Izlazna karakteristika BS170 MOSFET-a.

voltnim adapterom. Na gornjoj slici žuta žica (+12V poslije kontakta) je odspojena, te njezinim spajanjem na donjoj slici se vidi kako je mikrokontroler uključen, što nam daje potvrdu o ispravnom radu našeg sklopa.



(a) Stanje pri naponu 0V na anodi optocouplera.



(b) Stanje pri naponu 12V na anodi optocouplera.

Slika 6.13 Napajanje spojeno s razvojnom pločicom na eksperimentalnoj ploči.

Poglavlje 7

Zaključak

Spajanjem mikrokontrolera s napajanjem, ekranom i ulazom za tipkalo na radiju, uspješno se kontrolira njegovim funkcijama preko ugrađenih tipkala na volanu automobila. Na ekranu je na slici 7.1 napisan proizvoljan tekst "Ide gas". Moguća je daljnja ekspanzija projekta s prenamjenom ekrana za pokazivanje korisnih informacija kao npr. prikazivanje trenutne pjesme spajanjem s mobilnim uređajem, ili odabir prikaza različitih dijagnostičkih parametara s CAN sabirnice automobila.

Slika 7.1 Tekst napisan na ekranu u središnjoj konzoli.

Bibliografija

[1] M. Bąbik. (2023, 09) Ford to kenwood radio adapter howto. , s Interneta, https://init6.pomorze.pl/projects/kenwood_ford/

[2] *Schemi elettrici Megane N.T. 8206A*, Renault S.A., 2002.

[3] (2023, 09) Nec infrared transmission protocol. Altium. , s Interneta, https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol

[4] *Renesas 8-Bit Single-Chip Microcomputer H8 Family/H8/300L Super Low Power Series*, Renesas Electronics, 08 2006, rev. 6.

[5] *8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash, 7810D–AVR*, Atmel, 01 2015.

[6] *Field Effect Transistor - N-Channel, Enhancement Mode, BS170, MMBF170*, Semiconductor Components Industries, LLC, 04 2022, rev. 7.

[7] *TPSM84203, TPSM84205, TPSM84212 1.5-A, 28-V Input, TO-220 Power Module*, Texas Instruments, 08 2017, rev. A.

[8] *6A High-Speed Power MOSFET Drivers MCP1406/07*, Microchip Technology Inc., 2016, rev. 3.

[9] *Optocoupler, Phototransistor Output, with Base Connection 4N35, 4N36, 4N37*, Vishay Semiconductors, 01 2010, rev. 1.2.

Pojmovnik

ACK eng. *acknowledged*. 8

ADC eng. *analog to digital converter*. 3

ASCII eng. *American Standard Code for Information Interchange*. 13

CMOS eng. *complementary metal-oxide semiconductor*. 30–32

COM eng. *communication port*. 16

CS eng. *chip select*. 12, 13, 15, 21

GND eng. *ground*. 7, 11, 31

I2C eng. *Inter-Integrated Circuit*. vii, ix, 6–8, 10, 12–14, 18, 20

IDE eng. *integrated development environment*. vii, 16, 17

IO eng. *Input/Output*. 7

ISR eng. *interrupt service routine*. 15, 19, 22

LED eng. *light-emmiting diode*. 17

LSB eng. *least significant bit*. 20

MOSFET eng. *metal-oxide-semiconductor field-effect transistor*. viii, ix, 28, 30, 31, 33, 35, 36

NACK eng. *not acknowledged*. 8, 11

NKMM Nakon kratke matematičke manipulacije. 20

OEM eng. *original equipment manufacturer*. 2, 3

PDIP eng. *plastic dual in-line package*. viii, 31–33

PWM eng. *pulse width modulation*. 28

SCI eng. *Serial communication Interface*. 13

SCL eng. *serial clock*. 7, 8, 11, 15

SDA eng. *serial data*. 7, 8, 15

SPI eng. *Serial Peripheral Interface*. 12

TTL eng. *Transistor-Transistor Logic*. 15, 30, 31

TWBR eng. *two-wire bitrate register*. vii, 21

TWSR eng. *two-wire status register*. vii, 20, 21

USB eng. *Universal Serial Bus*. 15

Sažetak

U ovom završnom radu je opisan proces obrnutog inženjerstva komunikacije OEM radija osobnog automobila s tipkalom na upravljaču automobila koristeći logički analizator, te prevađanje postojećeg komunikacijskog protokola zamjenskom radiju korištenjem razvojne pločice bazirane na Atmel-ovoj AVR platformi. Također je opisan i razvoj samostalnog rješenja automatskog napajanja mikrokontrolera radi smanjenja struje u načinu mirovanja, kao i smanjenje napona automobilske baterije na napon kompatibilan s mikrokontrolerom.

Ključne riječi **— Renault, Megane, TunerList, Komunikacijski protokoli, I2C, NEC, Logički analizator, Kenwood, KMM-123Y, Obrnuto inženjerstvo, Optimizacija upravljanja energijom**

Abstract

This undergraduate dissertation describes the process of reverse engineering the communication of an OEM car stereo with the steering wheel controls using a logic analyzer. It also involves the translation of the protocol to an aftermarket head unit using a development board based on the Atmel's AVR platform. Additionally, it encompasses the development of a standalone solution aimed at optimizing the power management of the microcontroller by reducing idle power consumption, as well as lowering the voltage provided by the car's battery to a level compatible with the microcontroller.

Keywords **— Renault, Megane, TunerList, Communication protocols, I2C, NEC, Logic analyzer, Kenwood, KMM-123Y, Reverse engineering, Power management**

Dodatak A

Prilog

```
#include "Wire.h"
#define DISABLE_CODE_FOR_RECEIVER
#define USE_NO_SEND_PWM
#define USE_OPEN_DRAIN_OUTPUT_FOR_SEND_PIN
#include "PinDefinitionsAndMore.h" // Define macros for
   input and output pin etc.
#include <IRremote.hpp>
#define IR_SEND_PIN 5
#define DEBUG true
#define DEBUG_SERIAL \
if (DEBUG) Serial
#define DEBUG_LINE \
if (DEBUG) digitalWrite
#define L3_HIGH \
pinMode(I2CInterrupt, INPUT_PULLUP); \
attachInterrupt(digitalPinToInterrupt(I2CInterrupt), L3low
   , FALLING);
#define L3_LOW \
detachInterrupt(digitalPinToInterrupt(I2CInterrupt)); \
pinMode(I2CInterrupt, OUTPUT); \
digitalWrite(I2CInterrupt, LOW);
```

```
uint8_t buf[5] = { 0, 0, 0, 0, 0 };
const uint8_t I2CInterrupt = 2;
uint32_t previousMillis = 0;
const uint16_t interval = 500;
uint32_t currentMillis;
volatile bool isLow = false;

char* button[] = { "VOLPLUSP",
  "VOLPLUSH",
  "VOLMIN␣P",
  "VOLMIN␣H",
  "SRC␣R␣P␣",
  "SRC␣R␣H␣",
  "SRC␣L␣P␣",
  "SRC␣L␣H␣",
  "PAUSE␣␣",
  "OK␣PRESS",
  "OK␣HOLD␣",
  "ROT␣UP␣",
  "ROT␣DOWN" };

//SDA = pin 4
//SCL = pin 5

void setup() {
  DEBUG_SERIAL.begin(2000000);
  DEBUG_SERIAL.println("\nserial␣start");
  pinMode(I2CInterrupt, INPUT_PULLUP);
  Wire.begin();
  Wire.setWireTimeout(500000, true);
  TWSR = 0b10;
```

```
  TWBR = 71;
  //TWCR &= ~(1<<TWEA);
  #if DEBUG == true
  bool serialConfirm = true;
  while (serialConfirm) {
    int ifOne = Serial.read();
    if (ifOne == '1') {
      serialConfirm = false;
    }
  }
  #endif //DEBUG == true // Waiting for Serial Monitor

  L3_LOW;
  delayMicroseconds(600);
  for (uint8_t i = 0; i <= 2; i++) {
    Wire.beginTransmission(0x23);
    Wire.endTransmission();
    delayMicroseconds(1000);
  }
  L3_HIGH;
  delay(1000);
  L3_LOW;
  Wire.beginTransmission(0x23);
  Wire.write(0x01);
  Wire.write(0x10);
  Wire.endTransmission();
  L3_HIGH;
  DEBUG_SERIAL.println("end␣setup");
}

void L3low() {
  isLow = true;
```

```
}

void loop() {
  if (isLow) {
    readButtons();
  }
  currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    DEBUG_SERIAL.println("millis start");
    L3_LOW;

    Wire.beginTransmission(0x23);
    Wire.write(0x01);
    Wire.write(0x11);
    Wire.endTransmission();
    delay(40);

    if (Wire.requestFrom(0x23, 5)) {
      while (Wire.available()) {
        for (uint8_t i = 0; i <= 4; i++) {
          buf[i] = Wire.read();
          // DEBUG_SERIAL.printf("buf[%x] = %x\n", i, buf[
             i]);
          if (buf[0] == 0x01 && buf[1] == 0x01) {}
        }
      }
    }
    memset(buf, 0xFF, sizeof(buf));
    L3_HIGH;
  }
}
```

```
void readButtons() {
  previousMillis = currentMillis;
  DEBUG_SERIAL.println("readbuttons");
  L3_LOW;
  if (Wire.requestFrom(0x23, 5))
  while (Wire.available()) {
    for (uint8_t i = 0; i <= 4; i++) {
      buf[i] = Wire.read();
      //DEBUG_SERIAL.printf("buf[%x] = %x\n", i, buf[i]);
    }
  }
  if (buf[0] == 0x01 && buf[1] == 0x00) {
    Wire.beginTransmission(0x23);
    Wire.write(0x01);
    Wire.write(0x10);
    Wire.endTransmission();
    delay(6);
    previousMillis -= 500;
    L3_HIGH;
    isLow = false;
    memset(buf, 0xFF, sizeof(buf));
    return;
  }


  if (buf[0] == 0x04 && buf[1] == 0x82 && buf[2] == 0x91)
     {
    if (buf[3] == 0x00) {
      switch (buf[4]) {
        case 0x03:
        DEBUG_SERIAL.println("\nVOL+ press");
```

```
PressButton(0);
WriteToDisplay(button[0]); //length 8
return;
case 0x43:
DEBUG_SERIAL.println("\nVOL+ hold");
PressButton(1);
WriteToDisplay(button[1]); //length 8
delayWithInterrupt(500);
return;
case 0x04:
DEBUG_SERIAL.println("\nVOL- press");
PressButton(2);
WriteToDisplay(button[2]); //length 8
return;
case 0x44:
DEBUG_SERIAL.println("\nVOL- hold");
PressButton(3);
WriteToDisplay(button[3]); //length 8
delayWithInterrupt(500);
return;
case 0x01:
DEBUG_SERIAL.println("\nSourceR press");
PressButton(4);
WriteToDisplay(button[4]); //length 8
return;
case 0x41:
DEBUG_SERIAL.println("\nSourceR hold");
PressButton(5);
WriteToDisplay(button[5]); //length 8
delayWithInterrupt(500);
return;
case 0x02:
```

```
      DEBUG_SERIAL.println("\nSourceL press");
      PressButton(6);
      WriteToDisplay(button[6]); //length 8
      return;
      case 0x42:
      DEBUG_SERIAL.println("\nSourceL hold");
      PressButton(7);
      WriteToDisplay(button[7]); //length 8
      delayWithInterrupt(500);
      return;
      case 0x05:
      DEBUG_SERIAL.println("\nPause press");
      PressButton(8);
      WriteToDisplay(button[8]); //length 8
      return;
      case 0x00:
      DEBUG_SERIAL.println("\nOK press");
      PressButton(9);
      WriteToDisplay(button[9]); //length 8
      return;
      case 0x40:
      DEBUG_SERIAL.println("\nOK hold");
      PressButton(10);
      WriteToDisplay(button[10]); //length 8
      delayWithInterrupt(500);
      return;
    }
  } else if (buf[3] == 0x01) {
    if (buf[4] == 0x01) {
      DEBUG_SERIAL.println("\nRotary up");
      PressButton(11);
      WriteToDisplay(button[11]); //length 8
```

```
      } else if (buf[4] == 0x41) {
        DEBUG_SERIAL.println("\nRotary down");
        PressButton(12);
        WriteToDisplay(button[12]); //length 8
      }
    }
  } else {
    isLow = false;
    memset(buf, 0xFF, sizeof(buf));
  }
  isLow = false;
  memset(buf, 0xFF, sizeof(buf));
}

void delayWithInterrupt(uint16_t time) {
  uint32_t startTime = millis();
  while (millis() - startTime < time)
  ;
}


void PressButton(byte batun) {
  /*
  1  Volume + press
  2  Volume + hold
  3  Volume - press
  4  Volume - hold
  5  Source right press
  6  Source right hold
  7  Source left press
  8  Source left hold
  9  Pause
```

```
  10 OK press
  11 OK hold
  12 Rotary up
  13 Rotary down
  */
  const byte batunNEC[] = {
    0x14,
    0x14,
    0x15 0x15,
    0x0c,
    0x0c,
    0x0d,
    0x0d,
    0x07,
    0x07,
    0x0b,
    0x0c,
  } IrSender.sendNEC(0xb9, batunNEC[batun]);
  memset(buf, 0xFF, sizeof(buf));
}

void WriteToDisplay(char* msg) {
  return;
  delayMicroseconds(23360);
  DEBUG_SERIAL.println("WriteToDisplay");
  L3_LOW;
  delayMicroseconds(630);
  Wire.beginTransmission(0x23);
  Wire.write(0x0C);
  Wire.write(0x90);
  Wire.write(0x76);
  Wire.write(0x60);
```

```
  Wire.write(0x01);
  Wire.write(msg);
  Wire.endTransmission();
  DEBUG_SERIAL.println("written to disp");
  L3_HIGH;
  previousMillis = currentMillis;
  isLow = false;
  memset(buf, 0xFF, sizeof(buf));
}
```