PROJEKTNI ZADATAK

IZ PREDMETA

PRAKTIKUM AUTOMATIKE

Alarmni sistem za pomoć pri parkiranju u 2D prostoru

Autori:
Anel Mandal
Lejla Mehmedagić

Profesor:
red. prof. dr Samim Konjicija
Asistent:
mr Nedim Osmić, dipl. el. ing.

ELEKTROTEHNIČKI FAKULTET

SARAJEVO

February 16, 2020

Sažetak

U ovom radu prezentirati ćemo rješenje za naš projektni zadatak. Bilo je potrebno na bazi razvojnog sistema PIC16F1939 realizirati sistem koji omogućava pomoć pri parkiranju automobila. Sistem vrši očitavanje trenutne udaljenosti pomoću ultrazvučnih senzora udaljenosti (HS-SR04). Indikacija o udaljenosti od prepreke se šalje pomoću zvučne i svjetlosne indikacije. Što je prepreka blize zvučni signal kao i signalizacija na diodi se oglašava većom frekvencijom. Udaljenost od najbliže prepreke prikazuje se na računaru posredstvom serijske komunikacije. Rješenje će biti prikazano kroz nekoliko poglavlja; poglavlje ”Uvod” sadrži opis problema, ”Pregled korištenih komponenti” opisuje hardverski dio projekta, korištene komponente i njihovu ulogu u sistemu, ”Realizacija projektnog zadatka” sadrži električnu shemu sistema i softverski dio projekta, kod kojim preko mikrokontrolera upravljamo sistemom i opis aplikacije za ispis udaljenosti, ”Zaključak” rezimira značaj teme i diskutuje njeno rješenje i ”Literatura” predstavlja spisak korištene literature.

Abstract

In this paper, we will present a solution to our project assignment. It was necessary to implement, on the basis of the PIC16F1939 development system, a system that provides assistance with car parking. The system reads the current distance using ultrasonic distance sensors (HS-SR04). The obstacle distance indication is sent by sound and light indication. The closer the obstacle is, the audible signal as well as the signaling on the diode are advertised at a higher frequency. The distance from the nearest obstacle is displayed on the computer by serial communication. The solution will be presented through several chapters; chapter ”Uvod” contains a description of the problem, ”Pregled korištenih komponenti” describes the hardware part of the project, the components used and their role in the system, ”Realizacija projektnog zadatka” contains the electrical scheme of the system and the software part of the project, which control the system through the microcontroller, and application description for printing distance, ”Zaključak” summarizes the significance of the topic and discusses its solution and ”Literatura” presents a list of used literature.

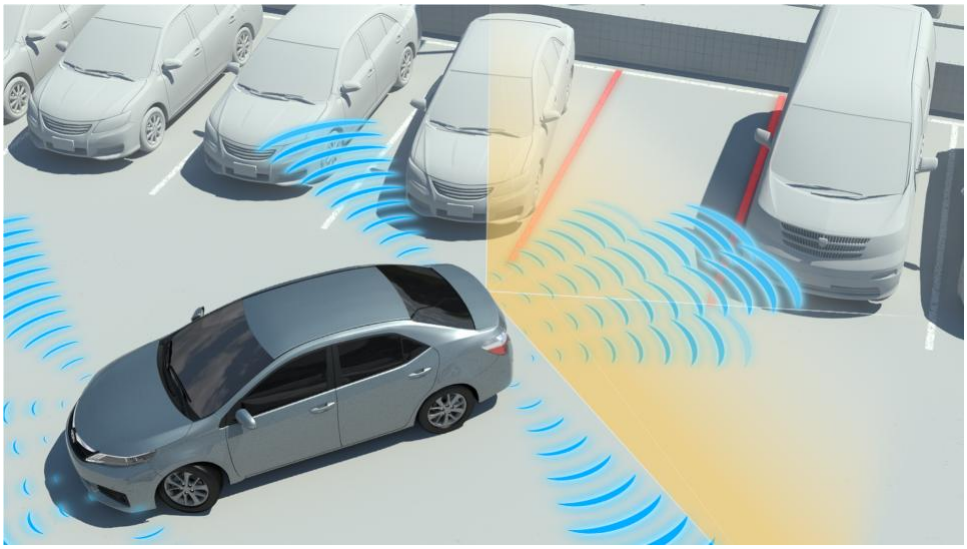
Sadržaj

1. Uvod

Gotovo je nemoguće zamisliti savremene automobile bez funkcionalnosti kao što je alarmni sistem kao pomoć pri parkiranju. U tu svrhu, koriste se sistemi sa ultrazvučnim senzorima udaljenosti, stražnjom kamerom, automatski i drugi. Ultrazvučni senzori se strateški raspoređuju po automobilu kako bi pokrili što širi prostor. Oni emituju ultrazvučni signal koji putuje kroz vazduh i u slučaju prepreke odbija se i vraća ka senzoru koji ga detektuje. U zavisnosti od udaljenosti prepreke, aktivira se alarmni sistem kao upozorenje vozaču.

Neke od prednosti korištenja upravo ove vrste senzora je to što je izlazna vrijednost linearna s udaljenošću između senzora i prepreke, odziv senzora ne ovisi o bojama prepreke, njene prozirnosti, osobina njene optičke refleksije ili njene površinske teksture. Također, veoma su precizni i detektuju čak i male objekte, a mogu raditi i u kritičnim uvjetima kao što su prašina ili prljavština.

S druge strane, nedostaci ovih senzora su to što moraju biti usmjereni na površine velike gustine za dobre rezultate, površine male gustine apsorbiraju zvučne valove koje oni emitiraju, imaju manje vrijeme odziva od drugih senzora, potrebno je upoznati se sa specifikacijama senzora jer mnogi imaju minimalnu udaljenost detekcije, te na ispravne rezultate utiču razni vanjski efekti kao što su temperatura, vlažnost i pritisak.



Slika 1.1: Ultrazvučni sistem parking senzora

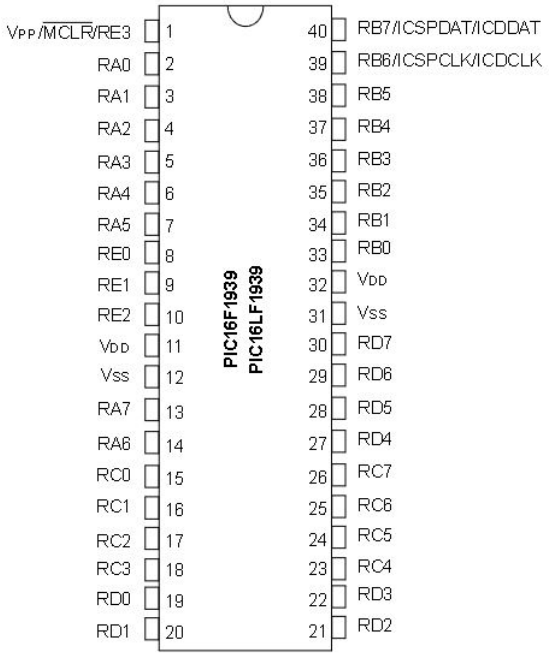
Dakle, cilj ovog projekta je realizirati sistem koji radi na ovom principu. Ultrazvučnim senzorom ćemo mjeriti udaljenost sistema od najbliže prepreke i u ovisnosti o primljenim podacima vršiti svjetlosnu i zvučnu indikaciju čime ćemo simulirati ponašanja sistema parkiranja prikazanog na slici 1.1 [2]. Radi pojednostavljenja, prikazat ćemo rješenje na samo jednom senzoru i aktivirati alarmni sistem **na udaljenosti od 50cm i manje**, što se naravno može modifikovati za realne situacije ali na način da uvijek treba imatu u obzir ograničenja samog senzora.

2. Pregled korištenih komponenti

Projekat ćemo realizovati na razvojnom sistemu na bazi mikrokontrolera **PIC16F1939** koji će izvršavati logiku programa i koji će povezivati sve ostale komponente. Očitavanje trenutne udaljenosti vršimo pomoću ultrazvučnog senzora udaljenosti **HC-SR04**, zvučnu indikaciju putem **YL-44 aktivnog buzzer-a**, a svjetlosnu pomoću zelene i crvene **3mm LED**. Očitanu udaljenost ispisujemo na računaru posredstvom serijske komunikacije za koju nam je potreban komunikacijski modul koji će **konvertovati TTL naponske nivoe** sa mikrokontrolera **u naponske nivoe USB protokola** koje koristi serijski port računara. U ovu svrhu je odabran **TTL-232-3V3-PCB** komunikacijski modul.

2.1. Mikrokontroler PIC16F1939

Mikrokontroler **PIC16F1939** proizvodi firma *Microchip*. Izrađen je u **CMOS** tehnologiji sa ugrađenom FLASH i EEPROM memorijom za spremanje podataka i programa, ima tipičnu RISC arhitekturu sa 49 14-bitnih instrukcija, memorija podataka je podijeljena u 32 banke po 128 bajta u svakoj, te ima hardverski stek sa 16 nivoa. Jezgro mikrokontrolera pakuje se 28-pinsko, 40-pinsko ili 44-pinsko pakovanje, od kojih ćemo mi koristiti 40-pinsko PDIP pakovanje čiji je raspored prikazan na slici ispod.



Slika 2.1: Raspored pinova mikrokontrolera PIC16F1939

Napajanje od +5V se dovodi na pinove V_{DD} (11 i 32), a masa na pinove V_{SS} (12 i 31). Pinovi RA7/OSC1 i RA6/OSC2 služe za priključivanje oscilatorskih komponenti. Pin 1 se koristi kao *reset*, ali i kao pin za dovođenje visokog napona (+12V) u procesu programiranja. Ostala 33 pina predstavljaju U/I linije, grupisani su u pet portova (A-E) i možemo ih konfigurisati kao izlazne ili ulazne.

Od perifernih karakteristika ovog mikrokontrolera možemo izdvojiti A/D konvertor sa 10-bitnom rezolucijom na 14 kanala, pet vremenskih modula od kojih je Timer1 16-bitni, a Timer0/2/4/6 8-bitni, dva standardna i tri poboljšana PWM modula, zatim podrška serijskoj sinhronoj i asinhronoj komunikaciji (EUSART modul), te dva komparatora. Također, mikrokontroler posjeduje ukupno 23 različita izvora prekida koje konfigurišemo pomoću INTCON, PIE1, PIE2 i PIE3 registra. Mi ćemo koristiti prekid vremenskog modula Timer0 i Timer1, zatim prekid na promjenu stanja na PORTB, zbog čega i koristimo pinove RB2-RB6 te EUSART modul.

2.1.1 Timer0 modul

Ovaj modul možemo koristiti i kao tajmer i kao brojač. U oba slučaja radi se o 8-bitnoj vrijednosti u **TMR0 registru** koji možemo čitati i u koji možemo upisivati. Modul posjeduje **3-bitni preskaler** sa programabilnim koeficijentom, moguće je koristiti interni ili vanjski signal, izabrati ivicu vanjskog signala te podržava i prekid na preteku, tj. kada vrijednost TMR0 registra pređe sa *0xFF* na *0x00*. Ova vrijednost se, ukoliko se ne koristi preskaler, povećava svaki instrukcijski ciklus za timer mod koji ćemo koristiti, a može se i manuelno postaviti s tim da se u tom slučaju brojanje zaustavlja sljedeća dva ciklusa.

Programabilni preskaler definira koliko će se instrukcijskih ciklusa brojati kao jedan a upisom vrijednosti u registar TMR0 se njegovo stanje anulira. Omogućava se postavljenjem PSA bita u registru OPTION_REG na nulu. Timer0 modul posjeduje osam opcija za preskaler u opsegu od 1:2 do 1:256. Ove opcije postavljamo pomoću 3 najniža bita OPTION_REG registra (PS biti) kao što je prikazano u sljedećoj tabeli:

PS<2,0>	Stopa Timer0
000	1:2
001	1:4
010	1:8
011	1:16
100	1:32
101	1:64
110	1:128
111	1:256

Tabela 2.1: Postavke preskalera modula Timer0

Prekid ovog modula omogućavamo postavljanjem TMR0IE bita u INTCON registru, zastavica TMR0IF se postavlja pri svakom preteku vrijednosti u registru TMR0 sa 0xFF na 0x00 neovisno od stanja prekida i mora se softverski vratiti na nulu. Naravno, da bismo koristili bilo koji prekid moramo postaviti bit GIE u INTCON registru.

2.1.2 Timer1 modul

Timer1 modul ima slične karakteristike kao i Timer0, jedna od osnovnih razlika je što je sada riječ o 16-bitnom brojaču/tajmeru. Registar TMR1 dolazi u paru od dva registra TMR1H:TMR1L u koji također možemo manuelno upisivati vrijednost što povlači iste posljedice kao kod Timer0 modula. Preskaler ovog modula je dvobitni i dozvoljava opseg od 1:1 do 1:8. Prekid ovog modula omogućavamo postavljanjem sljedeća četiri bita: TMR1ON u T1CON registru, TMR1IE u PIE1 registru, PEIE i GIE u INTCON registru. Da bi se izbjegla nepredvidljiva ponašanja, potrebno je resetovati bit zastavice TMR1IF, te par TMR1H i TMR1L na nulu prije omogućavanja prekida.

2.1.3 Prekid na promjenu PORTB

Pinove na PORTB možemo konfiguristi da operiraju kao IOC (Interrupt-On-Change) pinovi čime možemo generisati prekide na detektovanje uzlazne ili silazne ivice na bilo kojem pinu PORTB. Prekid nije ograničen na samo jedan pin, niti na samo jednu ivicu. Podešavanje ovih prekida ćemo opisati kroz nekoliko osnovnih bita koje koristimo:

- IOCBPx - omogućava(1)/onemogućava(0) prekid na uzlaznu ivicu na pinu x PORTB
- IOCBNx - omogućava(1)/onemogućava(0) prekid na silaznu ivicu na pinu x PORTB
- IOCBFx - zastavica koja indicira(1) da se desio prekid zbog promjene signala na pinu x PORTB
- IOCIF - zastavica koja indicira(1) da se desio prekid zbog promjene signala na nekom pinu PORTB
- IOCIE - omogućava(1)/onemogućava(0) prekid na promjenu PORTB

Vrijednosti svih navedenih bita su 0 po zadanom. Više informacija o ovom i ostalim modulima može se pronaći na oficijelnoj dokumentaciji mikrokontrolera PIC16F1939 [3].

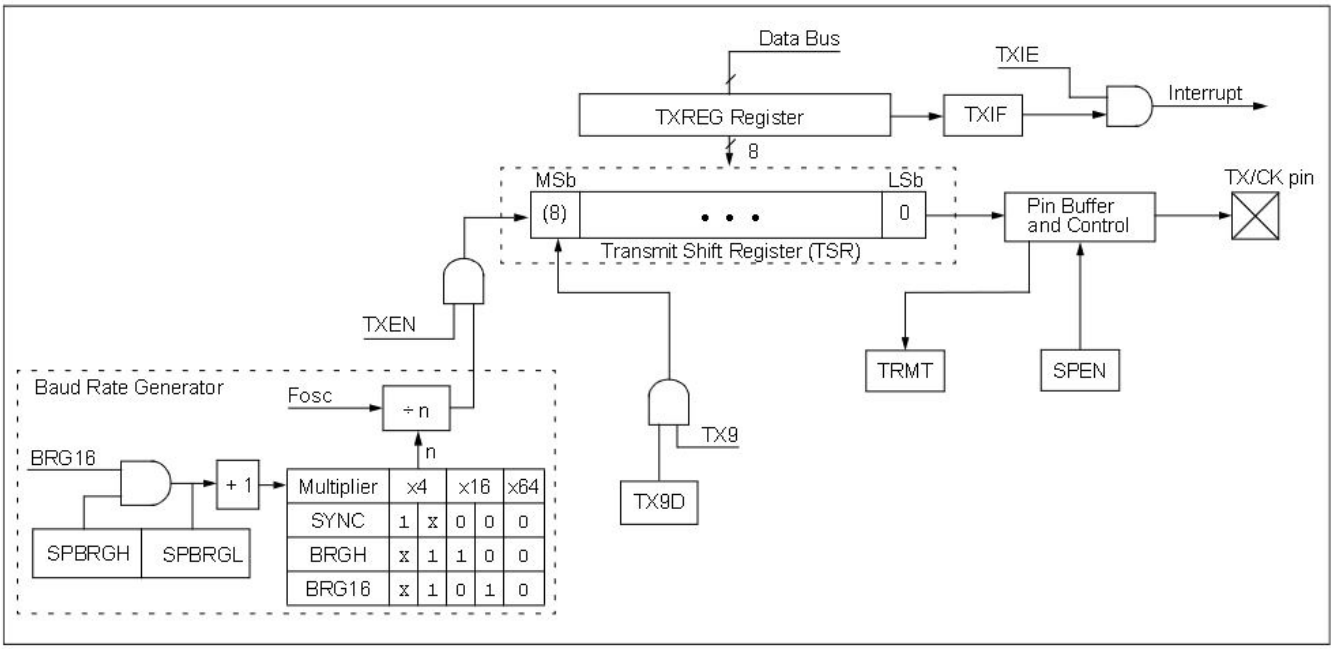
2.1.4 EUSART modul

Razvojni sistem koji koristimo, odnosno mikrokontroler **PIC16F1939**, posjeduje serijski U/I komunikacijski periferni modul pod nazivom **Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART**). Ovaj modul sadrži sve generatore sata, šift registre i bafere podataka potrebne za izvođenje serijskog slanja i primanja podataka neovisno o izvršenju programa. Isti se može konfigurisati kao full-duplex asinhroni ili semi-duplex sinhroni sistem. Kako je potrebno komunicirati sa perifernim uređajem (kao što je terminal, PC i slično), korisno je koristiti **full duplex asinhroni** režim rada. Operacije EUSART modula se kontrolišu putem tri registra:

1. Transmit Status and Control (TXSTA)
2. Receive Status and Control (RCSTA)
3. Baud Rate Control (BAUDCON)

EUSART šalje i prima podatke koristeći standardni **non-return-to-zero (NRZ) format**. NRZ je implementiran sa dva nivoa: V_{OH} stanje koje predstavlja bit 1, i V_{OL} stanje koje predstavlja bit 0. Ovaj format sugeriše da dva uzastopna transmitovana bita iste vrijednosti ostaju na istom nivou tog bita bez vraćanja na neutralni nivo između dvije uzastopne transmisije. NRZ format je u V_{OH} stanju kada je u stanju mirovanja. Svaka transmisija karaktera sastoji se od $Start$ bita nakon kojeg slijedi 8 (ili 9, u slučaju bita pariteta) bita podatka a terminira sa $Stop$ bitom. $Start$ bit je uvijek V_{OL} stanje a $Stop$ bit uvijek V_{OH} stanje. Prvi bit koji EUSART prima i šalje je najmanje značajan bit odnosno LSB. Predajnik i prijemnik su funkcionalno neovisni ali dijele isti format podataka i baud rate. Bit pariteta hardverski nije podržan ali se može implementirati softverski i sačuvati kao deveti bit.

Za ovaj projektni zadatak nije bilo potrebe za prijemnikom EUSART modula ali svakako jeste za predajnikom. Blok dijagrama za EUSART predajnik je prikazan na sljedećoj slici:



Slika 2.2: Blok dijagram EUSART predajnika

Za omogućavanje predajnika EUSART modula za asinhrone operacije, potrebno je konfigurisati sljedeća tri kontrolna bita:

1. $TXEN = 1$ - uključuje predajnički slop EUSART modula kakav je prikazan iznad
2. $SYNC = 0$ - konfigurisanje asinhronog režima rada
3. $SPEN = 1$ - ovaj bit uključuje EUSART modul i automatski konfiguriše TX/CK pin kao izlazni.

Glavni dio predajnika je **Transmit Shift Register (TSR)** kojem nije moguće pristupiti softverski. Ovaj registar dobija podatke od predajničkog bufera koji je zapravo **TXREG registar**. Predaja započinje upisivanjem karaktera u TXREG registar. Ukoliko je ovo prvi karakter, ili je prethodni karakter u potpunosti ispražnjen iz TSR registra, onda se podatak iz TXREG instantno prebacuje u TSR registar. Ako ovo nije slučaj, čeka se transmisija $Stop$ bita prethodnog karaktera. Neposredno nakon popunjavanja TSR registra započinje i transmisija tog karaktera sa odgovarajućim $Start$ i $Stop$ bitima. Veoma je bitan još i bit $TXIF$ registra $PIR1$. Ovaj bit je postavljen kada god je EUSART omogućen i nijedan karakter u TXREG registru nije stavljen na čekanje. Drugim riječima, TXIF bit je resetovan na 0 samo kada je TSR registar zauzet a novi karakter je stavljen na red čekanja u TXREG. TXIF flag bit postaje validan u drugom instrukcijskom ciklusu nakon upisivanja u TXREG registar.

Ostaje još pitanje brzina slanja podatka. Za tu svrhu koristi se **EUSART Baud Rate Generator (BRG)**. Ovaj generator je zapravo 8-bitni ili 16-bitni (zavisno od konfiguracije) tajmer koji pruža sinhrone i asinhrone EUSART operacije. Za svrhu ovog zadatka, baud rade od 9600bps će biti sasvim dovoljan. No, potrebno je vršiti određene kalkulacije kako bi se traženi baud rate dobio sa što manjom greškom. Za navedene kalkulacije koriste se sljedeće formule:

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	FOSC/[64 (n+1)]
0	0	1	8-bit/Asynchronous	FOSC/[16 (n+1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	FOSC/[4 (n+1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGH, SPBRGL register pair

Slika 2.3: Formule za računanje Baud rate-a

Pomoću ovih formula, tražimo takvo n za koje će greška između željenog i dobijenog baud rate-a biti minimalna. Ovo je naravno moguće sve izvesti manuelno, no, kako su isti podaci dostupni u zvaničnoj dokumentaciji mikrontrolera njima ćemo se i poslužiti. Pa tako, ukoliko se odlučimo za 8-bitnu asinhronu komunikaciju sa visokom stopom baud rate-a (BRGH = 1), dobijamo da je minimalna greška, za traženi baud rate od 9600bps, 16% kako je i prikazano na slici 2.4, a postiže se za vrijednost $SPBRG = 51$. Naravno, za ovo je potrebno poznavati frekvenciju oscilatora mikrokontrolera koja je u našem slučaju $8MHz$

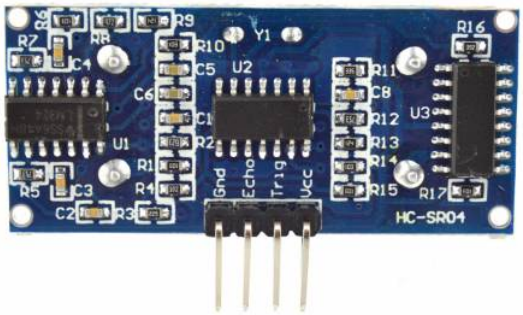
BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 0											
	FOSC = 8.000 MHz			FOSC = 4.000 MHz			FOSC = 3.6864 MHz			FOSC = 1.000 MHz		
	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)
300	—	—	—	—	—	—	—	—	—	300	0.16	207
1200	—	—	—	1202	0.16	207	1200	0.00	191	1202	0.16	51
2400	2404	0.16	207	2404	0.16	103	2400	0.00	95	2404	0.16	25
9600	9615	0.16	51	9615	0.16	25	9600	0.00	23	—	—	—
10417	10417	0.00	47	10417	0.00	23	10473	0.53	21	10417	0.00	5
19.2k	19231	0.16	25	19.23k	0.16	12	19.2k	0.00	11	—	—	—
57.6k	55556	-3.55	8	—	—	—	57.60k	0.00	3	—	—	—
115.2k	—	—	—	—	—	—	115.2k	0.00	1	—	—	—

Slika 2.4: Optimalna vrijednost SPBRG registra za baud rate od 9600bps

2.2. HC-SR04 ultrazvučni senzor

HC-SR04 je popularno i pristupačno rješenje za beskontaktno mjerenje udaljenosti sposobno za mjerenje udaljenosti od **2cm do 400cm** s preciznošću do oko 3mm. Ovaj modul uključuje ultrazvučni transmiter, ultrazvučni prijemnik i njegovo kontrolno kolo.

Slika 2.5: Izgled senzora odozgo



Slika 2.6: Izgled senzora odozdo

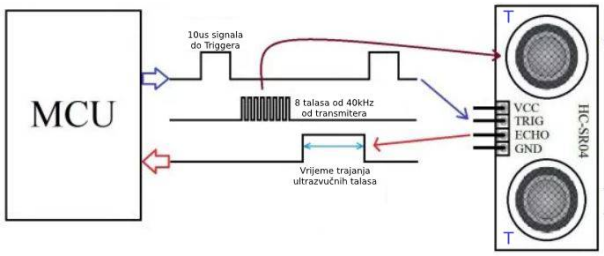
Kao što je prikazano na slici 2.5 modul ima četiri pina od kojih V_{CC} napaja senzor (tipično sa +5V), *Trigger* je ulazni pin koji inicijalizira mjerenje udaljenosti kada je postavljen na jedinicu u trajanju od $10\mu s$, *Echo* je izlazni pin i postavljen je na jedinicu onoliko vremena koliko je potrebno ultrazvučnom talasu da se vrati do senzora i *Ground* je spojen na masu. Rad ovog modula ćemo prikazati u nekoliko koraka:

1. Modul čeka *Trigger* signal, bar $10\mu s$ visokog napona
2. Transmiter senzora generiše zvuk frekvencije $40kHz$ osam uzastopnih ciklusa
3. Ukoliko se prepreka nalazi ispred senzora, na udaljenosti od (teoretski) najviše 4m, zvučni talas se odbija od nazad ka prijemniku senzora
4. Ukoliko se povratni zvučni talasi detektuju, *Echo* će biti u stanju visokog napona onoliko vremena koliko je prošlo od slanja do prijema talasa, ovo vrijeme traje od $150\mu s$ do $25ms$, a ukoliko prepreka nije detektovana, signal će trajati oko $38ms$.

Brzina zvuka u vazduhu (c_z) pri temperaturi od 22°C je konstantna i iznosi oko $343\frac{m}{s}$, a vrijeme nam je poznato jer imamo vrijeme trajanja *Echo* signala, pa udaljenost od prepreke računamo po formuli:

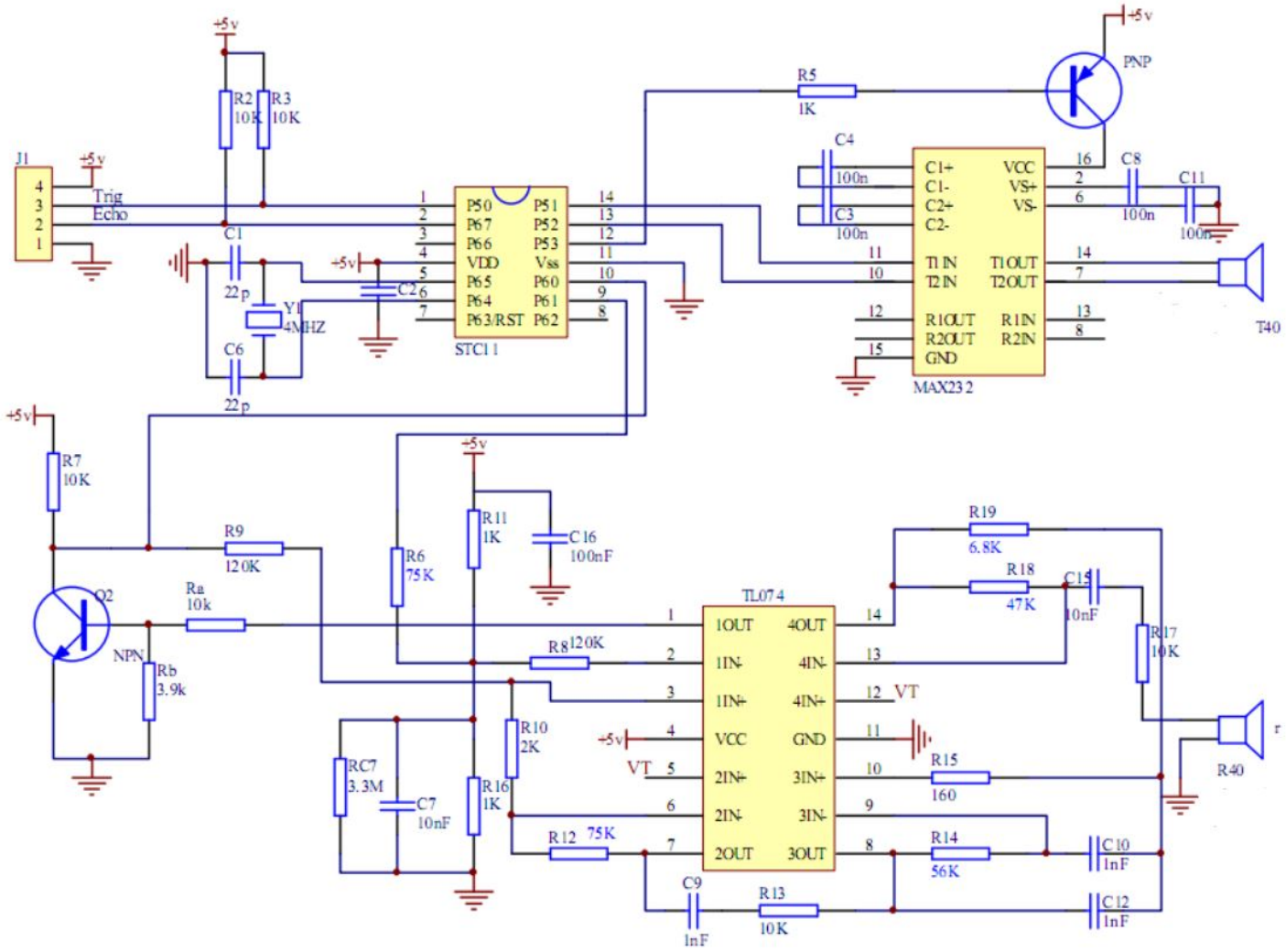
$$udaljenost = \frac{t_{echo}}{2} \cdot c \tag{2.1}$$

Kao što vidimo u relaciji 2.1, vrijeme trajanja *Echo* signala smo podijelili sa 2 jer to vrijeme zapravo predstavlja vrijeme za koje je talas prešao put od transmitera senzora do objekta te od objekta nazad do prijemnika senzora. Sve ovo možemo prikazati i slikovito kao što jeste na slici 2.7.



Slika 2.7: Povezivanje mikrokontrolera sa senzorom

U nastavku slijedi i interna električna shema ovog modula, a za detljniji opis modula predlažemo njegovu dokumentaciju [4] i slične izvore.



Slika 2.8: Shema modula HC-SR04

2.3. LED

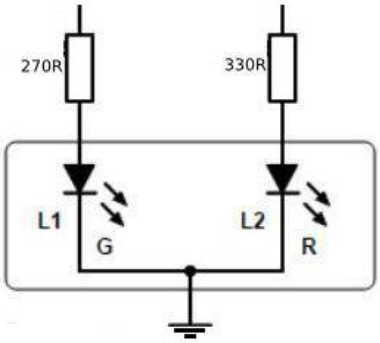
LED (eng. Light Emitting Diode) koje su korištene su crvene i zelene boje, obe su prečnika 3mm i difuznog su tipa. Za odgovarajući odsjaj zahtijevaju **struju od 10mA**, povezane su na pinove na $+5V$ te kao što je i poznato, pad napona na crvenoj LED iznosi $2V$, a na zelenoj LED $2.4V$. Ovim imamo sve potrebne informacije da proračunamo vrijednosti otpora za svaku od LED što prikazujemo u nastavku.

$$R_{red} = \frac{U - U_{red}}{I} = \frac{3V}{10 \cdot 10^{-3}A} = 300\blacksquare \Rightarrow R_{red} = 330\blacksquare \tag{2.2}$$

$$R_{green} = \frac{U - U_{green}}{I} = \frac{2.6V}{10 \cdot 10^{-3}A} = 260\blacksquare \Rightarrow R_{green} = 270\blacksquare \tag{2.3}$$

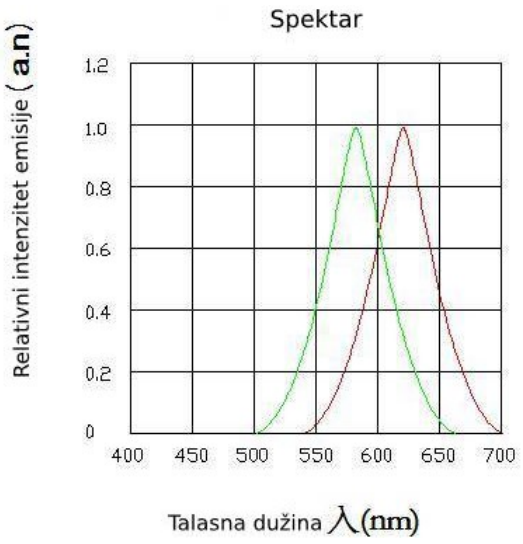
Kako želimo uključivati LED dovođenjem logičke jedinice, spajamo ih sa zajedničkom katodom, a anode spajamo na odgovarajuće pinove.

Slika 2.9: Zelena i crvena LED



Slika 2.10: Shema LED

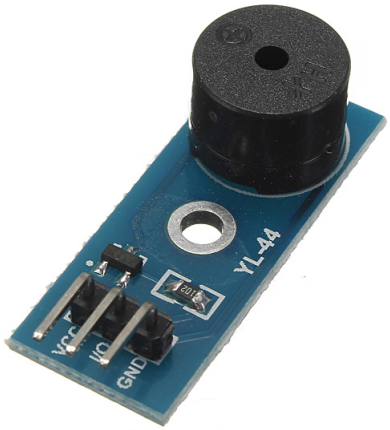
Zelena LED indicira da je najbliža prepreka na sigurnoj udaljenosti, tj. dalje od 50cm, a crvena suprotno. Prikazat ćemo još talasne dužine crvene i zelene svjetlosti kako bismo dodatno ukazali na njihove razlike što opravdava izbor različitog otpornika.



Slika 2.11: Talasne dužine crvene i zelene svjetlosti

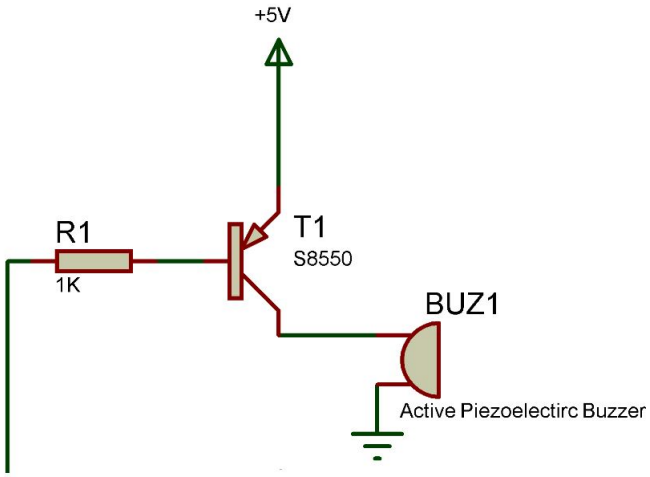
2.4. YL-44 aktivni buzzer modul

Za zvučnu signalizaciju korišten je **aktivni piezoelektrični buzzer**. Aktivni buzzer posjeduje interni oscilator stoga je za zvučni signal potrebna samo DC voltaža. Ukoliko se pak odluči za pasivni buzzer tada je potrebno dovesti signale četvrtke željene frekvencije (u intervalu od 2kHz - 5kHz).



Slika 2.12: YL-44 aktivni buzzer modul

Naravno, nije neophodno koristiti gotov modul kao što je YL-44, prikazan na slici 2.12, već je moguće koristiti samo piezoelektrični buzzer kakav jeste te tranzistor koji će služiti kao prekidač. Dodatno, na bazu tranzistora je i otpornik od $1K$ kako ne bi došlo do oštećenja poluprovodnika. S tim na umu, YL-44 se može predstaviti sljedećom shemom:



Slika 2.13: Shema YL-44 buzzer modula

Modul se koristi na sljedeći način:

1. Na Vcc pin dovodimo napon mikrokontrolera (u našem slučaju $+5V$)
2. Gnd spajamo sa masom
3. Pin I/O služi za aktiviranje buzzera. Kako se radi o PNP tranzistoru, ukoliko se na ovaj pin dovede logička 0 (odnosno 0V), ili ukoliko pin uopšte nije povezan, onda buzzer generiše zvuk.

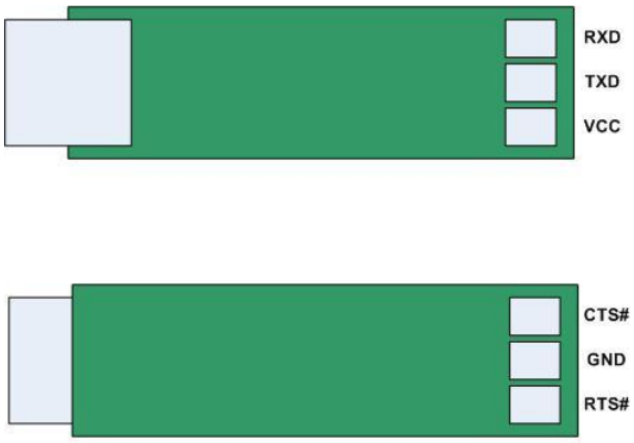
2.5. TTL-232-3V3-PCB kovertor

TTL-232R-PCB je konverter TTL serijskog UART u USB komunikacijski standard. Na njemu se nalazi integrisani čip **FT232RQ** koji obrađuje sve USB signalizacije i protokole. Ovaj PCB pruža brz i jednostavan način za konektovanje uređaja koji rade sa **TTL nivoima serijskog interfejsa na USB konektor A tipa** (isti je u potpunosti kompatibilan sa USB 2.0 standardom).

Postoje dva tipa:

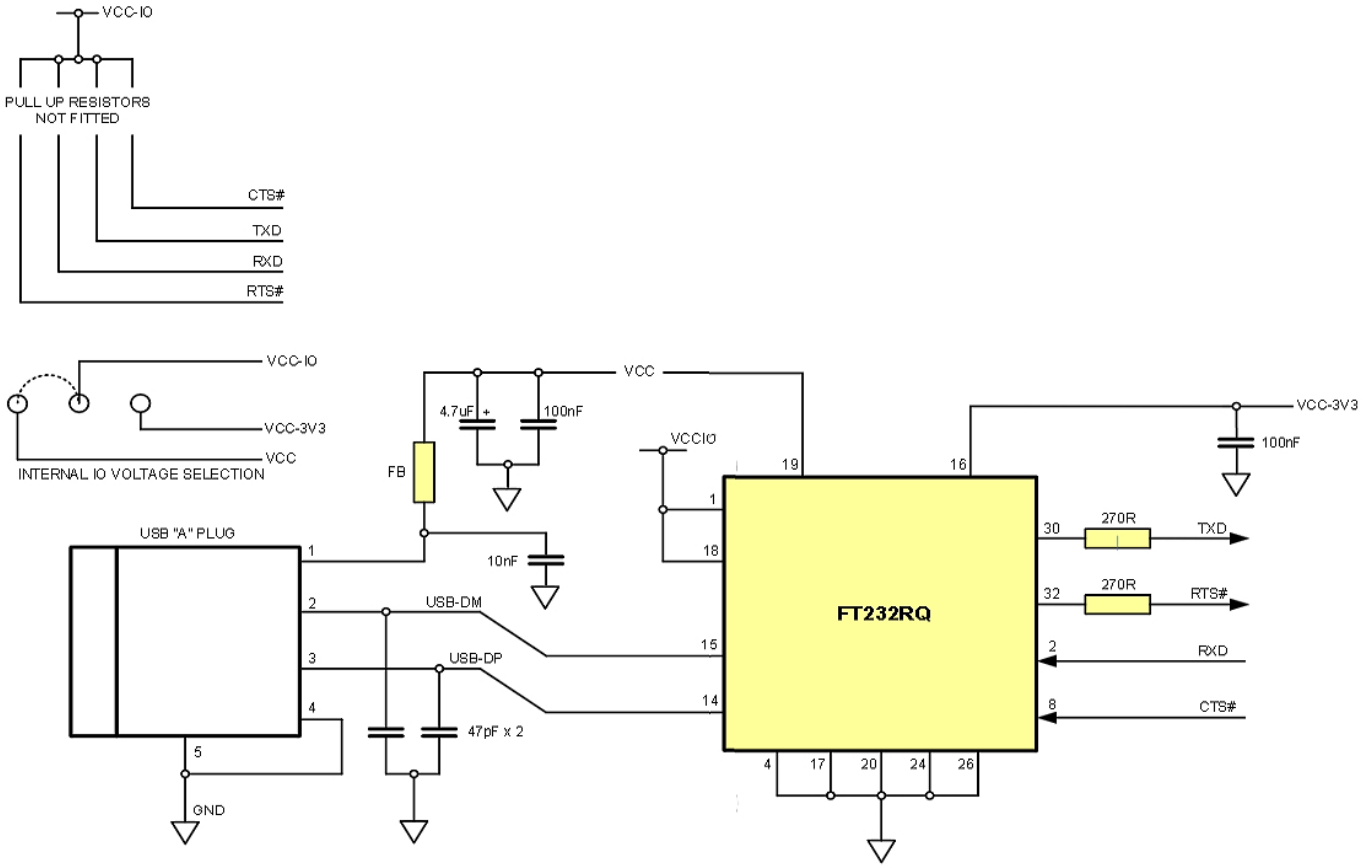
1. TTL-232R-5V-PCB USB to UART PCB sa selektivnim +5V TTL nivoima UART signala.
2. TTL-232R-3V3-PCB USB to UART PCB sa selektivnim +3.3V TTL nivoima UART signala. Ovaj je korišten pri izradi projekta.

Dakle, ovaj PCB će nam omogućiti uspostavljanje interfejsa između UARTA-a (ili u našem slučaju, EUSART-a) našeg mikrokontrolera sa USB što će nam znatno olakšati PIC to PC komunikaciju. Raspored pinova ovog PCB-a možemo vidjeti na sljedećoj slici:



Slika 2.14: Raspored pinova TTL-232R-PCB

Dodatno, s ciljem boljeg razumijevanja , prikazujemo i shemu ovog PCB-a. Pošto je integrisani čip **FT232RQ** na slici prikazan na nešto apstraktniji način, navodimo i dokumentaciju[7] za isti.

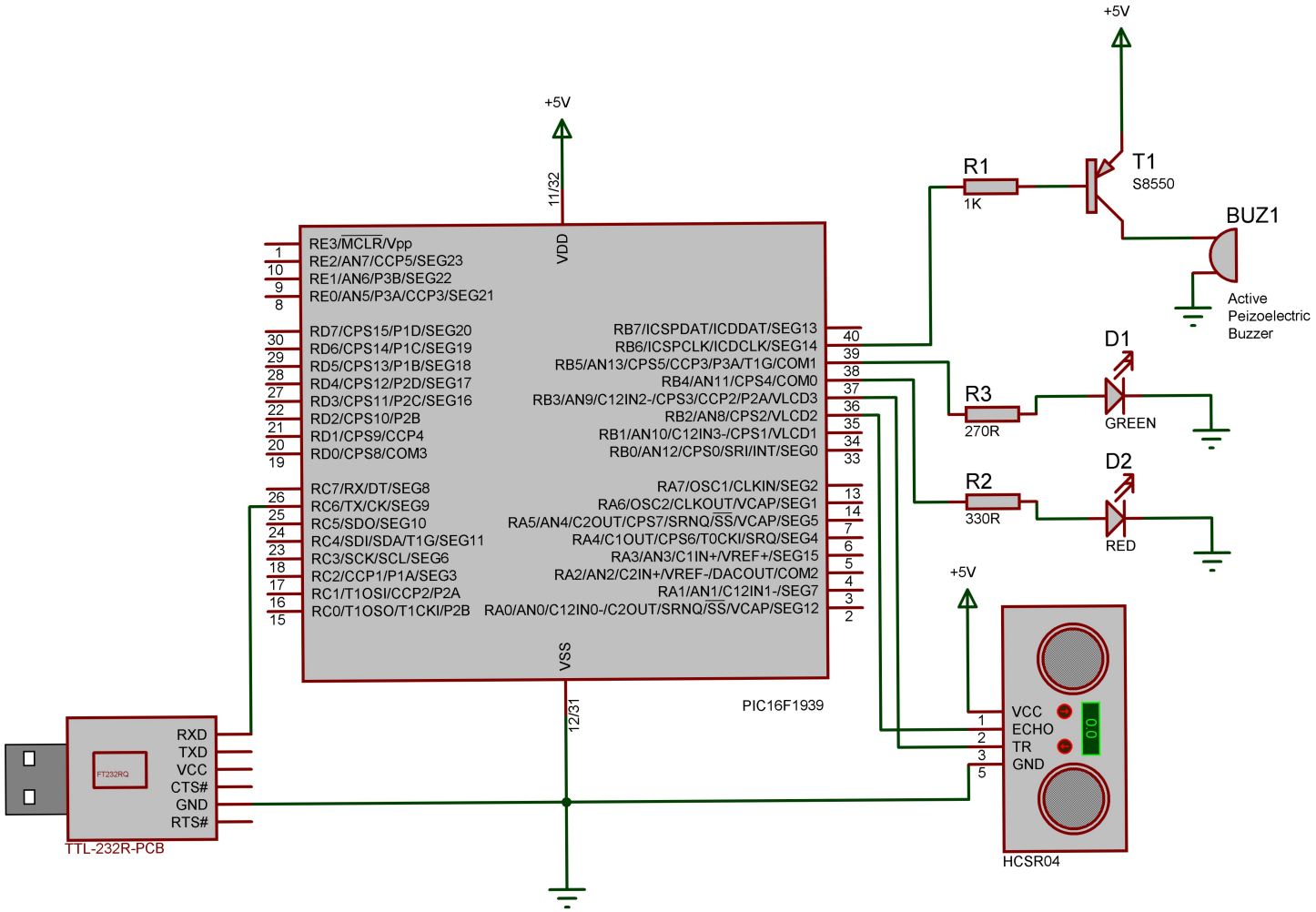


Slika 2.15: TTL-232R-PCB shema

3. Realizacija projektnog zadatka

3.1. Električna shema

Nakon što smo opisali korištene komponente potrebne za realizaciju ovog projekta te načine njihovog ispravnog povezivanja sa razvojnim sistemom mikrokontrolera PIC16F1939, u nastavku prilažemo i shemu:



Slika 3.1: Električna shema alarmnog sistema

3.2. Analiza programskog koda

Analizu koda vršiti ćemo na inkrementalni način, tj. postepeno ćemo objašnjavati dijelove izvornog koda. Za početak, koristeći direktivu **#define** definisali smo makroe kako bi sebi olakšali rad sa komponentama. Drugim riječima, 'reimenovali' smo pinove kao što su RB2, RB3 i ostale na način da njihovo ime intuitivnije objašnjava njihovu ulogu. Odmah ispod nalaze se 3 globalne varijable koje se frekventno ažuiraju a koje služe prilikom signalizacije i komunikacije (o njima će više riječi biti kasnije).

```
#define _XTAL_FREQ 8000000

#define echo RB2     // Echo pin of HC-SR04
#define trigger RB3 // Trigger pin of HC-SR04
#define LED_RED RB4
#define LED_GREEN RB5
#define buzzer RB6 // I/O pin of YL-44 active buzzer module

int old_centimeters = 0; // To avoid printing unnecessary data
int delay = -1, counter = 0;
__bit measureAgain = 0;
```

Nakon toga, slijedi prototip i implementacija funkcija koje se pozivaju na samom početku izvršenja programa (prve tri linije main funkcije, kao što ćemo i vidjeti) a koje služe za inicijalizaciju svih neophodnih modula i pinova.

```
void initialize_ports(){
    ANSELB = 0;
    TRISB = 0b10000111;
    TRISCbits.TRISC6 = 0; // Although not necessary due to SPEN = 1

    buzzer = 1;
    LED_RED = 0;
    LED_GREEN = 1;
}
```

Kao što možemo vidjeti, **PORTB** je konfigurisan na način da su pinovi **RB3-RB6** digitalni izlazi. Osim ovih pinova koristi se i pin **RB2** ali ovaj pin je digitalni ulaz jer je povezan sa *Echo* pinom **HC-SR04** modula. Razlog za ovo je korištenje **prekida na promjenu** pina **RB2** zbog već navedenog načina rada *Echo* pina.

```
1  void initialize_euart(){
2      // Baud rate = 9600bps
3      BRG16 = 0;
4      BRGH = 1;
5      SPBRG = 51;
6      SYNC = 0;
7      SPEN = 1;
8      RCIE = 0;
9      TXEN = 1;
10 }
```

Mada neke linije koda (3, 6 i 8) unutar ove funkcije nisu bile potrebne (jer su po zadanom vrijednosti već takve), one su napisane čisto radi bolje predstave. Kako je već rečeno, za traženi baud rate od 9600bps najmanje greška iznosi za konfiguraciju kakva je prikazana iznad. Osim toga, kako se radi o slanju informacija od mikrokontrolera ka PC-u, potrebno je omogućiti i predajnički sklop EUSART modula što se postiže linijama **SPEN = 1** i **TXEN = 1**.

```
void initialize_interrupts(){
    // IOC on RB2
    IOCBP2 = 1; // Detect a rising edge
    IOCBN2 = 1; // Detect a falling edge
    IOCBF2 = 0; // Clear status flags
    IOCIF = 0;  // Reset IOC flag
    IOCIE = 1;  // Enable Interrupt-On-Change

    // Timer0 for LED and Active Buzzer

    // Timer0 Clock Source Select bit (0 - Internal)
    OPTION_REGbits.TMR0CS = 0;
    // Timer0 Source Edge Select bit (0 - Rising edge)
    OPTION_REGbits.TMR0SE = 0;
    // Prescaler Assignment bit (0 - Prescaler is assigned)
    OPTION_REGbits.PSA = 0;
    // Prescaler Rate Select bits - 1:256 Prescale value
    OPTION_REGbits.PS = 7;

    TMR0IE = 1; // Enable Timer0 interrupt
    TMR0IF = 0; // Reset Timer0 flag

    // Timer1
    T1CONbits.TMR1CS = 0; // Timer1 Clock Source is instruction clock (Fosc/4)
    T1CONbits.T1CKPS = 1; // 1:2 Prescale Value
    TMR1ON = 0;           // Do not start Timer1 yet
    TMR1IE = 1;           // Enable Timer1 interrupt
    TMR1IF = 0;           // Reset Timer1 flag
    TMR1 = 0;             // Clear TMR1 Register

    PEIE = 1; // Set the peripheral interrupt enable bit
    GIE = 1;  // Set the global interrupt enable bit
}
```

Dakle, prvo konfigurišemo prekid i na uzlaznu i na silaznu ivicu pina **RB2**. Kako je potrebno računati vrijeme za koje je vrijednost **RB2** bila logička jedinica, potrebno je znati kada je došlo do promjene signala sa **LOW-HIGH** kao i promjene **HIGH-LOW**. Način na koji smo računali vrijeme između ove dvije promjene je putem Timer1 modula što ćemo vidjeti uskoro.

Nakon toga, slijedi inicijalizacija Timer0 modula. Ovaj modul je konfigurisan tako da radi sa internim satom instrukcijskog ciklusa uz preskaler 1:256. Analogna konfiguracija se radi i sa Timer1 samo što ćemo za isti koristiti preskaler 1:2.

Naravno, za oba tajmera omogućavamo i prekide. Osim toga, zbog Timer1 modula (koji spada u periferne prekide) potrebno je i postaviti $PEIE$ bit. U suštini, kako je čitav kod pokrijepljen komentarima nema potrebe za daljnjim, detaljnijim objašnjenjem.

Pogledajmo sada najbitniji dio programa, prekidnu rutinu.

```
void __interrupt() handle_interrupt(void){

    if (TMR0IE && TMR0IF){
        TMR0IF = 0;
        counter++;
        if (counter >= delay && delay != -1){
            counter = 0;
            LED_RED = 1 - LED_RED;
            buzzer = 1 - buzzer;
        }
    }

    if (IOCIE && IOCIF && IOCBF2){
        IOCIF = 0;
        IOCBF2 = 0;

        if (echo){
            // Start TIMER1
            TMR1 = 0;
            TMR1ON = 1;
        }
        else {
            // Stop TIMER1
            TMR1ON = 0;
            old_centimeters = display_distance(0.01715 * TMR1);
            measureAgain = 1;
        }
    }
}
```

Najbolje je započeti sa drugom **if naredbom**. Ova naredba služi za detekciju promjene stanja na pinu **RB2**. No, kako je moguća detekcija i na uzlaznu i silaznu ivicu, potrebno je dodatno još ispitati koja je promjena u pitanju.

Kako znamo da će **HC-SR04** modul neposredno nakon primanja **HIGH** impulsa na *Trigger* pinu u periodu od $10\mu sec$ započeti sa transmisijom zvučnog signala frekvencije $40kHz$, u istom tom trenutku će se na *Echo* pinu pojaviti $HIGH$ stanje (odnosno, pin **RB2** dobija vrijednost logičke jedinice što će dovesti do detekcije uzlazne ivice). Upravo za tu svrhu služi prva **if naredba** koja ispituje da li je pinu **RB2** logička jedinica. Ukoliko jeste, to nam je indikacija da treba započeti mjerenje dužine impulsa koji će ponuditi *Echo* pin. Zaustavljanje vremena se vrši u **else naredbi** kada znamo da je došlo do detekcije silazne ivice, gdje onda zaustavljamo Timer1. Naime, kada zaustavimo Timer1 registar TMR1 posjeduje neku vrijednost.

Postavlja se pitanje kako tu vrijednost iskoristiti za računanje vremenja za koje je pin **RB2** (odnosno *Echo* pin) imao vrijednost logičke jedinice. Odgovor leži u načinu kako je Timer1 modul konfigurisan. Kako se vrijednost registra TMR1 inkrementira svaki drugi instrukcijski ciklus, zbog korištenja preskalera, onda znamo da vrijednost tog registra ne predstavlja ništa drugo nego broj instrukcijskih ciklusa koliko je izvršeno dok je $Echo/RB2$ pin imao vrijednost logičke jedinice. Kako znamo da za naš mikrokontroler vrijedi da je frekvencija oscilatora $f_{osc} = 8MHz$ odnosno $T_{osc} = 0.125\mu sec$, shodno tome kao i formuli $T_i = 4 \cdot T_{osc}$ dobijamo da je vrijeme izvršavanja instrukcijskog ciklusa $T_i = 4 \cdot 0.125\mu sec = 0.5\mu sec$. Ovo možemo sada iskoristiti kako bi dobili tačno vrijeme koje je proteklo između dvije uzastopne promjene stanja na pinu **RB2**. Na primjer, ukoliko je pri zaustavljanju Timer1 modula TMR1 registar imao vrijednost 256, onda ukoliko taj broj pomonožimo sa vremenom trajanja dva instrukcijskog ciklusa dobijamo vrijeme $256 \cdot 2 \cdot 0.5\mu sec = 256\mu sec$.

Udaljenost do najbliže prepreke sada možemo dobiti na sljedeći način. Poznata nam je osnovna formula koja povezuje vrijeme, udaljenost i brzinu:

$$Udaljenost = Brzina \cdot Vrijeme \tag{3.1}$$

Kao što je već rečeno, vrijeme je moguće dobiti putem **TMR1** registra, pa shodno tome vrijedi:

$$Vrijeme = TMR1 \cdot 2 \cdot 0.5\mu s \tag{3.2}$$

Vrijeme je potrebno još dodatno podijeliti i sa 2 jer nas zanima samo vrijeme povratnog puta, odnosno vrijeme od prepreke ka prijemniku senzora. Za brzinu zvuka prihvatićemo vrijednost

$$Brzina = c_z = 343\frac{m}{s} = 0.0343\frac{cm}{\mu s} \tag{3.3}$$

Uvrštavajući sve navedeno dobijamo:

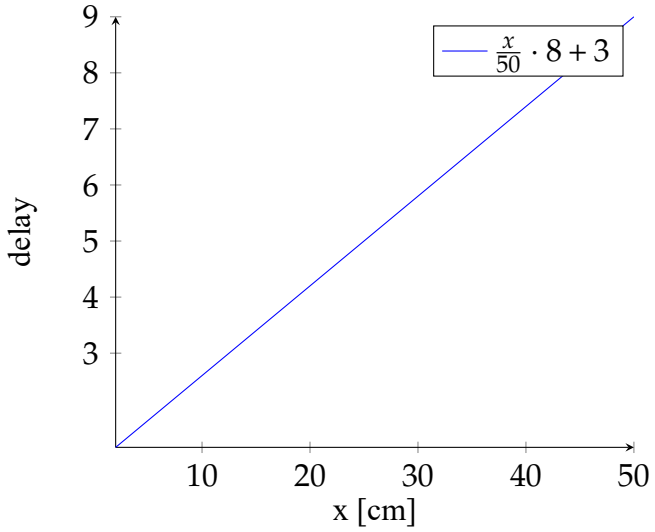
$$Udaljenost = 0.0343\frac{cm}{\mu s} \cdot \frac{TMR1 \cdot 0.5 \cdot 2\mu s}{2} = 0.01715 \cdot TMR1 \tag{3.4}$$

Ovakva proračuna vrijednost se zatim prosljeđuje funkciji *display_distance*.

```
void euart_write(char *text){
    GIE = 0; // Pause all interrupts
    char i;
    for (i = 0; text[i] != '\0'; i++){
        TXREG = text[i];
        while (!TXIF); // Polling TSR register
    }
    GIE = 1; // Enable interrupts again
}

int display_distance(float newCentimeters){
    trigger = 0;

    if (newCentimeters > 2 && (int)newCentimeters != old_centimeters){
        char str[10];
        sprintf(str, ",%d", (int)newCentimeters);
        euart_write(str);
    }

    if (newCentimeters > 2 && newCentimeters <= 50){
        LED_GREEN = 0;
        delay = (char)(newCentimeters / 50.0 * 8 + 3);
    }
    else{
        LED_RED = 0;
        LED_GREEN = 1;
        buzzer = 1;
        delay = -1;
    }

    return newCentimeters;
}
```

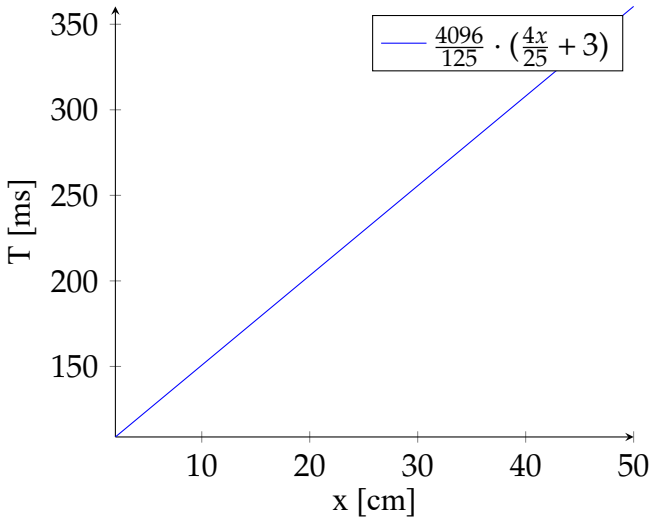
Ukoliko je proračunata udaljenost **veća od 2cm** i razlikuje se od prethodno izračunate udaljenosti (to je upravo svrha varijable *old_centimeters*) onda se poziva funkcija **euart_write**. Ova funkcija privremeno pauzira sve prekide kako ne bi došlo do nepredvidivih ishoda prilikom slanja podataka od mikrontrolera ka računaru. U registar **TXREG** se upisuje karakter po karakter, a prilikom svakog novog upisa čeka se prethodno oslobađanje **TSR** registra kako je to i već objašnjeno.

Do zvučne i svjetlosne indikacije dolazi ukoliko je udaljenost **manja od 50cm**. Naravno, ovo je moguće modifikovati za realne potrebe bez ikakvih problema. Dakle, ukoliko je udaljenost **veća od 2cm** (što su otprilike i sama ograničenja ovog ultrazvučnog senzora) a **manja od 50cm** dolazi do preračunavanja frekvencije kojom će se signalizirati buzzer i crvena LED. Relacija između udaljenosti i kašnjenja kojim odgađamo prekidnu rutinu Timer0 modula jeste sljedeća:



Slika 3.2: Veza udaljenosti i kašnjenja kojim odgađamo prekidnu rutinu Timer0 modula

Da bi bolje shvatili ulogu varijable **delay** potrebno je vratiti se na prekidnu rutinu Timer0 modula. Kako se koristi 1:256 preskaler, to nam govori da će do inkrementiranja TMR0 registra doći nakon 256 detekcija uzlazne ivice sata instrukcijskog ciklusa. S tim na umu, ovako konfigurisan modul daje maksimalno vrijeme od $256 \cdot 256 \cdot 0.5\mu s = 32768\mu s = 32.768ms$. Recimo sada da je $delay = 8$, što se može dobiti ukoliko je udaljenost recimo, $36cm$. Tada će se prekidna rutina (tj. njen glavni dio, odnosno dio unutar **if naredbe**) Timer0 modula dodatno odgoditi još 8 puta, što će reći da će se ista izvršiti tek nakon $32.768\mu s \cdot 6 = 262.144\mu s$. Možemo onda i direktno napisati vezu između proračunatih centimetara udaljenosti do prepreke i perioda između dvije uzastopne signalizacije buzzera i crvene LED.



Slika 3.3: Veza udaljenosti i vremena između dvije uzastopne signalizacije buzzera i crvene LED

Vratimo se sada ponovo na 25. liniju koda prekidne rutine. Novu proračunatu vrijednost udaljenosti u centimetrima ćemo dodijeliti varijabli **old_centimeters** iz već navedenih razloga. Na kraju, postavljamo i bit **measureAgain = 1** koji signalizira da je ponovo potrebno slati impuls na *Trigger* pin u periodu od $10\mu s$ kako bi započelo novo očitavanje.

Postavlja se još i pitanje da li je moguće, zbog načina na koji se vrši računanje udaljenosti, da TMR1 registar dostigne overflow? Odgovor je negativan. Naime, kako se radi o 16-bitnom registru onda najveća vrijednost koja može stati u ovom registru je 65535. Ultrazvučni senzor koji se koristi, ukoliko ne detektuje nikakvu prepreku koja mu je na udaljenosti od (teoretski) najviše 4m, će na *Echo* pinu dati impuls dužine 38ms, što je jednako $38000\mu s$. Iz ovog razloga je i korišten preskaler, pa se u registru TMR1 zapravo smješta vrijeme u mikrosekundama, koje nikada neće preći vrijednost 65535.

Pogledajmo i funkciju *measure_distance* u kojoj postavljenjem HIGH impulsa na Trigger pin započinjemo transmisiju zvučnog signala frekvencije 40kHz sa HC-SR04 modula. Ova transmisija će trajati $10\mu s$ nakon čega je zaustavljamo, te čekamo da se opet uđe u prekidnu rutinu odakle ispisujemo udaljenost.

```
1 void measure_distance(void){
2     // Initiate distance measuring again
3     TMR1 = 0;
4     TMR1ON = 1;
5     trigger = 1;
6     while (TMR1 <= 10); // 10 microseconds impulse on Trigger pin
7     trigger = 0;
8     TMR1ON = 0;
9 }
```

Preostaje nam još funkcija *main* koja se poziva na početku programa. Njene prve tri linije služe za inicijalizaciju portova, komunikacijskog modula i prekida. Zatim se zaustavljamo u petlji koja nakon svakog ispisa iznova poziva funkciju za mjerenje udaljenosti.

```
1  void main(void){
2      initialize_ports();
3      initialize_euart();
4      initialize_interrupts();
5
6      while (1){
7          if (measureAgain){
8              measureAgain = 0;
9              measure_distance();
10         }
11     }
12 }
```

3.3. Java aplikacija za ispisivanje udaljenosti

U svrhu čitanja podataka koji se šalju od mikrokontrolera ka računaru svakako je moguće koristiti neku od aplikacija koja već pruža funkcionalnost konektovanja na serijski port kao što je recimo **PuTTy**. No, kako bi projekat bio kompletniji, a i kako bi se stekao bolji uvid o tome šta je sve potrebno da bi se traženi podaci čitali sa serijskog porta, napravljena je jednostavna **GUI aplikacija u programskom jeziku Java**. Ovdje ćemo opisati samo fundamentalne stvari vezane za ovu aplikaciju jer, zaista, programiranje u Javi nije bilo prioritet tokom izrade ovog projektnog zadatka.

Kroz programsko okruženje **IntelliJ IDEA**, kao i biblioteku **JavaFX** za kreiranje grafičkih interfeja u **programskom jeziku Java**, kreirana je aplikacija sljedećeg izgleda:

Slika 3.4: Java aplikacija za čitanje serijskog porta

Kao što je i vidljivo, aplikacija je veoma intuitivna. Sve što radi jeste da prikazuje udaljenost do najbliže prepreke od senzora. Kako fokus definitivno nije na grafičkom dijelu aplikacije, osvrnuti ćemo se samo na način konektovanja sa serijskim portom.

Programski jezik **Java**, između mnogih ostalih, pruža i biblioteke za komunikaciju sa serijskim portom. Jedna od njih je i biblioteka **jSerialComm**. Importovanjem ove bilbioteke u projekat, moguće je kreirati instance klase **SerialPort**. Ova klasa pruža jako širok spektar funkcionalnosti kada je u pitanju komunikacija sa serijskim portom. Pogledajmo, za početak, sljedeću statičku metodu unutar ove klase a koja vraća instancu/objekat te klase:

```
1 getCommPort(String portDescriptor)
```

Ovom metodom moguće je dobiti instancu **SerialPort** koja je *vezana* za serijski port čije je ime proslijeđeno kao parametar. Na primjer, jedan način da se ova metoda pozove jeste:

```
1 SerialPort requiredComPort = SerialPort.getCommPort("COM3");
```

Sada nakon što smo dobili traženu *vezu*, moguće je manuelno **konfigurisati port** kako bi ga prilagodili za komunikaciju sa mikrokontrolerom. Recimo, za svrhu ovog zadatka korištena je sljedeća konfiguracija:

```
1 requiredComPort.setBaudRate(9600);
2 requiredComPort.setNumDataBits(8);
3 requiredComPort.setNumStopBits(SerialPort.ONE_STOP_BIT);
4 requiredComPort.setParity(SerialPort.NO_PARITY);
```

Tek onda kada se izvrši željena konfiguracija, port se može *otvoriti*.

```
1 requiredComPort.openPort();
```

Sada se postavlja pitanje kako ispravno čitati podatke koje se šalju na traženi **COM port**. U tu svrhu, biblioteka **jSerialComm** nudi 4 načina:

1. Neblokirajuće čitanje (Polling)
2. Blokirajuće i polu-blokirajuće čitanje
3. Koristeći standardne Java biblioteke za interakciju sa serijskim portom
4. **Čitanje zasnovano na događajima**

Ukoliko želimo da se čitanje serijskog porta zasniva na kompletno asinhronoj metodologiji, onda je preporučljivo koristiti 4. metod. Koristeći funkciju **addDataListener** koja predstavlja **callback funkciju** moguće je konfigurisati šta tačno žeimo da program radi ukoliko dođe do specifičnog događaja vezanog za serijski port. S tim na umu, koristeći ovu biblioteku moguće je, između ostalog, 'prisluškivati' i na događaj koji je aktuelan kada su **podaci spremni za čitanje**. Pogledajmo sljedeći isječak koda:

```
1  // Event Based Reading with jSerialComm
2  requiredComPort.addDataListener(new SerialPortDataListener() {
3      @Override
4      public int getListeningEvents() {
5          return SerialPort.LISTENING_EVENT_DATA_AVAILABLE; // Data Available for
       Reading
6      }
7
8      // This callback will be triggered whenever there is any data available to be
       read over the serial port
9      @Override
10     public void serialEvent(SerialPortEvent event) {
11         if (event.getEventType() != SerialPort.LISTENING_EVENT_DATA_AVAILABLE)
12             return;
13
14         byte[] newData = new byte[requiredComPort.bytesAvailable()];
15         requiredComPort.readBytes(newData, newData.length);
16
17         // Print read distance
18         System.out.println("Distance is: " + new String(newData));
19     }
20 });
```

Na svaki događaj (**SerialPort.LISTENING_EVENT_DATA_AVAILABLE**) poziva se funkcija **serialEvent** unutar koje korisnik piše kod koji želi da se tada izvrši.

Bitno je naglasiti da je brzina kojom mikrokontroler šalje podatke računaru, za potrebe ove aplikacije, u većini slučajeva prebrza. Dakle, ne postoji garancija da će ispisivanje udaljenosti na aplikaciji uvijek biti ispravno. Razlog je i prilično jasan. S obzirom da se koriste asinhroni pozivi, sasvim je prirodno očekivati da će doći do **stanja natjecanja** prilikom ažuriranja udaljenosti na aplikaciji. Uostalom, takvo brzo ažuriranje ne bi imalo ni smisla jer bi frekvencija ažuriranja bila toliko velika da krajnji korisnik ne bi mogao ni zaključiti koja mu se trenutna udaljenost na aplikaciji prikazuje. Rješenje je uvesti neki vid periodičnog pozivanja, pri čemu je vrijeme između dva uzastopna poziva podesivo. Ovakvu funkcionalnost nudi biblioteka **Timer**. Koristeći ovu biblioteku, ažuriranje grafičkog interfejsa je podešeno da se izvršava svakih 350 milisekundi.

```
TimerTask timerTask = new TimerTask() {
    @Override
    public void run() {
        try {
          // Change according to your needs, for example:
          mainController.updateDistance(currentDistance);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
};

Timer timer = new Timer();
timer.schedule(timerTask, 0, 350); // Update the GUI every 300 miliseconds.
```

4. Zaključak

Cilj rada je bio napraviti sistem koji mjeri udaljenost od najbliže prepreke i ispisuje je na računar putem serijske komunikacije. Rad je uspješno ostvaren i podijeljen je na dva dijela, hardverski i softverski.

U hardverskom dijelu opisane su sve komponente neophodne za realizaciju ovog projektnog zadatka. Glavna komponenta je ultrazvučni senzor udaljenosti, HC-SR04. Idealan je zbog svoje velike rasprostranjenosti, preciznosti ali i niske cijene. Da bi senzor pravilno mjerio udaljenost, potreban je i mikrokontroler koji će njime upravljati. Korišten je mikrokontroler PIC16F1939, proizvođača Microchip. Za indikaciju udaljenosti korišeni su LED i aktivni buzzer.

Softverski dio se najviše odnosi na programsku podršku, odnosno kōd kojim se preko mikrokontrolera upravlja cijelim sistemom. Kroz kod se pokreće transmisija ultrazvučnog signala pomoću *Trigger* pina, zatim definišu prekidne rutine od kojih je najbitnija ona koja se aktivira dok je *Echo* pin u stanju visokog napona. Vrijeme trajanja ovakvog stanja predstavlja vrijeme potrebno da se talas reflektira od objekta i vrati na senzor. Ovu informaciju smo koristili za računanje udaljenosti, te ispisivanje iste na računar.

Nakon osposobljavanja sistema, primijetili smo da su mjerenja pri umjerenoj temperaturi od 22°C i za čvršće prepreke veoma precizni. Zaključujemo da je parking senzor prikladan za upotrebu u stvarnom okruženju jer je prilikom parkiranja automobila najviše potrebno paziti na druge automobile, ograde i betonske zidove, što su čvrsti objekti.

5. Literatura

[1] Dr Samim Konjicija, Dr Abdulah Akšamović: *Predavanja na predmetu: Praktikum automatike,* Elektrotehnički fakultet, Univerzitet u Sarajevu, 2019/20.

[2] Parking Sensors Differences

[3] Dokumentacija za mikrokontroler Microchip PIC16F1939

[4] Dokumentacija za ultrazvučni senzor HC-SR04

[5] Dokumentacija za 3mm LED

[6] Dokumentacija za TTL-232R-PCB konvertor

[7] Čip FT232R