|  |  |  |
| --- | --- | --- |
| Description: UNIZNAK | UNIVERZITET U NOVOM SADU **FAKULTET TEHNIČKIH NAUKA** | Description: FTNZNAK |

Nikola Ristić

**Razvoj elektropneumatskog generatora kompresionih mehaničkih talasa**

- MASTER RAD -

Novi Sad, 2020.

|  |  |  |  |
| --- | --- | --- | --- |
| Redni broj, **RBR**: | |  | |
| Identifikacioni broj, **IBR**: | |  | |
| Tip dokumentacije, **TD**: | | Monografska publikacija | |
| Tip zapisa, **TZ**: | | Tekstualni štampani primerak | |
| Vrsta rada, **VR**: | | Master rad | |
| Autor, **AU**: | | Nikola Ristić | |
| Mentor, **MN**: | | Dr Dragan Šešlija | |
| Naslov rada, **NR**: | | Razvoj elektropneumatskog generatora kompresionih mehaničkih talasa | |
| Jezik publikacije, **JP**: | | Srpski | |
| Jezik izvoda, **ЈI**: | | Srpski | |
| Zemlja publikovanja, **ZP**: | | Srbija | |
| Uže geografsko područje, **UGP**: | | AP Vojvodina | |
| Godina, **GO**: | | 2020. | |
| Izdavač, **IZ**: | |  | |
| Mesto i adresa, **МА**: | | Fakultet tehničkih nauka, 21000 Novi Sad, Trg Dositeja Obradovića 6 | |
| Fizički opis rada**, FO**: (poglavlja/strana/citata/tabela/slika/grafika/priloga) | | (8/43/0/0/18/13/4) | |
| Naučna oblast, **NO**: | | Mehatronika | |
| Naučna disciplina, **ND**: | | Implementacija automatskih sistema | |
| Predmetna odrednica/Ključne reči, **PO**: | |  | |
| **UDK** | |  | |
| Čuva se, **ČU**: | | Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6 | |
| Važna napomena, **VN**: | |  | |
| Izvod, **IZ**: | |  | |
| Datum prihvatanja teme, **DP**: | |  | |
| Datum odbrane, **DO**: | | xx.yy.zzzz. | |
| Članovi komisije, **KO**: | Predsednik: | Dr Slobodan Dudić |
|  | Član: | Dr Dragan Rajnović | Potpis mentora |
|  | Član, mentor: | Dr Dragan Šešlija |  |

Obrazac **Q2.НА.04-05** - Izdanje 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accession number, **ANO**: | |  | | |
| Identification number, **INO**: | |  | | |
| Document type, **DT**: | | Monographic publication | | |
| Type of record, **TR**: | | Textual Printed Material | | |
| Contents code, **CC**: | | Master Thesis | | |
| Author, **AU**: | | Nikola Ristić | | |
| Mentor, **MN**: | | Ph.D. Dragan Šešlija | | |
| Title, **TI**: | | Electropneumatic generator of compressional mechanical waves development | | |
| Language of text, **LT**: | | Serbian | | |
| Language of abstract, **LA**: | | English | | |
| Country of publication, **CP**: | | Serbia | | |
| Locality of publication, **LP**: | | AP Vojvodina | | |
| Publication year, **PY**: | | 2020. | | |
| Publisher, **PB**: | |  | | |
| Publication place, **PP**: | | Faculty of technical sciences, 21000 Novi Sad, Trg Dositeja Obradovića 6 | | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | (8/43/0/0/18/13/4) | | |
| Scientific field, **SF**: | | Mechatronics | | |
| Scientific discipline, **SD**: | | Implementation of automatic systems | | |
| Subject/Key words, **S**/**KW**: | |  | | |
| **UC** | |  | | |
| Holding data, **HD**: | | The library of Faculty of technical sciences, Trg Dositeja Obradovića 6, Novi Sad | | |
| Note, **N**: | |  | | |
| Abstract, **AB**: | |  | | |
| Accepted by the Scientific Board on, **ASB**: | |  | | |
| Defended on, **DE**: | | xx.yy.zzzz. | | |
| Defended Board, **DB**: | President: | Ph.D. Slobodan Dudić |
|  | Member: | Ph.D. Dragan Rajnović | Menthor's signature |
|  | Member, Mentor: | Ph.D. Dragan Šešlija |  |

Obrazac **Q2.НА.04-05** - Izdanje 1

|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU ⚫ **FAKULTET TEHNIČKIH NAUKA**  21000 NOVI SAD, Trg Dositeja Obradovića 6 | Datum: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ МАСТЕР (MASTER) РАДА** | List/Listova: |
| 2/144 |

| Vrsta studija: | Master akademske studije |
| --- | --- |
| Studijski program: | **Mehatronika** |
| Rukovodilac studijskog programa: | **Prof. dr Mirko Raković** |

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | **Nikola Ristić** | Broj indeksa: | **H1 11/18** |
| Oblast: | **Implementacija automatskih sistema** | | |
| Mentor | **dr Dragan Šešlija, redovni profesor** | | |
| NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUSA FAKULTETA  IZDAJE SE ZADATAK ZA MASTER RAD, SA SLEDEĆIM ELEMENTIMA:   * problem – tema rada; * način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna; * literatura | | | |

**NASLOV MASTER RADA:**

|  |
| --- |
| **Razvoj elektropneumatskog generatora kompresionih mehaničkih talasa** |

**TEKST ZADATKA:**

|  |
| --- |
| * XXX * YYY * ZZZ |

|  |  |
| --- | --- |
| Rukovodilac studijskog programa: | Mentor rada: |
|  |  |

|  |
| --- |
| Primerak za:  - Studenta;  - Mentora |

Obrazac **Q2.НА.15-04** - Izdanje 2

**Zahvalnica**

Pre svega, želeo bih da izrazim zahvalnost svom mentoru prof. dr Draganu Šešliji za pomoć i ključne savete koje mi je pružio za vreme istraživanja ove teme. Znanje koje sam stekao na njegovim predavanjima ohrabrilo me je da se upustim u ovaj istraživački rad.

Srdačno se zahvaljujem prof. dr Draganu Rajnoviću na izvandrednoj saradnji, kao i na dugoročnom vremenu koje je uložio u ovaj rad. Njegov doprinos disertaciji je od neprocenjivog značaja.

Zahvaljujem se prof. dr Vladi Deliću i prof. dr Ivani Kovačić na njhovoj pomoći i korisnim predlozima koji su u velikoj meri doprineli istraživačkom radu. Takođe se zahvaljujem prof. dr Miodragu Žigiću i asistent-masteru Aleksandru Okuki na izuzetnoj saradnji i konstruktivnim savetima.

Zahvaljujem se prof. dr Slobodanu Dudiću i prof. dr Milanu Rackovu na njihovoj pomoći i sugestijama koje su značajno uticale na izradu disertacije, kao i na nesebičnom odvajanju vremena za saradnju sa mnom.

Zahvaljujem se prof. dr Vuletu Reljiću za pomoć i preporuke koje mi je pružio za izradu teze.

Posebnu zahvalnost dugujem svojim fizioterapeutima i prijateljima Stefanu Mirkoviću i Željani Vujin koji su me inspirisali da dođem do ideje za ovu disertaciju i podržali me da je sprovedem do kraja.

Zahvaljujem se svojim najbližim prijateljima sa kojima sam proveo studentske dane i koji su učinili da oni budu nezaboravni.

Na kraju, duboko se zahvaljujem svom ocu Davoru, dedi Milanu, bratu Strahinji i sestri Nini koji su mi pružili najveću podršku za vreme studiranja.

**Sadržaj**

**Spisak slika**

**Spisak tabela**

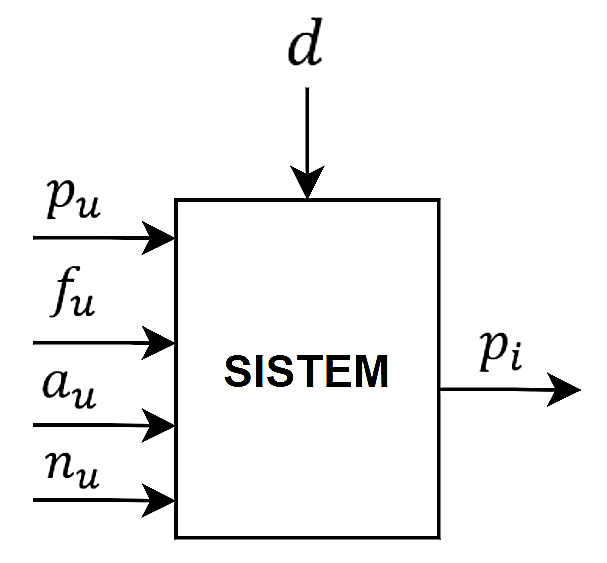
# **Uvod**

Elektropneumatska automatizacija procesa predstavlja jedan od najprimenjenijih načina za ostvarivanje željenog rada automatskih sistema. U opštem slučaju, izvršni organ koristi vazduh pod pritiskom kao energetski izvor za postizanje potrebnog funkcionisanja, dok se njegovo upravljanje ostvaruje upotrebom razvodnog ventila koji menja svoja stanja posredstvom elektromehaničkog mehanizma.

Postoje dva načina upravljanja elektropneumatskog procesa, a to su u otvorenoj i u zatvorenoj povratnoj sprezi. U prvom slučaju nije potrebna senzorska ili indikatorska komponenta za postizanje željenog ponašanja aktuatora, što smanjuje ukupne troškove postrojenja i pojednostavljuje implementaciju radne opreme. Međutim, takva realizacija ne pruža informacije o trenutnom stanju izvršnog organa. Iz tog razloga, često se koristi senzor ili indikator za zatvaranje povratne sprege kako bi se uspostavio tok informacija o realnim izlaznim veličinama nazad u sistem što omogućava regulaciju istog.

Tema ove teze je razvoj elektropneumatskog izvršnog organa koji generiše kompresione mehaničke talase, kao i korisnički interfejs koji je namenjen za njegovo upravljanje. Postoji mogućnost da se ovakav sistem primeni u domenu medicine koja se bavi fizikalnom terapijom telesnih povreda [1], ali su za to potrebna dodatna ispitivanja koja prevazilaze okvire ovog rada.

Generisanje talasnog polja posredstvom primenjenog aktuatora u ovom istraživanju zasnovano je na principu mehanike sudara čvrstih tela, gde jedno telo manje mase ubrzava ka nepomičnom telu veće mase usled dejstva vazduha pod pritiskom. Pneumatski signal se generiše upravljanjem na bazi impulsno širinske modulacije (*pulse width modulation*, *PWM*) čije se karakteristike mogu menjati posredstvom korisničkog interfejsa. Na slici 1 prikazana je šema sistema na kojoj se mogu primetiti ulazni i izlazni parametri:



Slika 1 - Šematski prikaz sistema

gde je:

* ulazni vazdušni pritisak,
* je frekvencija impulsno širinske modulacije,
* predstavlja faktor ispune *PWM* signala, a
* je broj perioda istog.

Izlazna veličina je pritisak talasnog polja koje generiše izvršni organ, dok je poremećaj koji nastaje usled dejstva spoljašnje sredine na sistem. Efekat poremećaja je od izuzetnog značaja, s obzirom da ne postoji zatvorena povratna sprega u sistemu.

TBC

# **Energija sistema**

## **Zakon održanja ukupne energije**

### Kinetička energija

U procesu vršenja rada dolazi do prenosa energije sa jednog tela na drugo ili do promene njegovog oblika. Međutim, da bi rad mogao da bude izvršen, mora da postoji bar jedna sila koja deluje na telo, usled čega dolazi do njegovog pomeranja u prostoru. Ukoliko je geometrijski zbir sila koje deluju na telo konstantan, rad koji one izvršavaju je:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

odnosno:

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

gde je ugao između glavnog vektora sistema sila koji uzrokuje rad i vektora pomeraja tela . U slučaju da je promenljiva funkcija koja zavisi od pozicije tela, a pomeranje se vrši duž krive linije, ukupan rad može se izraziti kao zbir elementarnih radova na konačnom broju delova puta na koje se podeli

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

pri čemu su dovoljno mali da se mogu smatrati pravolinijskim rastojanjima, predstavlja srednju vrednost sila na -om delu puta, a je ukupan broj delova . Kako kriva linija ne može da se podeli na konačan broj pravolinijskih delova, za računanje rada sila može se primeniti granični slučaj u kojem teži nuli, a teži beskonačnosti:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

gde je početna tačka putanje , a je krajnja.

S obzirom da se dejstvo sila manifestuje kao promena impulsa u obliku:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

jednačina (4) može se predstaviti u formi:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

pri čemu je početna brzina tela, a je krajnja. Rešavanjem integralne jednačine (6) dobija se izraz za promenu kinetičke energije kao:

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

što znači da je ukupan rad sistema:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

Jednačina (7) ukazuje da u opštem slučaju promena kinetičke energije tela zavisi od njene mase, kao i od njene početne i krajnje brzine. Ukoliko je početna brzina tela jednaka nuli, promena kinetičke energije istog je:

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

odakle sledi da je rad sila koji se ulaže u promenu brzine tela koje u početnom momentu miruje jednak:

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

Dakle, kinetička energija predstavlja sposobnost tela da vrši rad zahvaljujući svom kretanju.

Pri diferencijalno malim kretanjima, elementarni rad može se izraziti kao:

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

Uvrštavanjem jednačine (5) u jednačinu (11), dobija se:

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

što znači da je elementarni rad koji vrše sile pri pomeranju tela jednak elementarnoj promeni kinetičke energije.

### Potencijalna energija

Sile kod kojih rad ne zavisi od pređenog puta ili oblika putanje, već samo od početnog ili krajnjeg položaja tela nazivaju se konzervativne. Karakteristika takvih sila je da je njihov rad po zatvorenoj putanji jednak nuli [2].

Energija koja zavisi samo od položaja jednog tela u odnosu na druga sa kojima intereaguje, bilo to neposrednim kontaktom (npr. oprugom) ili posredstvom fizičkog polja (npr. gravitaciono polje) naziva se potencijalna energija. Ona se meri kao rad sila interakcija koji je potreban da se telo premesti sa jednog mesta (definisano vektorom položaja ) na drugo (definisano radijus vektorom ), tj. jednaka je negativnom radu konzervativnih sila:

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

ili drugačije zapisano:

|  |  |  |
| --- | --- | --- |
|  |  | (14) |

gde predstavlja razliku potencijalnih energija i , a je rad geometrijskog zbira svih konzervativnih sila koje deluju na sistem .

Ukoliko se potencijalna energija u tački usvoji kao referentna (što znači da je potencijalna energija u toj tački jednaka nuli), jednačina (13) može se prikazati u obliku:

|  |  |  |
| --- | --- | --- |
|  |  | (15) |

ili u opštem slučaju:

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

pri čemu predstavlja vektor položaja tela u odnosu na radijus vektor . Diferenciranjem jednačine (16) dobija se:

|  |  |  |
| --- | --- | --- |
|  |  | (17) |

što znači da je elementarni rad konzervativnih sila jednak negativnoj elementarnoj promeni potencijalne energije.

### Mehanička energija

Razmotrimo slučaj u kojem na sistem deluju samo konzervativne sile. Elementarni rad sila je tada:

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

odakle sledi:

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

Uvrštavanjem jednačina (12) i (17) u jednačinu (19) dobija se izraz:

|  |  |  |
| --- | --- | --- |
|  |  | (20) |

što znači da je:

|  |  |  |
| --- | --- | --- |
|  |  | (21) |

Zbir kinetičke i potencijalne energije naziva se mehanička energija . Jednačina (21) može se zapisati u obliku:

|  |  |  |
| --- | --- | --- |
|  |  | (22) |

Dakle, kinetička i potencijalna energija sistema na koji deluju samo konzervativne sile mogu da se menjaju, ali ukupna mehanička energija ne može. To znači da kinetička energija može da se pretvara u potencijalnu i obratno, ali mehanička energija sistema ostaje konstantna. Ovo predstavlja zakon održanja mehaničke energije.

Analizirajmo sada slučaj u kojem na sistem osim konzervativnih deluju i nekonzervativne sile. U takvim okolnostima, elementarni rad sila je:

|  |  |  |
| --- | --- | --- |
|  |  | (23) |

odnosno:

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

gde je elementarni rad nekonzervativnih sila. Ukoliko se u jednačinu (24) uvrste jednačine (12) i (17), dobija se izraz:

|  |  |  |
| --- | --- | --- |
|  |  | (25) |

odakle sledi:

|  |  |  |
| --- | --- | --- |
|  |  | (26) |

Znači, ako na sistem deluju i konzervativne i nekonzervativne sile, zakon održanja mehaničke energije ne važi jer je ukupna mehanička energija jednaka zbiru rada nekonzervativnih sila i početne mehaničke energije:

|  |  |  |
| --- | --- | --- |
|  |  | (27) |

pri čemu je početna mehanička energija sistema. Drugim rečima, mehanička energija sistema može se pretvoriti u druge oblike energije (npr. toplotnu ili akustičnu) ali se ne može uništiti niti ni iz čega stvoriti. Ukupna energija sistema uvek ostaje konstantna. Ovo pravilo naziva se zakon održanja ukupne energije.

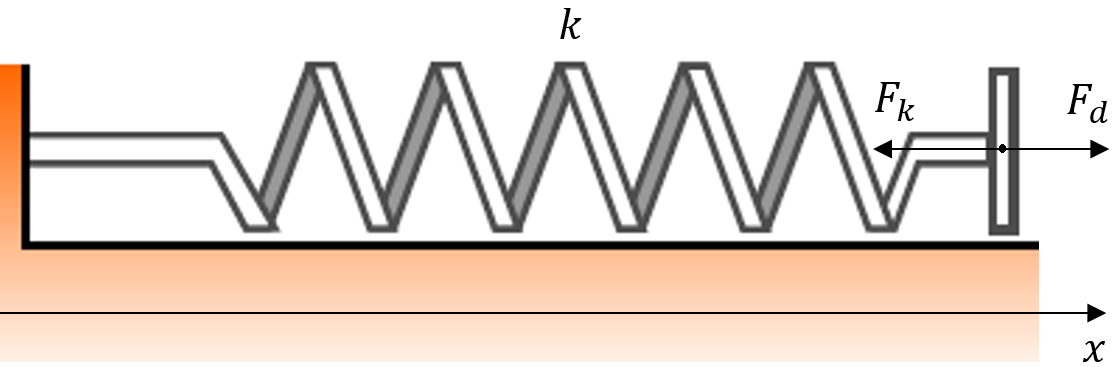
U opštem slučaju, često se početna mehanička energija sistema usvaja kao referentna. To znači da je njena vrednost jednaka nuli, a jednačina (27) se može napisati u obliku:

|  |  |  |
| --- | --- | --- |
|  |  | (28) |

## **Potencijalna energija elastične deformacije**

Elastičnost predstavlja svojstvo čvrstih tela da pod uticajem spoljašnje sile menjaju svoj oblik, pri čemu se posle prestanka dejstva iste vraćaju u prvobitno stanje. Ovakva karakteristika tela se u analizi potencijalne energije elastične deformacije modelira oprugom. Veza između opterećenja i deformacije tela opisuje se Hukovim zakonom [3].

Posmatrajmo oprugu krutosti koja je istegnuta usled dejstva aksijalne sile deformacije kao na slici 2.



Slika 2 - Deformisana opruga

Restituciona sila koja teži da vrati oprugu u prvobitno stanje može se izraziti kao linearna funkcija rastojanja (elongacije) :

|  |  |  |
| --- | --- | --- |
|  |  | (29) |

Uvrštavanjem jednačine (29) u integralnu jednačinu (4), dobija se da je rad sile koja teži da vrati oprugu u prvobitni oblik:

|  |  |  |
| --- | --- | --- |
|  |  | (30) |

S obzirom da je potencijalna energija jednaka negativnom radu konzervativnih sila, potencijalna energije opruge je:

|  |  |  |
| --- | --- | --- |
|  |  | (31) |

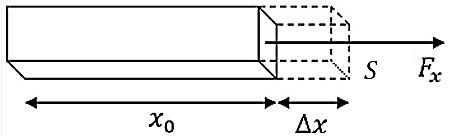
Potencijalna energija elastične deformacije može se shvatiti kao energija koja je akumulirana u deformisanoj opruzi, bilo da je sabijena ili istegnuta. Kada bi sila sabijene opruge delovala na telo mase kao na slici 3, telo bi se pomerilo u smeru dejstva sile opruge (pod pretpostavkom da ne postoji trenje između podloge i tela). Drugim rečima, potencijalna energija sabijene opruge pretvorila bi se u kinetičku energiju tela.



Slika 3 - Dejstvo sile opruge na telo mase m

Krutost tela može se odrediti primenom Hukovog zakona. Ako se telo izloži sili zatezanja kao na što je prikazano na slici 4, njena deformacija može se izračunati kao:

|  |  |  |
| --- | --- | --- |
|  |  | (32) |



Slika 4 - Deformacija tela pri dejstvu sile zatezanja

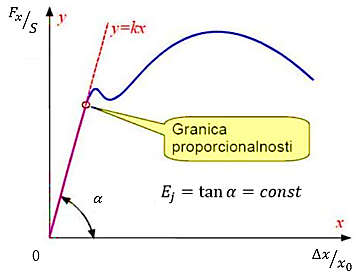
Jednačina (32) može se napisati u formi:

|  |  |  |
| --- | --- | --- |
|  |  | (33) |

gde je dužina tela pre deformacije, je površina poprečnog preseka tela, a je Jangov modul elastičnosti. Odavde sledi:

|  |  |  |
| --- | --- | --- |
|  |  | (34) |

Dakle, krutost tela zavisi od njegovih geometrijskih karakteristika i Jangovog modula elastičnosti. Međutim, relacija (34) važi samo u određenoj oblasti opterećenja materijala koja se može uočiti na dijagramu naprezanja koji je prikazan na slici 5.



Slika 5 - Oblast važenja linearne deformacije

# **Mehanički talasi**

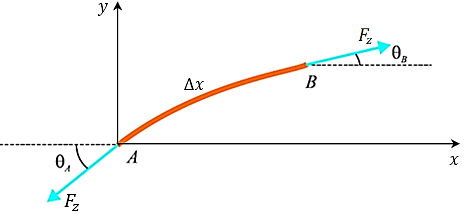
Oscilovanje čestice oko svog ravnotežnog položaja u neprekidnoj elastičnoj sredini, pri čemu se energija oscilovanja prenosi sa jedne na drugu česticu, naziva se mehanički talas. Da bi postojao takav talas, neophodno je da postoji:

* izvor poremećaja koji premešta česticu iz svog ravnotežnog položaja,
* medijum u kojem može da se izvede prostiranje talasa, i
* određena fizička veza koja međusobno povezuje čestice i posredstvom koje utiču jedna na drugu.

Kada se kruto telo pobudi na oscilacije, ono postaje izvor mehaničkih talasa usled njegovih elastičnih osobina koje omogućavaju prostiranje energije poremećaja kroz njega [4]. Takav slučaj postoji pri radu izvršnog organa koji je tema ove teze.

## **Talasna jednačina**

Jednačina koja opisuje vezu brzine mehaničkog talasa sa vremenskom i prostornom promenom njegove elongacije, naziva se talasna jednačina [5]. Kako bi je odredili, posmatrajmo prostiranje talasa na zategnutoj žici koji se dešava usled pobuđivanja oscilacijama jednog njenog kraja. Na slici 6, prikazan je jedan delić dužine žice koji je zategnut silom zatezanja , pri čemu krajevi tog delića zaklapaju uglove i sa x osom. Pretpostavimo da se oscilacije dešavaju samo u vertikalnom pravcu .



Slika 6 - Delić žice pobuđen na oscilacije

Rezultujuća sila koja deluje na delić žice u vertikalnom pravcu je:

|  |  |  |
| --- | --- | --- |
|  |  | (35) |

Kako se radi o malim uglovima, sinusna funkcija može se aproksimirati tangensnom, tako da će izraz (35) biti:

|  |  |  |
| --- | --- | --- |
|  |  | (36) |

S obzirom da tangens ugla predstavlja koeficijent pravca tangente na krivu, jednačina (36) može se predstaviti u obliku:

|  |  |  |
| --- | --- | --- |
|  |  | (37) |

Primenom Njutnovog zakona sile, rezultujuća sila može se prikazati u formi:

|  |  |  |
| --- | --- | --- |
|  |  | (38) |

pri čemu je je masa delića žice koja se može izračunati kao:

|  |  |  |
| --- | --- | --- |
|  |  | (39) |

gde je masa po jedinici dužine žice. Ukoliko se jednačina (39) uvrsti u izraz (38) i izjednači sa jednačinom (37), dobija se:

|  |  |  |
| --- | --- | --- |
|  |  | (40) |

odnosno:

|  |  |  |
| --- | --- | --- |
|  |  | (41) |

Pošto je prvi parcijalni izvod bilo koje funkcije po definiciji:

|  |  |  |
| --- | --- | --- |
|  |  | (42) |

jednačina (41) može se zapisati kao:

|  |  |  |
| --- | --- | --- |
|  |  | (43) |

što predstavlja diferencijalnu jednačinu kretanja žice linijske gustine koja je zategnuta silom zatezanja .

Može se dokazati da je sinusna talasna funkcija rešenje jednačine (43). Naime, ukoliko je zapišemo u obliku:

|  |  |  |
| --- | --- | --- |
|  |  | (44) |

gde je amplituda, je kružna frekcencija, a je talasni broj koji se definiše kao:

|  |  |  |
| --- | --- | --- |
|  |  | (45) |

pri čemu je talasna dužina sinusne funkcije, odgovarajući parcijalni izvodi su:

|  |  |  |
| --- | --- | --- |
|  |  | (46) |

|  |  |  |
| --- | --- | --- |
|  |  | (47) |

Uvrštavanjem izraza (46) i (47) u jednačinu (43), dobija se:

|  |  |  |
| --- | --- | --- |
|  |  | (48) |

odakle sledi:

|  |  |  |
| --- | --- | --- |
|  |  | (49) |

Kako se brzina prostiranja talasa računa kao:

|  |  |  |
| --- | --- | --- |
|  |  | (50) |

talasni broj se može izraziti kao:

|  |  |  |
| --- | --- | --- |
|  |  | (51) |

Zamenom jednačine (51) u izraz (49), brzina talasa može se zapisati u formi:

|  |  |  |
| --- | --- | --- |
|  |  | (52) |

Konačno, uvrštavanjem izraza (52) u diferencijalnu jednačinu (43), dobija se talasna jednačina žice čija je talasna brzina :

|  |  |  |
| --- | --- | --- |
|  |  | (53) |

Jednačina talsa koja je izvedena u ovom slučaju opisuje vertikalno kretanje delića zategnute žice. U opštem slučaju, talasno kretanje žice u trodimenzionalnom prostoru koji je karakterisan Dekartovim pravouglim koordinatnim sistemom može se opisati jednačinom:

|  |  |  |
| --- | --- | --- |
|  |  | (54) |

pri čemu je funkcija pomeraja žice, a je faktor brzine prostiranja talasa. Na sličan način može se odrediti talasna jednačina različitih tipova talasa u zavisnosti od medijuma kroz koji se prostiru. Ako se u medijumu pojave više talasa usled dejstva neke pobudne sile, može se definisati pojam talasnog fronta. Naime, talasni front predstavlja geometrijsko mesto tačaka koje osciluju sa istom fazom. Postoje dva idealizovana tipa talasnog fronta, a to su ravanski i sferni [4]. U realnim uslovima, širenje mehaničkih talasa u prostoru nikada nema idealan ravanski ili sferni geometrijski oblik, ali nekada postoje okolnosti pod kojima se njihov talasni front može tako aproksimirati.

## **Mehanički talasi u čvrstim telima**

Jedan od načina generisanja mehaničkih talasa je oscilovanje čvrstim telom. Vibracijom njegovih površina postiže se sabijanje i razređivanje čestica sredine u kojoj se nalazi [4]. Drugim rečima, energija deformacije površinskih čestica čvrstog tela prenosi se na čestice medijuma koji ga okružuju. Na taj način čvrsto telo postaje izvor mehaničkih talasa.

U čvrstim telima mehanički talasi se klasifikuju u zavisnosti od pravca njihovog prostiranja u odnosu na pravac oscilovanja čestica. Talasi kod kojih se oscilacije dešavaju u pravcu koji je paralelan sa pravcem prostiranja talasa nazivaju se kompresioni (longitudinalni). Primer takvih talasa predstavljen je na slici 7.



Slika 7 - Kompresioni talasi

Elastična deformacija molekula pri prostiranju kompresionog talasa kroz čvrsto telo ilustrovana je na slici 8. Molekuli mase međusobno su povezani oprugama koje modeluju međumolekularne veze između njih. Ukoliko se na jednu od čestica kratkotrajno deluje nekom spoljašnjom silom dovoljno velikog intenziteta, dolazi do lokalne deformacije sredine, odnosno uneta je energija deformacije u sistem. Ona se manifestuje kao kretanje čestice na osnovu kinetičke energije koju je dobila. Drugim rečima, u sistemu dolazi do pomeranja molekula u pravcu dejstva sile i postepenog prelaska kinetičke energije njenog kretanja u potencijalni oblik u susednim oprugama. Ovo stanje je predstavljeno na segmentu slike pod 8b. Poremećaj se dalje prenosi na susedne čestice kao što se može videti na segmentu slike pod 8c. Na slici se takođe može uočiti da čestica *n* osciluje oko svog ravnotežnog položaja. Taj proces će postojati dokle god čestica sadrži kinetičku energiju.



Slika 8 - Elastična deformacija sredine pri prostiranju kompresionog talasa

Brzina promene intenziteta sile koja deluje na čvrsto telo i koja uzrokuje njegovu lokalnu deformaciju je od suštinske važnosti za pojavu značajnih talasnih prostiranja u istom. Ako bi promena intenziteta sile bila previše spora, progresija talasa bi bila previše brza u odnosno na promenu sile, ali bi elongacije čestica bile previše male. Deformacija bi u tom slučaju bila zanemarljiva i došlo bi samo do translatornog pomeranja tela duž pravca delovanja sile, što predstavlja makroskopsko pomeranje materije. Dakle, za pobudnu silu treba da važi:

|  |  |  |
| --- | --- | --- |
|  |  | (55) |

pri čemu mora biti ispunjen uslov:

|  |  |  |
| --- | --- | --- |
|  |  | (56) |

Veličina naziva se trzaj tela i definiše se kao:

|  |  |  |
| --- | --- | --- |
|  |  | (57) |

Brzina longitudinalnog talasa u čvrstom telu može se izračunati primenom obrasca:

|  |  |  |
| --- | --- | --- |
|  |  | (58) |

gde je brzina kompresionog talsa, a je gustina medijuma. Talasna jednačina pri elongaciji koja se opisuje funkcijom je tada:

|  |  |  |
| --- | --- | --- |
|  |  | (59) |

Ukoliko se oscilacije čestica čvrstog tela dešavaju upravno u odnosu na pravac prostiranja talsa, takvi talasi se nazivaju transferzalni (savijajući) i mogu se videti na slici 9. Njihova brzina može se izračunati na sličan način kao i u slučaju longitudinalnih. Razlika je što umesto Jangovog modula elastičnosti, u izrazu za brzinu talasa figuriše modul smicanja :

|  |  |  |
| --- | --- | --- |
|  |  | (60) |

pri čemu je brzina transferzalnog talasa. Talasna jednačina savijajućih talasa pri elongaciji koja se opisuje funkcijom je:

|  |  |  |
| --- | --- | --- |
|  |  | (61) |



Slika 9 - Transferzalni talasi

## **Komprseioni talasi**

Longitudinalni talasi mogu se prostirati kroz čvrsta tela, ali za razliku od transferzalnih, mogu se kretati i u sredinama koje su tečnog ili gasovitog agregatnog stanja. Njihova talasna jednačina može se predstaviti izrazom (59), ali će njihova brzina zavisiti od medijuma u kojem se nalaze. Međutim, kompresioni talasi se manifestuju kao promena pritiska okoline u kojoj se prostiru, tako da postoji još jedan vid jednačine koji se koristi za opisivanje istih, čija je forma slična jednačini (59):

|  |  |  |
| --- | --- | --- |
|  |  | (62) |

Funkcija opisuje promenu pritiska u zavisnosti od vremena i jedne dimenzije prostora. Ako se posmatra trodimenzionalna promena pritiska, longitudinalna talasna jednačina glasi:

|  |  |  |
| --- | --- | --- |
|  |  | (63) |

gde je funkcija koja opisuje trodimenzionalnu prostornu promenu pritiska u zavisnosti od vremena. Na slici 10 može se videti promena pritiska u prostoru kroz koji se kreću kompresioni talasi pri deformaciji sredine usled sinusnog pobudnog signala.



Slika 10 - Promena pritiska u prostoru kroz koji se prostiru kompresioni talasi pri sinusnoj promeni pobudnog signala

Veza između elongacije talasa i promene pritiska okoline u kojoj se kreću zavisi od sredine kroz koju se prostire energija deformacije jer i brzina talasa zavisi od istog faktora. Tu vezu ilustrovaćemo na primeru u kojem medijum kroz koji se talas kreće predstavlja fluid. Primenom izraza (58) može se izračunati brzina talasa u čvrstom telu. Međutim, na sličan način se može izračunati brzina talasa i u fluidu. Razlika je što će u izrazu za brzinu umesto Jangovog modula elastičnosti figurisati modul stišljivosti fluida:

|  |  |  |
| --- | --- | --- |
|  |  | (64) |

pri čemu je brzina longitudinalnog talasa u fluidu, a je modul stišljivosti istog. Opšta veza između elongacije i pritiska u fluidu [6] može se izraziti jednačinama:

|  |  |  |
| --- | --- | --- |
|  |  | (65) |

|  |  |  |
| --- | --- | --- |
|  |  | (66) |

U ovom primeru pretpostavićemo da se prostiranje talasa dešava u formi kosinusne funkcije:

|  |  |  |
| --- | --- | --- |
|  |  | (67) |

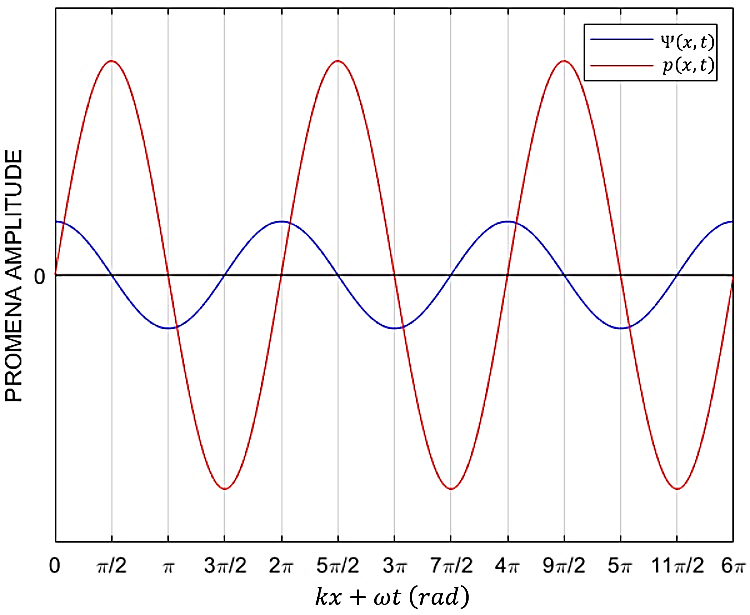
gde je amplituda elongacije. U tom slučaju, promena pritiska [7] može se opisati jednačinom:

|  |  |  |
| --- | --- | --- |
|  |  | (68) |

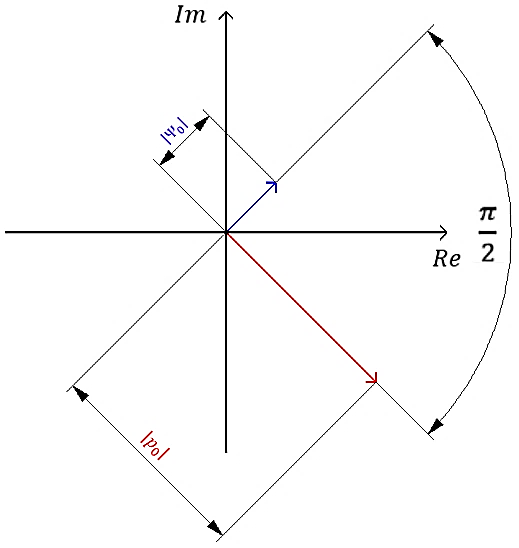
pri čemu je amplituda pritiska. Odavde sledi da je veza između amplitude elongacije i amplitude pritiska:

|  |  |  |
| --- | --- | --- |
|  |  | (69) |

Na slici 11 predstavljen je grafik promene signala pritiska i elongacije, a na slici 12 se može videti fazorski dijagram istih. Na njima se može primetiti da su talasna dužina, frekvencija i talasna brzina iste za oba signala, ali da fazor elongacije prednjači u odnosu na fazor pritiska za . To znači da je apsolutna vrednost amplitude pritiska maksimalna kad je elongacija jednaka nuli i obratno.



Slika 11 - Signali pritiska i elongacije

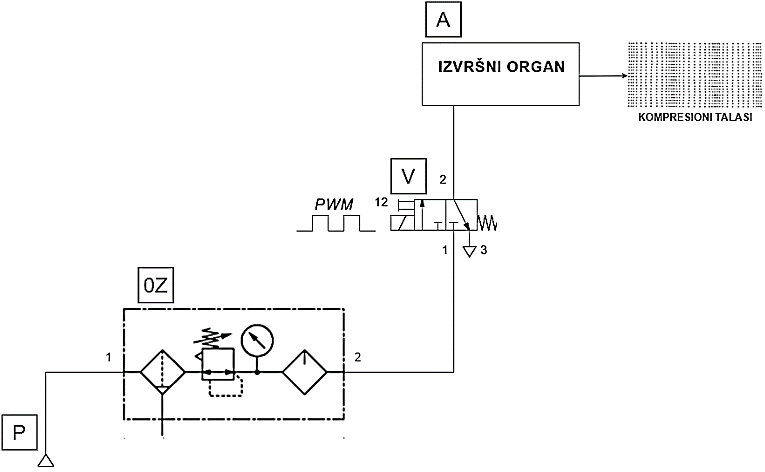


Slika 12 - Fazorski dijagram pritiska i elongacije

Karakteristika longitudinalnih talasa da se manifestuju kao prostorna promena pritiska je od velikog značaja za analizu njihovih karakteristika. Takođe, pojednostavljuje proces merenja njihovih svojstva. Kada se kompresiono talasno polje meri u jednoj fiksnoj tački prostora (što je u praksi skoro uvek slučaj pošto je takva tačka definisana položajem senzora kojim se registruje stanje u polju), talasni pritisak se svodi samo na vremensku funkciju . Merni instrument koji je namenjen za merenje takve promene pritiska je mikrofon.

# **Princip rada elektropneumatskog generatora kompresionih mehaničkih talasa**

Elektropneumatski procesi bez povratne sprege najčešće se sastoje od jednog ili više izvršnih organa koji koriste isključivo vazduh pod pritiskom kao izvor energije, dok se za razvodne ventile koji upravljaju njima koristi i električna energija. U sistemu koji se izučava u ovom radu, električni signal kojim se upravlja elektromagnetnim ventilom je u obliku impulsno širinske modulacije. Na taj način se na njegovom izlazu formira pneumatski *PWM* signal kojim se upravlja aktuatorom. Elektropneumatska šema sistema predstavljena je na slici 13.



Slika 13 – Elektropneumatska šema sistema

## **Izvršni organ**

Namena izvršnog organa je pretvaranje ulazne pneumatske energije u polje kompresionih mehaničkih talasa na svom izlazu. Na slici 14 može se videti trodimenzionalni *CAD* (*computer aided design*) model aktuatora, a na slici 15 realan model. Takođe, u prilogu 1 dati su sklopni crtež i sastavnica, kao i radionički crteži neophodni za njegovu izradu.



Slika 14 - CAD model izvršnog organa

Slika 15 - Realan model izvršnog organa

### Integracija razvodnog ventila u izvršni organ

Primenjeni elektromagnetni ventil u sistemu je 3/2 normalno zatvoreni monostabilni električno aktivirani razvodnik kompanije *FESTO* sa oznakom *MHE2-MS1H-3/2G-M7*. Pogodan je za aplikaciju s obzirom da sadrži ugrađenu elektroniku koja omogućava veoma kratko vreme uključenja i isključenja istog [8] (oko 2 *ms*). U prilogu 2 nalaze se njegove tehničke specifikacije.

Na slici 16 može se uočiti način na koji je razvodni ventil integrisan u izvršni organ. Pričvršćen je unutar kućišta pomoću navojnog spoja zavrtnjeva i navrtki, pri čemu vazduh kroz njega dospeva u aktuator posredstvom spoja pneumatskog utičnog i cilindričnog priključka. Međutim, prigušnica zvuka se nalazi izvan kućišta kako bi se sprečilo povećanje pritiska unutar istog. Pneumatsko crevo, kao i električni vodovi razvodnog ventila zaštićeni su od spoljašnjih uticaja okoline rebrastim crevom koje ih obavija. Ono je stegnuto za cilindrični segment poklopca kućišta dejstvom obujmice.



Slika 16 - Prikaz integracije razvodnog ventila u izvršni organ

Razvodnim ventilom se upravlja na način kojim se obezbeđuje generisanje pneumatskog *PWM* signala na njegovom izlazu. Takva funkcija postignuta je dovodom naponske impulsno širinske modulacije na njegov električni ulaz, pri čemu se na njegovom pneumatskom ulazu nalazi konstantan pritisak. Na slici 17 prikazani su signali razvodnog ventila za vreme radnog režima izvršnog organa.



Slika 17 - Signali razvodnog ventila za vreme radnog režima izvršnog organa

Kako se koristi pneumatski *PWM* signal za upravljanje aktuatorom, poželjno je da razvodni ventil bude montiran unutar kućišta istog. Naime, rasterećenje vazdušnog pritiska unutar izvršnog organa je na taj način brže u odnosu na slučaj kada se razvodnik nalazi daleko od njega. Takvom konfiguracijom aktuator ima više vremena da se vrati u početno stanje pre početka sledeće periode impulsno širinske modulacije, a to omogućava korišćenje veće frekvencije i/ili faktora ispune upravljačkog signala.

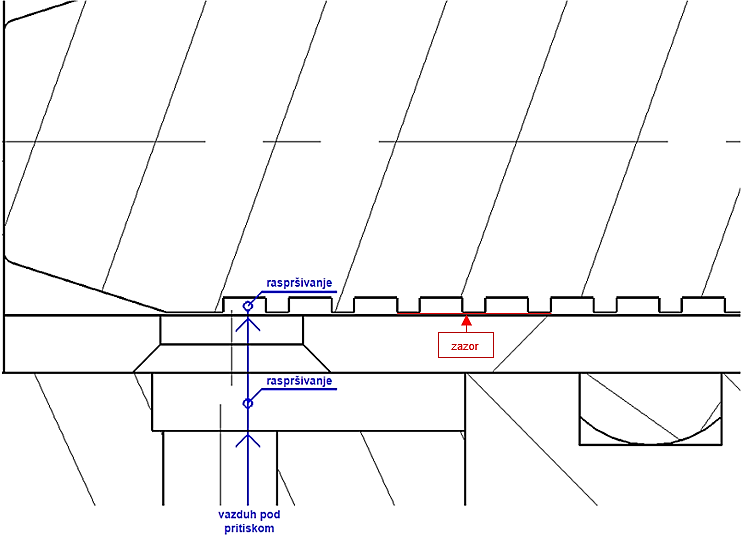
### Pogon projektila

Projektil je jedan od dva čvrsta tela koja učestvuju u sudaru koji se dešava prilikom rada sistema. U početnom položaju nalazi se u stanju mirovanja kao što je prikazano na slici 18. Sa slike je uklonjena unutrašnja cev u kojoj se nalazi zbog preglednosti. Može se primetiti da metak dodiruje silikonsko crevo svojom čeonom površinom. U takvoj poziciji nalazi se usled dejstva privlačne sile magnetnog cilindra.



Slika 18 - Početni položaj projektila

Polarizacijom električnih vodova razvodnog ventila, vazduh pod pritiskom na izlazu pneumatskog cilindričnog priključka prolazi kroz otvor držača unutrašnje cevi. Tada, sila pneumatskog pritiska savladava silu magnetnog cilindra i ubrzava (pogoni) projektil. Žljebovi na metku poboljšavaju njegova aerodinamička svojstva, a to omogućava veće ubrzanje istog, kao i brži prolaz kroz unutrašnju cev. Međutim, vazuh pod pritiskom se takođe prostire kroz unutrašnju cev prolazeći kroz zazor između cevi i projektila koji se može uočiti na slici 19. Položaj u kojem se nalazi projektil na slici može se takođe videti na sklopnom crtežu u prilogu 1 (*Детаљ В*).



Slika 19 - Protok vazduha pod pritiskom do projektila

Držač unutrašnje cevi zaptiven je posredstvom O-prstena između njega i stabilizatora unutrašnje cevi kao što je prikazano na slici 20. Dok vazduh pod pritiskom protiče do projektila, on se raspršuje unutar komore držača, usled čega raste pneumatski pritisak u njoj. Taj pritisak će početi da se smanjuje kada razvodni ventil pređe u zatvoreno stanje.



Slika 20 - Položaj otvora unutrašnje cevi u izvršnom organu

O-prsten između unutrašnje cevi i njenog stabilizatora ima dve funkcije u sistemu. Jedna je sprečavanje vibracije i zakretanje unutrašnje cevi za vreme rada izvršnog organa, a druga je zaptivanje unutrašnje sredine stabilizatora kako ne bi došlo do povećanja pritiska u njoj.

### Kretanje projektila prema membrani

Ubrzani projektil se usled impulsa pneumatskog *PWM* signala kreće kroz unutrašnju cev prema membrani. Kako je vazduh pod pritiskom zauljen, zauljena je i površina cevi po kojoj metak klizi. Prema tome, sila trenja može se zanemariti u analizi kretanja istog. Takođe, postoji zazor između projektila i cevi, što znači da njegovo kretanje nije idealno pravolinijski. Međutim, u nastavku tumačenja smatra da se metak kreće isključivo u pravcu koji je paralelan sa uzdužnom osom unutrašnje cevi.

Uticaj sile primenjenog neodijumskog magnetnog cilindra na projektil smanjuje se proporcionalno kvadratnom rastojanju između njih [9]. Eksperimentalno je utvrđeno da privlačno dejstvo cilindra ne deluje na projektil ukoliko je distanca između njih veće od 6 *mm*, pri čemu je merenje obavljeno u uslovima gde se između magnetnog cilindra i metka nalazi silikonsko crevo dužine 3 *mm* (što je slučaj koji postoji u izvršnom organu). Iz tog razloga, uticaj magnetnog polja cilindra na brzinu i ubrzanje projektila je zanemaren u analizi njegovih opterećenja.

Na slici 21 prikazana su opterećenja koja deluju na projektil dok se isti kreće prema membrani. Ugao nagiba projektila u odnosu na horizontalu označen je sa . Osa je kolinearna sa uzdužnom osom unutrašnje cevi, a i su brzina i ubrzanje metka.



Slika 21 - Sile koje utiču na brzinu i ubrzanje projektila za vreme njegovog kretanja prema membrani

Kao što se može primetiti sa slike, na brzinu i ubrzanje projektila utiču tri značajne sile:

1. aktivna komponenta sile zemljine teže,
2. sila koja postoji usled pogonskog pneumatskog pritiska koji ubrzava projektil prema membrani (u nastavku pneumatska pogonska sila) i
3. sila koja postoji usled vazdušnog pritiska sredine na suprotnoj strani projektila u odnosu na dejstvo pogonskog pneumatskog pritiska (u nastavku sila suprotsavljajućeg pritiska).

Na osnovu smerova dejstva sila može se napisati jednačina:

|  |  |  |
| --- | --- | --- |
|  |  | (70) |

Kako je masa projektila poznata veličina i faktor trenja je zanemarljiv zbog zauljenosti unutrašnje cevi, uticaj zemljine teže na metak može se jednostavno odrediti. Primenom obrasca:

|  |  |  |
| --- | --- | --- |
|  |  | (71) |

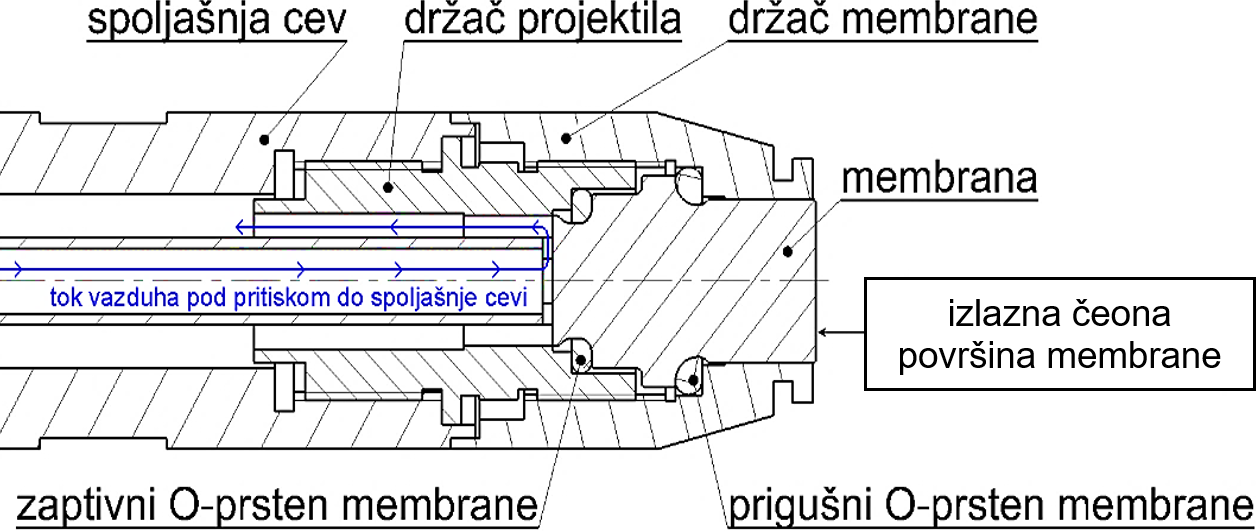
gde je konstanta gravitacionog ubrzanja, može se izračunati intenzitet aktivne komponente sile zemljine teže [10]. U skladu sa time mogu se napisati izrazi za brzinu i ubrzanje projektila dok se kreće prema membrani kao:

|  |  |  |
| --- | --- | --- |
|  |  | (72) |

|  |  |  |
| --- | --- | --- |
|  |  | (73) |

pri čemu je vremenski interval između početnog i posmatranog trenutka kretanja, i su brzina i ubrzanje koje metak dobija usled uticaja pneumatske pogonske sile, a i su brzina i ubrzanje projektila zbog dejstva sile suprotsavljajućeg pritiska. Početna brzina i ubrzanje mogu se smatrati nulama pošto je metak u početnom momentu bio u stanju mirovanja.

Pneumatska pogonska sila je posledica dejstva vazduha pod pritiskom iz razvodnog ventila koja podstiče kretanje projektila prema membrani. U isto vreme, vazduh protiče kroz zazor između unutrašnje cevi i projektila, a time raste intenzitet sile suprotstavljajućeg pritiska koji usporava kretanje metka. Isti vazduh prolazi kroz otvor unutrašnje cevi i držača projektila, čime dospeva do komore spoljašnje cevi u kojoj se akumulira. Tok vazduha do spoljašnje cevi pikazan je na slici 22.



Slika 22 - Tok vazduha pod pritiskom do spoljašnje cevi

Za razliku od uticaja zemljine teže, karakteristike pneumatske pogonske sile i sile suprotstavljajućeg pritiska nije jednostavno odrediti, kao ni njihove doprinose brzini i ubrzanju projektila. One zavise od periode i faktora ispune pneumatskog upravljačkog signala, zapremina komora u aktuatoru, kao i od pritiska i protoka ulaznog vazduha pod pritiskom

### Sudar projektila i membrane

Sudar projektila i membrane (sonde, talasovoda) predstavlja ključni potproces koji se dešava za vreme rada sistema. Ovakav tip sudara naziva se centralni čeoni sudar dva tela [2]. Za takav događaj važi:

* postoji prava linija koja je paralelna sa pravcem kretanja tela koja učestvuju u sudaru i prolazi kroz centre masa istih (centralni sudar), i
* oba tela pre i posle sudara se kreću duž istog pravca (čeoni sudar)

Prilikom sudara, dolazi do lokalne deformacije sonde. Na mikroskopskom nivou, na mestu sudara dešava se odstupanje molekula materijala membrane iz svog ravnotežnog položaja u smeru kretanja projektila prema njoj, usled čega dolazi do prostiranja longitudinalnih talasa kroz nju u istom smeru [11]. Energija deformacije se prenosi na susedne čestice sve dok talas ne stigne do izlazne površine talasovoda. Tada, površinske čestice sonde predaju jedan deo svoje energije molekulima okoline, čime se u toj sredini generišu kompresioni mehanički talasi.

Sudar predstavlja događaj koji se dešava u kratkom vremenskom intervalu. Za modelovanje takvog procesa primenjena je Hercova teorija o sudarima [12]. Naime, tela u sudaru se mogu modelovati kao nedeformabilna u oblasi koju okupiraju, osim lokalne okoline mesta kontakta. U toj okolini može se smatrati da postoji visko-elastični sloj nanet na nedeformabilnu celinu tela koji se predstavlja visko-elastičnom komponentom. Elementi koji se mogu iskoristiti za aproksimaciju karakteristika visko-elastične komponente su razni (Hukovo telo, Maksvelovo telo, Kelvin-Vojtovo telo, Zenerovo telo, itd.). U zavisnosti od odabranog elementa, aproksimacija će biti bliža ili dalja realnim svojstvima.

Kako se na membrani nalaze dva O-prstena koji utiču na karakteristike sudara, odabrani element za modelovanje visko-elastičnih svojstva istih je Kelvin-Vojtovo telo. Sastoji se od paralelne veze opruge krutosti i prigušnice (amortizera) faktora prigušenja na način koji je prikazan na slici 23.



Slika 23 - Kelvin-Vojtovo telo

Dinamičko svojstvo Kelvin-Vojtovog tela opisuje se jednačinom:

|  |  |  |
| --- | --- | --- |
|  |  | (74) |

gde je sila Kelvin-Vojtovog tela, a je elongacija istog. Eksperimentalno je utvrđeno da Kelvin-Vojtovo telo može dovoljno blizu da aproksimira visko-elastične karakteristike O-prstena [13] i postoje razvijene metode kojima se mogu ustanoviti vrednost parametara i . U aplikaciji su primenjeni s obzirom da je njihova krutost veća ako su izloženi pritisku nego uvijanju ili smicanju. Takođe, njihov koeficijent krutosti može se smatrati konstantnim pri malim deformacijama i mogu da amortizuju (apsorbuju) veliku količinu energije usled unutrašnjeg klizanja u strukturi gume [14]. Na slici 24 mogu se videti projektil i membrana zajedno sa dva O-prstena koji je obavijaju. Namena manjeg O-prstena je zaptivanje unutrašnje sredine držača sonde kako bi se sprečio protok vazduha u nju, a veći O-prsten služi za prigušenje energije sudara.



Slika 24 - Membrana i projektil

Element koji je upotrebljen za aproksimaciju visko-elastičnih svojstva projektila i talasovoda je Hukovo telo. Predstavlja se oprugom konstantne krutosti kao što je prikazano na slici 25.



Slika 25 - Hukovo telo

U slučaju redne veze dva Hukova tela krutosti i , ekvivalentna krutost opruga može se izračunati kao:

|  |  |  |
| --- | --- | --- |
|  |  | (75) |

Na slici 26 mogu se videti projektil i membrane u jednom vremenskom trenutku sudara.



Slika 26 - Sudar projektila i membrane

Analizirajmo sudar tumačenjem njegovog početnog i kasnijeg momenta koji se mogu videti na slici 27. Primenjena su dva nepokretna koordinatna sistema čije su ose kolinearne, pri čemu je distanca između koordinatnih početaka istih obeležena oznakom i posmatra se u pravcu koji je paralelan sa njima. Takođe, ose membrane i projektila su pod nagibom koji je definisan uglom u odnosu na horizontalni pravac i paralelne su sa koordinatnim sistemima. U tabeli 1 nalazi se spisak parametara koji služi za opis fizičkih veličina procesa sudara. Jednačine dinamike glase:

|  |  |  |
| --- | --- | --- |
|  |  | (76) |

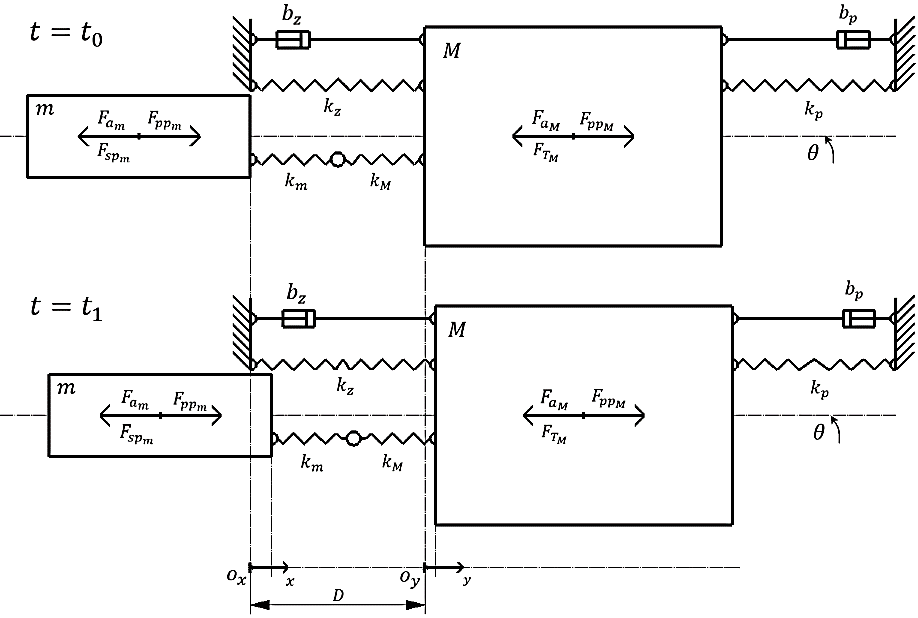
|  |  |  |
| --- | --- | --- |
|  |  | (77) |

pri čemu su poznata dva početna uslova:

|  |  |  |
| --- | --- | --- |
|  |  | (78) |

|  |  |  |
| --- | --- | --- |
|  |  | (79) |

Karakteristike pogonske pneumatske sile i sile suprotsavljajućeg pritiska koje deluju na projektil nisu matematički jednostavne odredljive veličine. To predstavlja razlog zbog kojeg početna brzina i ubrzanje metka pri sudaru nisu poznate veličine. Takođe, dok se projektil kreće, na talasovod deluje sila pritiska akumuliranog vazduha koji je prošao kroz zazor metka i unutrašnje cevi. Iako na nju deluje sila Kelvin-Vojtovog tela prigušnog  
O-prstena, ne može se sa potpunom sigurnošću znati da li membrana ima ili nema brzinu u početnom trenutku sudara.



Slika 27 - Model dinamike sudara projektila i membrane

|  |  |  |
| --- | --- | --- |
| **redni broj** | **oznaka** | **specifikacija** |
| **1.** |  | **početni trenutak sudara** |
| **2.** |  | **proizvoljni trenutak sudara** |
| **3.** |  | **masa projektila** |
| **4.** |  | **masa membrane** |
| **5.** |  | **faktor prigušenja zaptivnog O-prstena membrane** |
| **6.** |  | **faktor prigušenja prigušnog O-prstena membrane** |
| **7.** |  | **koeficijent krutosti zaptivnog O-prstena membrane** |
| **8.** |  | **koeficijent krutosti prigušnog O-prstena membrane** |
| **9.** |  | **koeficijent krutosti projektila** |
| **10.** |  | **koeficijent krutosti membrane** |
| **11.** |  | **aktivna komponenta sile zemljine teže koja deluje na projektil** |
| **12.** |  | **aktivna komponenta sile zemljine teže koja deluje na membranu** |
| **13.** |  | **pneumatska pogonska sila koja deluje na projektil** |
| **14.** |  | **sila vazduha pod pritiskom koja deluje na membranu** |
| **15.** |  | **sila suprotstavljajućeg pritiska koja deluje na projektil** |
| **16.** |  | **sila trenja koja deluje na membranu** |
| **17.** |  | **koordinatni početak ose** |
| **18.** |  | **koordinatni početak ose** |
| **19.** |  | **distanca između koordinatnih početaka i** |
| **20.** |  | **ugao nagiba između ose membrane i horizontalnog pravca** |

Tabela 1 - Lista fizičkih veličina koje opisuju dinamički proces sudara

Na sličan način na koji se računa intenzitet sile može se odrediti i intenzitet sile aktivne komponente zemljine teže koja deluje na membranu. S obzirom da su masa membrane i ugao nagiba poznate veličine, intenzitet sile se može izračunati primenom obrasca:

|  |  |  |
| --- | --- | --- |
|  |  | (80) |

Takođe, može se odrediti i sila trenja koja deluje na talasovod kao [10]:

|  |  |  |
| --- | --- | --- |
|  |  | (81) |

gde je koeficijent trenja klizanja.

### Izlazni signal

Kao što je pokazano pri izvođenju talasne jednačine užeta u poglavlju 3, bila je potrebna pobudna sila zatezanja da uzrokuje progresiju talasa duž nje. Analogno, sila kojom projektil deluje na sondu izaziva prostiranje longitudinalnih mehaničkih talasa kroz nju. Međutim, zbog pojave refleksije i toplotne disipacije samo jedan deo energije tih talasa uspeva da se prenese na okolinu koja okružuje izlaznu čeonu površinu talasovoda. Ta energija se manifestuje kao promena pritiska medijuma na koji deluje izvršni organ. Upravo ta promena pritiska predstavlja izazni signal sistema.

Da bi se mogla jednoznačno matematički odrediti zavisnost izlaznog pritiska od vremena i prostora, neophodno je rešiti talasnu jednačinu koja opisuje kretanje mehaničkih talasa u membrani, a posle toga i u sredini na koju deluje. Taj proces prevazilazi okvire rada pošto predstavlja analitički složen proces koji podrazumeva utvrđivanje graničnih uslova koji važe u datim okolnostima za sve granične površine u zoni gde postoji talasno polje. U praksi postoje softverski paketi koji to rešavanje obavljaju numerički na osnovu diskretizacije prostora u kojem postoji talasno polje. Takav postupak naziva se metoda konačnih elemenata, ali i u tom slučaju i dalje postoji problem određivanja graničnih uslova [4].

Željeni izlazni signal pri jednom sudaru projektila i sonde zasnovan je na izmerenim karakteristikama pritiska koji generiše sličan aktuator identične namene koji je opisan u radu [15]. S obzirom da izvršni organ treba da deluje na ljudsku kožu, medijum u kojem se želi postići izlazni signal je voda (talasna impedansa vode je je slična talasnoj impedansi tkiva jer se ono pretežno sastoji od vode [1]).

Ciljana maksimalna pozitivna promena pritiska duž ose talasovoda u zavisnosti od rastojanja od centra njene čeone izlazne površine predstavljena je dijagramom na slici 28. Na njemu se može primetiti linearna zavisnost posmatranih parametara, što znači da je odnos promene pritiska i rastojanja konstantan. Iako ovo predstavlja svojstvo sfernog talasnog fronta, zona u kojoj postoji promena pritiska ima oblik zarubljene kupe čija je baza u obliku sfernog isečka. Željeni ugao nagiba kupe je 76,5. Na slici 29 prikazan je geometrijski oblik talasnog polja.



Slika 28 - Promena maksimalne pozitivne vrednosti pritiska duž ose membrane u odnosu na centar njene izlazne čeone površine



Slika 29 – Geometrijski oblik talasnog polja

Vreme trajanja oscilacije pri kojoj se javlja maksimalna pozitivna promena pritiska u bilo kojoj tački polja treba da bude najviše 11 *μs*, pri čemu vreme trajanja pozitivne promene pritiska treba da iznosi maksimalno 8 *μs*. Grafička ilustracija navedenih vremenskih karateristka data je na slici 30.



Slika 30 - Vreme trajanja oscilacije sa dominantnom pozitivnom promenom pritiska u bilo kojoj tački talasnog polja

### Povratak izvršnog organa u početno stanje posle sudara

Povratak izvršnog organa u početno stanje posle sudara podrazumeva vraćanje projektila i membrane u početnu poziciju. Rasterećenje vazdušnog pritiska koji postoji u komorama aktuatora takođe spada u taj proces

Kako na sondu deluju visko-elastične sile dva O-prstena koji je obavijaju, očigledno je da se posle sudara membrana vrati u početni položaj relativno brzo u odnosu na metak. Za razliku od talasovoda, projektil treba da pređe put kroz unutrašnju cev da bi dospeo do svoje početne lokacije. Povratak mu omogućava kinetička energija koju ima usled sudara, kao i sila suprotstavljajućeg pritiska. Pneumatska pogonska sila usporava kretanje metka prema početnoj poziciji, dok doprinos aktivne komponente sile zemljine teže brzini istog zavisi od ugla nagiba.

U momentu kada se projektil dovoljno približi svom početnom položaju, privlačna sila mangetnog cilindra ga pozicionira na površinu silikonskog creva koja potpuno amortizuje njegovu kinetičku energiju. Na taj način se obezbeđuje povratak metka u početno stanje. U toj poziciji će se nalaziti sve dok se ponovo ne pojavi pneumatski impuls u aktuatoru.

### Kontinualan rad aktuatora

Svi procesi koji se dešavaju između pogona projektila i vraćanja aktuatora u početno stanje čine jedan ciklus njegovog radnog režima. Da bi se obezbedila ponovljivost izlaznog signala neophodno je da se svaki ciklus završi pre početka naredne periode pneumatske impulsno širinske modulacije.

Namena izvšnog organa je generisanje između 2000 i 2200 impulsa kompresionih talasa na svom izlazu za vreme jednog neprekidnog radnog režima. Željena frekvencija kojom se odvijaju ciklusi je između 8 i 10 *Hz*. Navedene vrednosti bazirane su na analizi koje su obavljene u radu [1].

## **Upravljačka kutija**

Podešavanje karakteristika upravljačkih signala razvodnog ventila omogućeno je posredstvom korisničkog interfejsa koji se sastoji od električnih i pneumatskih komponenti. Interfejs se nalazi na upravljačkoj kutiji koja je prikazana na slici 31. Sve hardverske komponente sa kojima korisnik može direktno interagovati su naznačene osim tastera zbog preglednosti. Oni su posebno izvojeni na slici 32.



Slika 31 - Upravljačka kutija

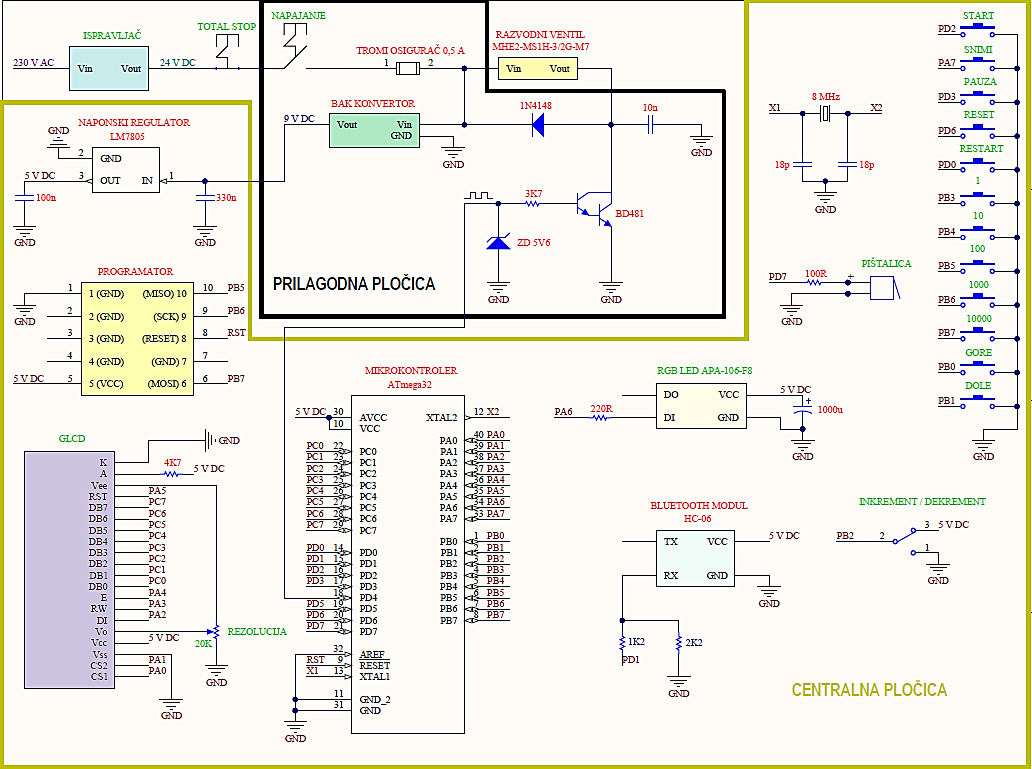


Slika 32 - Tasteri

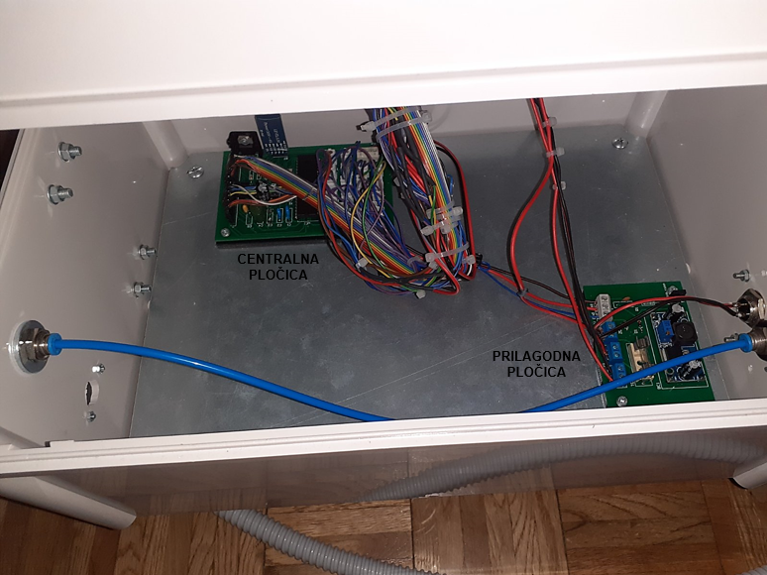
Pneumatska pripremna grupa omogućava podešavanje vazdušnog pritiska pomoću kojeg se upravlja izvršnim organom. Predstavlja jedinu neelektričnu celinu sa kojom korisnik može direktno da interaguje. Funkcije ostalih električnih komponenti usklađene su posredstvom interne elektronike upravljačke kutije. Ona sadrži mikrokontroler u kojem je implementiran firmver (*firmware*) koji sinhronizuje rad svih električnih periferija.

1. **Upravljačka elektronika**

Upravljačka elektronika sadrži dve štampane pločice različitih funkcija. Jedna objedinjuje mikrokontroler *ATmega32* i periferije koje su namenjene za interakciju sa korisnikom (centralna pločica), dok je druga namenjena za prilagođavanje napona napajanja različitih električnih komponenti (prilagodna pločica). Na slici 33 može se videti šematski prikaz upravljačke elektronike sistema. Zelenom bojom naznačeni su nazivi komponenti koji su dostupni korisniku za direktnu interakciju. Način na koji su štampane pločice montirane unutar upravljačke kutije prikazan je na slici 34.



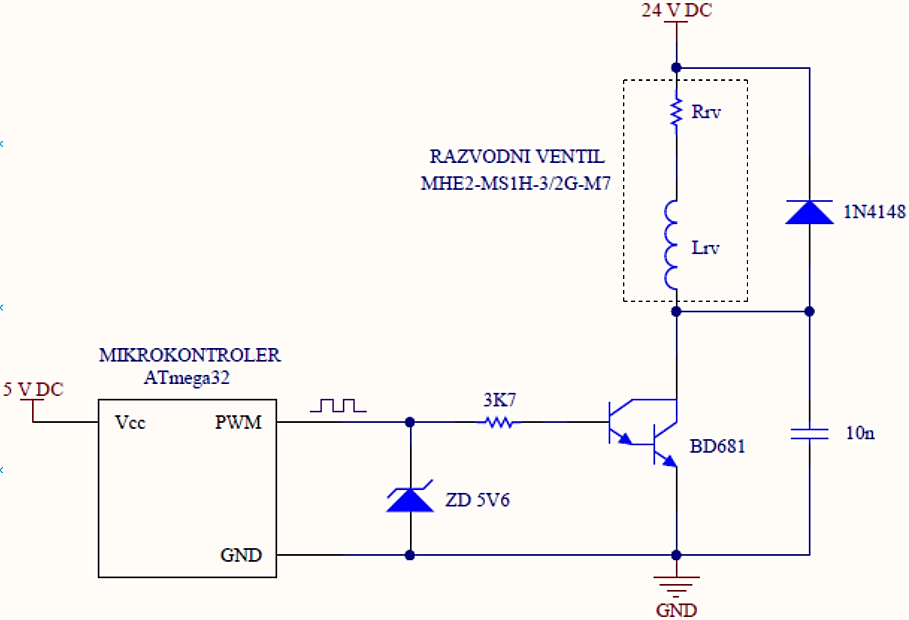
Slika 33 - Šematski prikaz upravljačke elektronike sistema



Slika 34 - Centralna i prilagodna štampana pločica

Kao što se može uočiti na slici 33, osim prekidača za napajanje postoji i *TOTAL STOP* sigurnosni prekidač kojim se direktno može prekinuti protok električne energije u sistemu. Montirani su na upravljačkoj kutiji na način koji je prikazan na slici 31. Takođe, u rednoj vezi sa njima postavljen je tromi osigurač kao vid zaštite od pojave prevelike struje u sistemu.

Električno potkolo namenjeno za upravljanje razvodnim ventilom izdvojeno je na slici 35. Električni model razvodnika predstavljen je rednom vezom otpornika i kalema [8]. Istim se upravlja posredstvom Darlingtonove sprege bipolarnih *NPN* tranzistora koji rade u prekidačkom režimu. Na bazu ulaznog tranzistora dovodi se naponska impulsno širinska modulacija sa izlaznog pina mikrokontrolera. Na taj način ostvaruje se pneumatski *PWM* signal na izlaznom vodu elektromagnetnog ventila kojim se upravlja radom izvršnog organa. S obzirom da su vremena uključenja i isključenja razvodnika veoma sliča i kratka, faktor ispune i perioda *PWM* signala na izlaznom pinu mikrokontrolera su približno ista kao i izlazna pneumatska impulsno širinska modulacija razvodnog ventila.

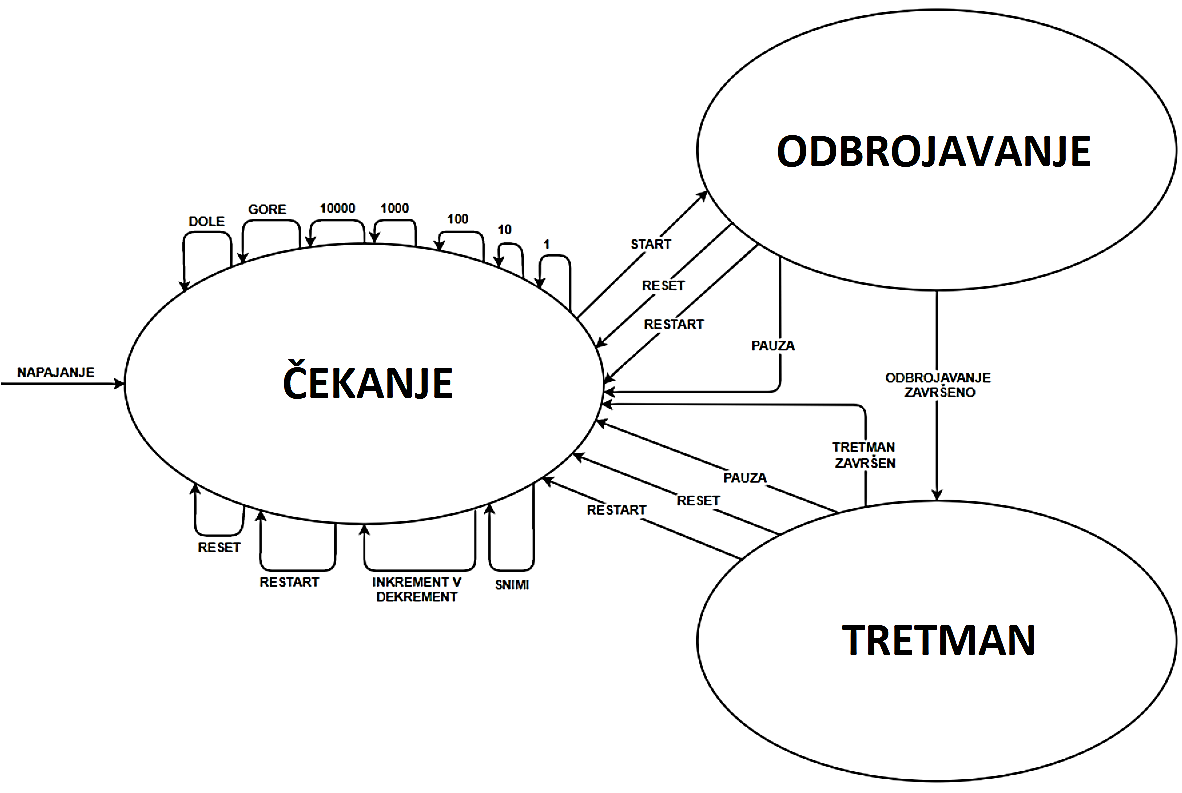


Slika 35 - Električno potkolo namenjeno za upravljanje razvodnim ventilom

Napon napajanja razvodnog ventila veći je nego napon neophodan za funkcionisanje mikrokontrolera i svih periferija centralne pločice. Pretvaranje električne energije u energiju koja omogućava rad tih komponenti vrši se primenom redne veze bak konvertora i naponskog regulatora. Međutim, vremensko usklađivanje rada celokupnog električnog segmenta sistema omogućava firmver koji je implementiran u mikrokontroleru.

1. **Firmverski algoritam**

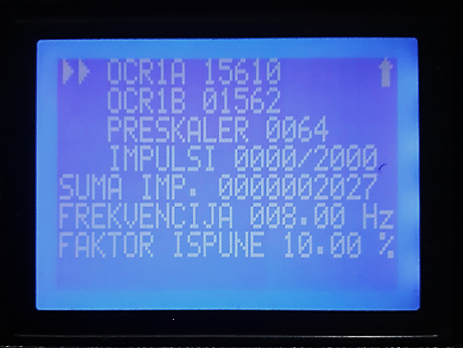
Firmverski algoritam zasnovan je na principu rada konačne mašine stanja (režima). Na slici 36 predstavljen je blok dijagram algoritma na kojem se mogu videti mogući događaji i režimi automata. Kompletan programski kod dat je u prilogu 3.



Slika 36 - Konačna mašina stanja

Kao što se može primetiti na slici 36, mašina sadrži 3 stanja. Takođe, postoji 16 različitih mogućih događaja. U zavisnosti od događaja i trenutnog režima automata, mašina će preći u naredno odgovarajuće stanje. Nalepnica ispod (ili pored) hardverske komponente sa kojom korisnik može direktno da interaguje sadrži naziv događaja koji mikrokontroler registruje ukoliko je ta komponenta okinuta.

Uspostavljanjem napajanja sistema, podešavaju se početne vrednosti internih registara mikrokontrolera i automat se postavlja u stanje *ČEKANJE*. Taj režim namenjen je za podešavanje karakteristka naponske impulsno širinske modulacije. Svi podesivi parametri su korisniku prikazani na ekranu *GLCD*-a (*graphic liquid crystal display*) kao na slici 37. Ekran *GLCD*-a ima strukturu matrice fromata 8x21. Svakom elementu matrice dodeljen je jedan karakter.



Slika 37 - Prikaz podesivih parametara naponske impulsno širinske modulacije

Frekvencija *PWM* signala zavisi od vrednosti parametara *OCR1A* i *PRESKALER*. U osnovi, oni predstavljaju vrednosti internih registara mikrokontrolera. Učestalost naponske impulsno širinske modulacije prikazuje se u šestom redu displeja i određena je izrazom:

|  |  |  |
| --- | --- | --- |
|  |  | (82) |

gde je frekvencija taktnog signala. Kako mikrokontroler koristi kristalni oscilator koji se može videti na slici 33 kao spoljašnji izvor taktnog signala, vrednost iznosi 8 *MHz*.

Faktor ispune upravljačkog singala takođe zavisi od vrednosti parametra *OCR1A*, ali i od parametra (registra) *OCR1B*. Njegova vrednost definisana je izrazom:

|  |  |  |
| --- | --- | --- |
|  |  | (83) |

i prikazuje se u sedmom redu ekrana.

Postoji mogućnost podešavanja željenog broja perioda *PWM* signala za vreme jednog radnog režima izvršnog organa. Odabrana vrednost prikazuje se sa desne strane karaktera ’/’, u četvrtom redu displeja i naziva se željeni broj impulsa. Vrednost sa leve strane tog znaka predstavlja izvršeni broj impulsa. Ona se inkrementira na svaku padajuću ivicu naponske impulsno širinske modulacije, tako da je korisnik ne može podesiti na vrednost po sopstvenom izboru. Anulira se početkom izvršavanja programskog koda.

Promena vrednosti parametara koji su prikazani na displeju vrši se posredstvom tastera i kliznog prekidača. U prvoj koloni prvog reda ekrana na slici 37 može se videti pokretni marker (kursor) čija pozicija određuje koji parametar je odabran za podešavanje. Njegov položaj može se promeniti pritiskom zelenog tastera *GORE* ili *DOLE* (odnosno uzrokovanjem događaja *GORE* ili *DOLE* u mašini stanja). Opseg mogućih pozicija kursora ograničen je na prve četiri vrste ekrana, ali redni broj kolone ostaje uvek isti.

Pritiskom bilo kojeg plavog tastera promeniće se vrednost parametra na koji pokazuje pokretni marker za odgovarajući korak. Veličina koraka koji je dodeljen svakom tasteru naznačen je brojem na nalepnici koji se može videti ispod svakog od njih. Na nalepnici se može uočiti da su mogući koraci 1, 10, 100, 1000 i 10000. Na ovaj način omogućeno je brzo i precizno podešavanje parametara.

Položaj kliznog prekidača definiše predznak koraka, odnosno određuje da li će se vrednost parametra na koji pokazuje kursor inkrementirati ili dekrementirati za korak koji je dodeljen pritisnutom plavom tasteru. Iako je nalepnicama pored kliznog prekidača naznačena funkcija koja odgovara svakom njegovom položaju, na displeju postoji indikator koji pruža informacije o trenutnom odabranom predznaku. Indikator je u vidu strele čiji smer odgovara trenutnoj funkciji prekidača i nalazi se u poslednjoj koloni prvog reda ekrana. Ukoliko je strela usmerena prema gore predznak koraka je pozitivan. U suprotnom je negativan.

Opseg mogućih vrednosti nije isti za sve parametre koji su prikazani na ekranu. Oni zavise od granica koje su određene u programskom kodu, ali i od strukture internih registra mikrokontrolera. U tabeli 2 mogu se videti moguće vrednosti svih parametara.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Tabela 2 - Opseg vrednosti podesivih parametara električnog PWM signala

Niz mogućih vrednosti parametra *PRESKALER* sadrži samo 5 različitih vrednosti. To predstavlja posledicu interne strukture registra mikrokontrolera koji sadrži bitove za definisanje vrednost delitelja učestalosti taktnog signala. Za razliku od svih ostalih podesivih parametara, vrednost *PRESKALER* moguće je promeniti samo pritiskom plavog tastera kojem je dodeljen korak 1. Svakim njegovim pritiskom promeniće se njegova vrednost na narednu vrednost niza ukoliko je indikatorska strela usmerena prema gore. U suprotnom će se promeniti na prethodnu vrednost.

Vrednost parametra *OCR1A* uvek je veća od vrednosti *OCR1B*. Ovo predstavlja ograničenje koje je implementirano u programskom kodu firmvera kako bi se uklonila mogućnost podešavanja faktora ispune na vrednost koja je jednaka ili veća od 100 *%*. Bez takvog ograničenja postojala bi mogućnost nepredviđenog ponašanja automata.

Posle podešavanja parametara, njihove vrednosti mogu se umemorisati u interni *EEPROM* mikrokontrolera pritiskom tastera *SNIMI*. Snimljene vrednosti biće na taj način standardne pri uspostavljanju napajanja sistema. Dokle god je taster *SNIMI* u zatvorenom stanju, korisniku se u osmom redu ekrana prikazuje povratna informacija koja potvrđuje da su vrednosti umemorisane.

Pritiskom tastera *RESTART* za vreme bilo kojeg režima automata aktiviraće se *watchdog* vremenski brojač mikrokontrolera. Kada odbroji do kraja posle nekoliko milisekundi, programski kod će početi da se izvršava od početka, ali će vrednosti parametara biti iste koje su bile prilikom poslednjeg pritiska tastera *SNIMI*.

Prelazak automata u režim *ODBROJAVANJE* vrši se pritiskom tastera *START*. Tada započinje odbrojavanje u trajanju od pet sekundi pre nego što mašina pređe u stanje *TRETMAN*. Odbrojavanje se prikazuje u osmom redu ekrana. Takođe, svake sekunde ozvučava se pištalica kako bi korisnik imao uvid o odbrojavanju, a da ne mora da posmatra povratne informacije na displeju.

Režim *ODBROJAVANJE* postoji u automatu kako bi korisnik mogao da spreči aktivaciju radnog režima izvršnog organa u slučaju da je slučajno pritisnuo taster *START*. Povratak u stanje *ČEKANJE* pre kraja odbrojavanjamoguće je pritiskom tastera *PAUZA*, *RESET* ili *RESTART*.

Posle završetka odbrojavanja, mašina prelazi u stanje *TRETMAN*. U tom trenutku formira se *PWM* signal na izlaznom pinu mikrokontrolera koji je spregnut sa bazom ulaznog tranzistora Darlingtonovog spoja. Naponska impulsno širinska modulacija postojaće dokle god je izvršeni broj impulsa manji od željenog. Kada se te vrednosti izjednače, prestaje dejstvo upravljačkog signala i ozvučava se pištalica kako bi korisnik znao da se automat vraća u režim *ČEKANJE* za nekoliko sekundi. Takođe, u osmom redu ekrana će se pojaviti povratna informacija koja potvrđuje kraj režima *TRETMAN*.

Pritiskom bilo kojeg tastera kojim se može prekinuti odbrojavanje, može se obustaviti i dejstvo upravljačkog signala. Ukoliko je režim *TRETMAN* ili *ODBROJAVANJE* prekinut pritiskom tastera *PAUZA*, automat će preći u režim *ČEKANJE*, a izvršeni broj impulsa će ostati nepromenjen. Kada sledeći put mašina pređe u stanje *TRETMAN*, impulsi će nastaviti da se broje od te vrednosti. Za razliku od tastera *PAUZA*, ako se prelazak iz bilo kojeg režima u stanje *ČEKANJE* izvrši pritiskom tastera *RESET*, izvršeni broj impulsa će se anulirati.

Za vreme režima *TRETMAN*, osim broja izvršenih impulsa inkrementira se i suma impulsa koja je prikazana u petom redu na ekranu. Ta vrednost se pri završetku stanja *TRETMAN* skladišti u *EEPROM* mikrokontroler kako bi se omogućilo neprekidno brojanje radnih ciklusa izvršnog organa. Suma impulsa može se anulirati držanjem tastera *RESET* u zatvorenom stanju duže od pet sekundi bez prekida. Odbrojavanje se prikazuje u osmom redu displeja, posle kojeg se ispisuje povratna informacija korisniku da je vrednost uspešno anulirana.

Pored ekrana i pištalice, povratne informacije o mašini stanja pruža RGB svetleća dioda. Svaki režim karakteriše drugačija boja svetljenja diode. Kao indikacija da je automat u režimu *ČEKANJE*, dioda svetli crvenom bojom. Analogno, dioda svetli žutom bojom dok je mašina u stanju *ODBROJAVANJE*, a zelenom bojom ako je u režimu *TRETMAN*.

Prilikom početka izvršavanja programskog koda, podešavaju se interni registri mikrokontrolera. Za vreme tog procesa, dioda treperi belom bojom. Kada se registri podese, ozvučiće se pištalica kao indikacija da su početni procesi završeni, a mašina će se postaviti u svoje početno stanje. Tada će dioda početi da svetli bojom koja je karakteristična za taj režim.

Ukoliko automat registruje da je neki taster pritisnut ozvučiće se kratko pištalica, a dioda će početi da svetli tamno plavom bojom i ostaće te boje dokle god je taster u zatvorenom stanju. Izuzetak predstavlja pritisak tastera *RESET*. Tada dioda svetli svetlo plavom bojom dok se ne anulira vrednost sume impulsa. Posle toga nastavlja da svetli tamno plavom bojom ukoliko taster i dalje nije otpušten. Prelaskom registrovanog tastera u otvoreno stanje dioda će početi da svetli bojom koja je u skladu sa trenutnim režimom mašine.

Postoji kratak vremenski interval od nekoliko sekundi koji je namenjen da signalizira korisniku kraj radnog režima izvršnog organa ukoliko se broj izvršenih i željenih impulsa izjednači dok je automat u stanju *TRETMAN*. Za to vreme ozvučava se pištalica, ali i dioda svetli ružičastom bojom. Prelaskom mašine u stanje *ČEKANJE*, dioda počinje ponovo da svetli crvenom bojom.

# **Eksperimenti**

Nastaviti pisanje ovde

1. **Zaključak**

# **Literatura**

[1] Tanasijan L. (2011), *Terapija udarnim talasima*, diplomski rad, Novi Sad: Univerzitet u Novom Sadu, Prirodno-matematički fakultet.

[2] Belić S. Dragoljub (1994), *Fizika I*, Beograd: Univerzitet u Beogradu, Fakultet za fizičku hemiju.

[3] Verbič B. (1989), *Otpornost materijala*, Sarajevo: Univerzitet u Sarajevu, Građevinski fakultet.

[4] Pavlović Š. Dragana, Mijić M. (2017), *Elektroakustika*, Beograd: Univerzitet u Beogradu, Elektrotehnički fakultet.

[5] Cvetić J. (2003), *Talasi*, Beograd: Univerzitet u Beogradu, Elektrotehnički fakultet.

[6] *The wave equation for sound*, internet, dostupno na: https://www.animations.physics.unsw.edu.au/jw/sound-wave-equation.htm (datum pristupa: 31.07.2020.).

[7] *Lecture 15: Sound Waves: Sound Speed*, internet, dostupno na: <https://sites.ualberta.ca/~pogosyan/teaching/PHYS_130/FALL_2010/lectures/lect15/lecture15.html> (datum pristupa: 31.07.2020.).

[8] Čajetinac S. (2012), *Prilog razvoju modela odlučivanja za izbor elektropneumatskog upravljanja*, doktorska disertacija, Novi Sad: Univerzitet u Novom Sadu, Fakultet tehničkih nauka.

[9] Camacho, Juan Manuel & Sosa, Victor (2013), *Alternative method to calculate the magnetic field of permanent magnets with azimuthal symmetry*, Mexican journal of physics, vol. 59, pp. 8-17.

[10] Šimonji J. (2008), *Kretanje tela na strmoj ravni*, diplomski rad, Novi Sad: Univerzitet u Novom Sadu, Prirodno-matematički fakultet.

[11] Ueberle, Friedrich & Rad, Abtin (2012), *Ballistic Pain Therapy Devices: Measurement of Pressure Pulse Parameters*, Biomedical engineering, vol. 57, pp. 700-703

[12] Spasić D. (2015), *Mehanika*, Novi Sad: Univerzitet u Novom Sadu, Fakultet tehničkih nauka.

[13] Al-Bender, Farid & Colombo, Federico & Reynaerts, Dominiek & Villavicencio, R. & Waumans, Tobias (2017), *Dynamic Characterization of Rubber O-Rings: Squeeze and Size Effects*, Advances in Tribology, vol. 2017, pp. 1-12.

[14] Miltenović V. (2009), *Mašinski elementi - oblici, proračun, primena*, Niš: Univerzitet u Nišu, Mašinski fakultet.

[15] Liu, Yajun & Chen, Xiaodong & Guo, Anyi & Liu, Sijin & Hu, Guoqing (2018), *Extracorporeal Shock Wave Therapy: Quantitative Assessments of Mechanical Responses upon Radial Extracorporeal Shock Wave Therapy*, Advanced Science, vol. 5, pp. 1-12.

# **Prilog**

Prilog 1:

Prilog 2:

Prilog 3:

/\* main.c \*/

#include <avr/interrupt.h>

#include "common.h"

#include "ports.h"

#include "USART.h"

#include "timer\_0.h"

#include "PWM\_1.h"

#include "PWM\_2.h"

#include "GLCD.h"

#include "watch\_dog\_timer.h"

#include "buzzer.h"

#include "buttons\_and\_switches.h"

#include "RGB\_LED.h"

#include "state\_machine.h"

int main(void){

ports\_initialize();

USART\_initialize();

timer\_0\_initialize();

PWM\_1\_initialize();

PWM\_2\_initialize();

GLCD\_initialize();

buttons\_and\_switches\_initialize();

watch\_dog\_timer\_initialize();

RGB\_LED\_initialize();

state\_machine\_initialize();

sei();

buzzer\_initialize();

USART\_print\_line("APLIKACIJA JE U FUNKCIJI");

while(INFINITE\_LOOP){

state\_machine\_routine();

}

//return 0;

}

/\* common.h \*/

#ifndef COMMON\_H\_

#define COMMON\_H\_

#define INFINITE\_LOOP 1

unsigned char common\_get\_digit(unsigned char digit);

unsigned char common\_get\_tenth\_digit(*uint32\_t* number);

unsigned char common\_get\_ninth\_digit(*uint32\_t* number);

unsigned char common\_get\_eighth\_digit(*uint32\_t* number);

unsigned char common\_get\_seventh\_digit(*uint32\_t* number);

unsigned char common\_get\_sixth\_digit(*uint32\_t* number);

unsigned char common\_get\_fifth\_digit(*uint32\_t* number);

unsigned char common\_get\_fourth\_digit(*uint32\_t* number);

unsigned char common\_get\_third\_digit(*uint32\_t* number);

unsigned char common\_get\_second\_digit(*uint32\_t* number);

unsigned char common\_get\_first\_digit(*uint32\_t* number);

#endif /\* COMMON\_H\_ \*/

/\* common.h \*/

#include <avr/io.h>

#include "common.h"

unsigned char common\_get\_digit(unsigned char digit){

unsigned char character = '0';

switch(digit){

case 1:{

character = '1';

break;

}

case 2:{

character = '2';

break;

}

case 3:{

character = '3';

break;

}

case 4:{

character = '4';

break;

}

case 5:{

character = '5';

break;

}

case 6:{

character = '6';

break;

}

case 7:{

character = '7';

break;

}

case 8:{

character = '8';

break;

}

case 9:{

character = '9';

break;

}

}

return character;

}

unsigned char common\_get\_tenth\_digit(*uint32\_t* number){

return common\_get\_digit(number / 1000000000);

}

unsigned char common\_get\_ninth\_digit(*uint32\_t* number){

return common\_get\_digit((number % 1000000000) / 100000000);

}

unsigned char common\_get\_eighth\_digit(*uint32\_t* number){

return common\_get\_digit((number % 100000000) / 10000000);

}

unsigned char common\_get\_seventh\_digit(*uint32\_t* number){

return common\_get\_digit((number % 10000000) / 1000000);

}

unsigned char common\_get\_sixth\_digit(*uint32\_t* number){

return common\_get\_digit((number % 1000000) / 100000);

}

unsigned char common\_get\_fifth\_digit(*uint32\_t* number){

return common\_get\_digit((number % 100000) / 10000);

}

unsigned char common\_get\_fourth\_digit(*uint32\_t* number){

return common\_get\_digit((number % 10000) / 1000);

}

unsigned char common\_get\_third\_digit(*uint32\_t* number){

return common\_get\_digit((number % 1000) / 100);

}

unsigned char common\_get\_second\_digit(*uint32\_t* number){

return common\_get\_digit((number % 100) / 10);

}

unsigned char common\_get\_first\_digit(*uint32\_t* number){

return common\_get\_digit(number % 10);

}

/\* ports.h \*/

#ifndef PORTS\_H\_

#define PORTS\_H\_

#include <stdint-gcc.h>

#include <avr/io.h>

#define SET\_BIT(PORTx,n) PORTx|=(1<<n)

#define RESET\_BIT(PORTx,n) PORTx&=~(1<<n)

#define TOGGLE\_BIT(PORTx,n) PORTx^=(1<<n)

#define READ\_PIN(PINx,n) PINx&(1<<n)

#define PORTA\_INITIAL\_DIRECTION 0b01111111

#define PORTB\_INITIAL\_DIRECTION 0b00000000

#define PORTC\_INITIAL\_DIRECTION 0b11111111

#define PORTD\_INITIAL\_DIRECTION 0b10110010

#define PORTA\_INITIAL\_STATE 0b10000000

#define PORTB\_INITIAL\_STATE 0b11111111

#define PORTC\_INITIAL\_STATE 0b00000000

#define PORTD\_INITIAL\_STATE 0b01001101

#define A0 0

#define A1 1

#define A2 2

#define A3 3

#define A4 4

#define A5 5

#define A6 6

#define A7 7

#define B0 0

#define B1 1

#define B2 2

#define B3 3

#define B4 4

#define B5 5

#define B6 6

#define B7 7

#define C0 0

#define C1 1

#define C2 2

#define C3 3

#define C4 4

#define C5 5

#define C6 6

#define C7 7

#define D0 0

#define D1 1

#define D2 2

#define D3 3

#define D4 4

#define D5 5

#define D6 6

#define D7 7

void ports\_initialize(void);

void ports\_set\_PINxn\_of\_PORTA\_direction\_to\_output(*uint8\_t* xn);

void ports\_set\_PINxn\_of\_PORTA\_direction\_to\_input(*uint8\_t* xn);

void ports\_set\_PINxn\_of\_PORTB\_direction\_to\_output(*uint8\_t* xn);

void ports\_set\_PINxn\_of\_PORTB\_direction\_to\_input(*uint8\_t* xn);

void ports\_set\_PINxn\_of\_PORTC\_direction\_to\_output(*uint8\_t* xn);

void ports\_set\_PINxn\_of\_PORTC\_direction\_to\_input(*uint8\_t* xn);

void ports\_set\_PINxn\_of\_PORTD\_direction\_to\_output(*uint8\_t* xn);

void ports\_set\_PINxn\_of\_PORTD\_direction\_to\_input(*uint8\_t* xn);

*uint8\_t* ports\_read\_port\_A(void);

*uint8\_t* ports\_read\_port\_B(void);

*uint8\_t* ports\_read\_port\_C(void);

*uint8\_t* ports\_read\_port\_D(void);

void ports\_set\_OC1A\_pin\_direction\_to\_output(void);

void ports\_set\_OC2\_pin\_direction\_to\_output(void);

void ports\_set\_OC1A\_pin\_direction\_to\_input(void);

#endif /\* PORTS\_H\_ \*/

/\* \*\*\*PINOUT\*\*\*

PORTA

PA0 - OUTPUT (CS1 PIN OF THE GLCD)

PA1 - OUTPUT (CS2 PIN OF THE GLCD)

PA2 - OUTPUT (DI (RS) PIN OF THE GLCD)

PA3 - OUTPUT (RW PIN OF THE GLCD)

PA4 - OUTPUT (E (EN) PIN OF THE GLCD)

PA5 - OUTPUT (RST PIN OF THE GLCD)

PA6 - OUTPUT (RGB LED DATA INPUT PIN)

PA7 - INPUT (BUTTON SAVE PARAMETERS)

PORTB

PB0 - INPUT (BUTTON GLCD CURSOR UP)

PB1 - INPUT (BUTTON GLCD CURSOR DOWN)

PB2 - INPUT (INCREMENT / DECREMENT)

PB3 - INPUT (BY 1)

PB4 - INPUT (BY 10)

PB5 - INPUT (BY 100)

PB6 - INPUT (BY 1000)

PB7 - INPUT (BY 10000)

PORTC

PC0 - OUTPUT (D0 PIN OF THE GRAPHIC LCD DISPLAY)

PC1 - OUTPUT (D1 PIN OF THE GRAPHIC LCD DISPLAY)

PC2 - OUTPUT (D2 PIN OF THE GRAPHIC LCD DISPLAY)

PC3 - OUTPUT (D3 PIN OF THE GRAPHIC LCD DISPLAY)

PC4 - OUTPUT (D4 PIN OF THE GRAPHIC LCD DISPLAY)

PC5 - OUTPUT (D5 PIN OF THE GRAPHIC LCD DISPLAY)

PC6 - OUTPUT (D6 PIN OF THE GRAPHIC LCD DISPLAY)

PC7 - OUTPUT (D7 PIN OF THE GRAPHIC LCD DISPLAY)

PORTD

PD0 - INPUT (BUTTON RESTART)

PD1 - OUTPUT (TXD PIN OF USART)

PD2 - INPUT (BUTTON START)

PD3 - INPUT (BUTTON PAUSE)

PD4 - OUTPUT (PWM SIGNAL FOR CONTROLING THE ELECTROPNEUMATIC SOLENOID VALVE)

PD5 - OUTPUT (TOP VALUE FOR PWM 1)

PD6 - INPUT (BUTTON RESET)

PD7 - OUTPUT (PWM SIGNAL FOR CONTROLING THE BUZZER)

\*/

/\* ports.c \*/

#include "ports.h"

void ports\_initialize(void){

// output pin direction => 1

// input pin direction => 0

DDRA = PORTA\_INITIAL\_DIRECTION;

PORTA = PORTA\_INITIAL\_STATE;

DDRB = PORTB\_INITIAL\_DIRECTION;

PORTB = PORTB\_INITIAL\_STATE;

DDRC = PORTC\_INITIAL\_DIRECTION;

PORTC = PORTC\_INITIAL\_STATE;

DDRD = PORTD\_INITIAL\_DIRECTION;

PORTD = PORTD\_INITIAL\_STATE;

}

void ports\_set\_PINxn\_of\_PORTA\_direction\_to\_output(*uint8\_t* xn){

SET\_BIT(DDRA, xn);

}

void ports\_set\_PINxn\_of\_PORTA\_direction\_to\_input(*uint8\_t* xn){

RESET\_BIT(DDRA, xn);

}

void ports\_set\_PINxn\_of\_PORTB\_direction\_to\_output(*uint8\_t* xn){

SET\_BIT(DDRB, xn);

}

void ports\_set\_PINxn\_of\_PORTB\_direction\_to\_input(*uint8\_t* xn){

RESET\_BIT(DDRB, xn);

}

void ports\_set\_PINxn\_of\_PORTC\_direction\_to\_output(*uint8\_t* xn){

SET\_BIT(DDRC, xn);

}

void ports\_set\_PINxn\_of\_PORTC\_direction\_to\_input(*uint8\_t* xn){

RESET\_BIT(DDRC, xn);

}

void ports\_set\_PINxn\_of\_PORTD\_direction\_to\_output(*uint8\_t* xn){

SET\_BIT(DDRD, xn);

}

void ports\_set\_PINxn\_of\_PORTD\_direction\_to\_input(*uint8\_t* xn){

RESET\_BIT(DDRD, xn);

}

*uint8\_t* ports\_read\_port\_A(void){

*uint8\_t* port\_state = 0;

for(*uint8\_t* i = 0; i < 7; i++)port\_state |= (READ\_PIN(PINA, i) << i);

return port\_state;

}

*uint8\_t* ports\_read\_port\_B(void){

*uint8\_t* port\_state = 0;

for(*uint8\_t* i = 0; i < 7; i++)port\_state |= (READ\_PIN(PINB, i) << i);

return port\_state;

}

*uint8\_t* ports\_read\_port\_C(void){

*uint8\_t* port\_state = 0;

for(*uint8\_t* i = 0; i < 7; i++)port\_state |= (READ\_PIN(PINC, i) << i);

return port\_state;

}

*uint8\_t* ports\_read\_port\_D(void){

*uint8\_t* port\_state = 0;

for(*uint8\_t* i = 0; i < 7; i++)port\_state |= (READ\_PIN(PIND, i) << i);

return port\_state;

}

void ports\_set\_OC1A\_pin\_direction\_to\_output(void){

ports\_set\_PINxn\_of\_PORTD\_direction\_to\_output(D5);

}

void ports\_set\_OC2\_pin\_direction\_to\_output(void){

ports\_set\_PINxn\_of\_PORTD\_direction\_to\_output(D7);

}

void ports\_set\_OC1A\_pin\_direction\_to\_input(void){

ports\_set\_PINxn\_of\_PORTD\_direction\_to\_input(D5);

}

/\* USART.h \*/

#ifndef USART\_H\_

#define USART\_H\_

#include <avr/io.h>

#define F\_CPU 8000000UL

#define BAUD\_PRESCALE (((F\_CPU / (USART\_BAUDRATE \* 16UL))) - 1)

#define NEW\_LINE 10

#define ENTER 13

extern char string[20];

void USART\_initialize(void);

void USART\_initialize\_with\_baud\_rate(long USART\_BAUDRATE);

char USART\_get\_character(void);

char\* USART\_get\_string(void);

void USART\_print\_character(char character);

void USART\_print(char \*string);

void USART\_print\_line(char \*string);

void USART\_print\_number(*uint32\_t* number);

void USART\_print\_float(float float\_number);

#endif /\* USART\_H\_ \*/

/\* USART.c \*/

#include <avr/io.h>

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include "common.h"

#include "USART.h"

char string[20];

void USART\_initialize(void){

USART\_initialize\_with\_baud\_rate(9600);

}

void USART\_initialize\_with\_baud\_rate(long USART\_BAUDRATE){

UCSRB |= (1 << TXEN) /\*| (1 << RXEN)\*/;

UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);

UBRRL = BAUD\_PRESCALE;

UBRRH = (BAUD\_PRESCALE >> 8);

}

char USART\_get\_character(void){

while((UCSRA & (1 << RXC)) == 0);

return(UDR);

}

char\* USART\_get\_string(void){

char character = '\0';

*uint8\_t* iterator = 0;

do{

character = USART\_get\_character();

}while(character == NEW\_LINE);

while(character != ENTER){

string[iterator] = character;

iterator++;

character = USART\_get\_character();

}

string[iterator] = '\0';

return(string);

}

void USART\_print\_character(char character){

while(!(UCSRA & (1 << UDRE)));

UDR = character;

}

void USART\_print(char \*string){

unsigned char i = 0;

while(string[i] != 0){

USART\_print\_character(string[i]);

i++;

}

}

void USART\_print\_line(char \*string){

USART\_print(string);

USART\_print\_character('\n');

}

void USART\_print\_number(*uint32\_t* number){

char string\_number[11] = {'0','0','0','0','0','0','0','0','0','0','\0'};

string\_number[0] = common\_get\_tenth\_digit(number);

string\_number[1] = common\_get\_ninth\_digit(number);

string\_number[2] = common\_get\_eighth\_digit(number);

string\_number[3] = common\_get\_seventh\_digit(number);

string\_number[4] = common\_get\_sixth\_digit(number);

string\_number[5] = common\_get\_fifth\_digit(number);

string\_number[6] = common\_get\_fourth\_digit(number);

string\_number[7] = common\_get\_third\_digit(number);

string\_number[8] = common\_get\_second\_digit(number);

string\_number[9] = common\_get\_first\_digit(number);

if(string\_number[0] == '0' && string\_number[1] == '0' && string\_number[2] == '0'  
 && string\_number[3] == '0' && string\_number[4] == '0' && string\_number[5] == '0'  
 && string\_number[6] == '0' && string\_number[7] == '0' && string\_number[8] == '0'){

char\* new\_string\_number = &string\_number[9];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0' && string\_number[1] == '0' && string\_number[2] == '0'  
 && string\_number[3] == '0' && string\_number[4] == '0' && string\_number[5] == '0'  
 && string\_number[6] == '0' && string\_number[7] == '0'){

char\* new\_string\_number = &string\_number[8];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0' && string\_number[1] == '0' && string\_number[2] == '0'  
 && string\_number[3] == '0' && string\_number[4] == '0' && string\_number[5] == '0'  
 && string\_number[6] == '0'){

char\* new\_string\_number = &string\_number[7];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0' && string\_number[1] == '0' && string\_number[2] == '0'  
 && string\_number[3] == '0' && string\_number[4] == '0' && string\_number[5] == '0'){

char\* new\_string\_number = &string\_number[6];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0' && string\_number[1] == '0' && string\_number[2] == '0'  
 && string\_number[3] == '0' && string\_number[4] == '0'){

char\* new\_string\_number = &string\_number[5];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0' && string\_number[1] == '0' && string\_number[2] == '0'  
 && string\_number[3] == '0'){

char\* new\_string\_number = &string\_number[4];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0' && string\_number[1] == '0' && string\_number[2] == '0'){

char\* new\_string\_number = &string\_number[3];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0' && string\_number[1] == '0'){

char\* new\_string\_number = &string\_number[2];

USART\_print(new\_string\_number);

}

else if(string\_number[0] == '0'){

char\* new\_string\_number = &string\_number[1];

USART\_print(new\_string\_number);

}

else{

USART\_print(string\_number);

}

}

void USART\_print\_float(float float\_number){

*uint16\_t* integer\_part = *trunc*(float\_number);

float fraction\_part = float\_number - integer\_part;

*uint16\_t* after\_point\_number = (*uint16\_t*)(fraction\_part \* 100);

char string\_after\_point\_number[4] = {'0','0','\0'};

string\_after\_point\_number[0] = common\_get\_second\_digit(after\_point\_number);

string\_after\_point\_number[1] = common\_get\_first\_digit(after\_point\_number);

USART\_print\_number(integer\_part);

USART\_print\_character('.');

USART\_print(string\_after\_point\_number);

}

/\* timer\_0.h \*/

#ifndef TIMER\_0\_H\_

#define TIMER\_0\_H\_

#include <avr/io.h>

//clear timer on compare match and set OC0 on compare match

#define TCCRO\_INITIAL\_STATE 0x0B

//define bottom value

#define TCNT0\_INITIAL\_STATE 0x00

//define number of counts before the timer interrupt occurs

#define OCR0\_INITIAL\_STATE 0x7D

#define ENABLE\_TIMER\_0\_OUTPUT\_COMPARE\_MATCH\_INTERRUPT 0b00000010

#define NUMBER\_OF\_COUNTS 4

#define RESET\_SUM\_OF\_IMPULSES\_TIME\_COUNT 5

#define ONE\_SECOND 1000

extern volatile *uint32\_t* timer\_counter;

extern volatile *uint16\_t* sum\_of\_impulses\_timer;

extern volatile *uint8\_t* reset\_sum\_of\_impulses\_time\_count;

extern volatile *uint32\_t* countdown\_timer;

extern volatile *uint8\_t* number\_of\_counts;

void timer\_0\_initialize(void);

void timer\_0\_delay\_in\_milliseconds(*uint32\_t* time\_of\_delay\_in\_miliseconds);

*uint8\_t* timer\_0\_get\_reset\_sum\_of\_impulses\_time\_count(void);

void timer\_0\_reset\_sum\_of\_impulses\_time(void);

*uint8\_t* timer\_0\_reset\_sum\_of\_impulses\_time\_has\_passed(void);

void timer\_0\_reset\_countdown\_timer(void);

void timer\_0\_reset\_number\_of\_counts(void);

*uint8\_t* timer\_0\_get\_number\_of\_counts(void);

*uint8\_t* timer\_0\_countdown\_is\_over(void);

#endif /\* TIMER\_0\_H\_ \*/

/\* timer\_0.c \*/

#include <avr/interrupt.h>

#include <stdbool.h>

#include "timer\_0.h"

#include "GLCD.h"

#include "buzzer.h"

volatile *uint32\_t* timer\_counter = 0;

volatile *uint16\_t* reset\_sum\_of\_impulses\_timer = 0;

volatile *uint8\_t* reset\_sum\_of\_impulses\_time\_count = RESET\_SUM\_OF\_IMPULSES\_TIME\_COUNT;

volatile *uint32\_t* countdown\_timer = 0;

volatile *uint8\_t* number\_of\_counts = NUMBER\_OF\_COUNTS;

ISR (TIMER0\_COMP\_vect){

timer\_counter++;

countdown\_timer++;

reset\_sum\_of\_impulses\_timer++;

}

void timer\_0\_initialize(void){

TCCR0 = TCCRO\_INITIAL\_STATE;

TCNT0 = TCNT0\_INITIAL\_STATE;

OCR0 = OCR0\_INITIAL\_STATE;

TIMSK = ENABLE\_TIMER\_0\_OUTPUT\_COMPARE\_MATCH\_INTERRUPT;

}

void timer\_0\_delay\_in\_milliseconds(*uint32\_t* time\_of\_delay\_in\_milliseconds){

timer\_counter = 0;

while(timer\_counter != time\_of\_delay\_in\_milliseconds);

}

void timer\_0\_reset\_sum\_of\_impulses\_time(void){

reset\_sum\_of\_impulses\_timer = 0;

reset\_sum\_of\_impulses\_time\_count = RESET\_SUM\_OF\_IMPULSES\_TIME\_COUNT;

}

*uint8\_t* timer\_0\_get\_reset\_sum\_of\_impulses\_time\_count(void){

return reset\_sum\_of\_impulses\_time\_count;

}

*uint8\_t* timer\_0\_reset\_sum\_of\_impulses\_time\_has\_passed(void){

if(reset\_sum\_of\_impulses\_timer == ONE\_SECOND){

reset\_sum\_of\_impulses\_timer = 0;

reset\_sum\_of\_impulses\_time\_count--;

GLCD\_print\_sum\_of\_impulses\_time\_count();

}

if(!reset\_sum\_of\_impulses\_time\_count)return true;

else return false;

}

void timer\_0\_reset\_countdown\_timer(void){

countdown\_timer = 0;

}

void timer\_0\_reset\_number\_of\_counts(void){

number\_of\_counts = NUMBER\_OF\_COUNTS;

}

*uint8\_t* timer\_0\_get\_number\_of\_counts(void){

return number\_of\_counts;

}

*uint8\_t* timer\_0\_countdown\_is\_over(void){

if(countdown\_timer == ONE\_SECOND){

countdown\_timer = 0;

number\_of\_counts--;

GLCD\_print\_number\_of\_counts();

if(number\_of\_counts)buzzer\_activate\_countdown\_tone();

}

if(!number\_of\_counts)return true;

else return false;

}

/\* PWM\_1.h \*/

#ifndef PWM\_1\_H\_

#define PWM\_1\_H\_

#include <stdbool.h>

#define TCNT1H\_INITIAL\_STATE 0x00

#define TCNT1L\_INITIAL\_STATE 0x00

#define ICR1H\_INITIAL\_STATE 0x00

#define ICR1L\_INITIAL\_STATE 0x00

#define CLOCK\_FREQUENCY 8000000.0f

#define PRESCALER\_MASC 0b00000111

#define MAX\_PRESCALER\_MASC 0b00000101

#define MIN\_PRESCALER\_MASC 0b00000001

#define PRESCALER\_NO\_CLOCK\_SOURCE 0b00000000

#define PRESCALER\_IS\_1 0b00000001

#define PRESCALER\_IS\_8 0b00000010

#define PRESCALER\_IS\_64 0b00000011

#define PRESCALER\_IS\_256 0b00000100

#define PRESCALER\_IS\_1024 0b00000101

#define MIN\_DUTY\_CYCLE 5

#define MAX\_NUMBER\_OF\_IMPULSES 9999

#define MIN\_NUMBER\_OF\_IMPULSES 1

#define PRESCALER\_ADDRESS (*uint8\_t*\*)0

#define OCR1BL\_ADDRESS (*uint8\_t*\*)1

#define OCR1BH\_ADDRESS (*uint8\_t*\*)2

#define OCR1AL\_ADDRESS (*uint8\_t*\*)3

#define OCR1AH\_ADDRESS (*uint8\_t*\*)4

#define MAX\_NUMBER\_OF\_IMPULSES\_ADDRESS (*uint16\_t*\*)5

#define SUM\_OF\_IMPULSES\_ADDRESS (*uint32\_t*\*)7

extern volatile *uint16\_t* impulse\_counter;

extern volatile *uint16\_t* max\_number\_of\_impulses;

extern volatile *uint32\_t* sum\_of\_impulses;

extern volatile bool enable\_first\_impulse;

void PWM\_1\_initialize(void);

void PWM\_1\_enable\_non\_inverted\_wave\_form(void);

void PWM\_1\_disable(void);

void PWM\_1\_minimaly\_increase\_prescaler(void);

void PWM\_1\_minimaly\_decrease\_prescaler(void);

void PWM\_1\_set\_prescaler(*uint16\_t* prescaler);

*uint16\_t* PWM\_1\_get\_prescaler(void);

void PWM\_1\_minimaly\_increase\_duty\_cycle\_percentage(void);

void PWM\_1\_minimaly\_decrease\_duty\_cycle\_percentage(void);

float PWM\_1\_get\_duty\_cycle\_percentage(void);

void PWM\_1\_minimaly\_increase\_period(void);

void PWM\_1\_minimaly\_decrease\_period(void);

void PWM\_1\_minimaly\_increase\_frequency(void);

void PWM\_1\_minimaly\_decrease\_frequency(void);

float PWM\_1\_get\_frequency(void);

void PWM\_1\_minimaly\_increase\_max\_number\_of\_impulses(void);

void PWM\_1\_minimaly\_decrease\_max\_number\_of\_impulses(void);

*uint16\_t* PWM\_1\_get\_OCR1A(void);

*uint16\_t* PWM\_1\_get\_OCR1B(void);

*uint16\_t* PWM\_1\_get\_max\_number\_of\_impulses(void);

void PWM\_1\_set\_impulse\_counter(*uint16\_t* number\_of\_impulses);

void PWM\_1\_reset\_impulse\_counter(void);

*uint16\_t* PWM\_1\_get\_impulse\_counter(void);

void PWM\_1\_set\_impulse\_counter\_to\_max\_number\_of\_impulses(void);

void PWM\_1\_reset\_sum\_of\_impulses(void);

*uint32\_t* PWM\_1\_get\_sum\_of\_impulses(void);

void PWM\_1\_save\_parameters(void);

void PWM\_1\_save\_the\_sum\_of\_impulses(void);

void PWM\_1\_load\_parameters(void);

#endif /\* PWM\_1\_H\_ \*/

/\* PWM\_1.c\*/

#include <avr/interrupt.h>

#include <avr/eeprom.h>

#include <math.h>

#include <stdbool.h>

#include "common.h"

#include "ports.h"

#include "PWM\_1.h"

#include "USART.h"

#include "buttons\_and\_switches.h"

volatile *uint16\_t* impulse\_counter = 0;

volatile *uint16\_t* max\_number\_of\_impulses = 0;

volatile *uint32\_t* sum\_of\_impulses = 0;

volatile bool enable\_first\_impulse = false;

ISR (TIMER1\_COMPB\_vect){ //ovaj prekid ima veci prioritet od prekida tajmera 0

if(enable\_first\_impulse){

impulse\_counter++;

sum\_of\_impulses++;

}

else enable\_first\_impulse = true;

}

void PWM\_1\_initialize(void){

TCCR1A = (0 << COM1A1) | (0 << COM1A0) | (0 << COM1B1) | (0 << COM1B0) | (0 << FOC1A) | (0 << FOC1B) | (1 << WGM11) | (1 << WGM10);

TCCR1B = (0 << ICNC1) | (0 << ICES1) | (0 << 0) | (1 << WGM13) | (1 << WGM12) | (0 << CS12) | (0 << CS11) | (0 << CS10);

TCNT1H = TCNT1H\_INITIAL\_STATE;

TCNT1L = TCNT1L\_INITIAL\_STATE;

ICR1H = ICR1H\_INITIAL\_STATE;

ICR1L = ICR1L\_INITIAL\_STATE;

PWM\_1\_load\_parameters();

if(OCR1BH == OCR1AH && OCR1BL == OCR1AL)PWM\_1\_minimaly\_increase\_period();

if(OCR1BH == OCR1AH && OCR1BL == OCR1AL)PWM\_1\_minimaly\_decrease\_duty\_cycle\_percentage();

*uint16\_t* prescaler = PWM\_1\_get\_prescaler();

if(prescaler != 1 &&

prescaler != 8 &&

prescaler != 64 &&

prescaler != 256 &&

prescaler != 1024)PWM\_1\_set\_prescaler(8);

if(max\_number\_of\_impulses > MAX\_NUMBER\_OF\_IMPULSES || max\_number\_of\_impulses < MIN\_NUMBER\_OF\_IMPULSES)max\_number\_of\_impulses = MAX\_NUMBER\_OF\_IMPULSES;

if(sum\_of\_impulses == 4294967295)sum\_of\_impulses = 0;

}

void PWM\_1\_enable\_non\_inverted\_wave\_form(void){

cli(); //global disable all interrupts

*uint16\_t* prescaler = PWM\_1\_get\_prescaler(); //remember prescaler

PWM\_1\_set\_prescaler(0); //stop counting in TCNT1

TCNT1 = 0; //reset the counter

SET\_BIT(TIFR, OCF1B); //clearing the interrupt flag

SET\_BIT(TIMSK, OCIE1B); //enable interrupt on compare match

TCCR1A = (TCCR1A & (~(1 << COM1B0))) | (1 << COM1B1); //start PWM signal

sei(); //global disable all interrupts

PWM\_1\_set\_prescaler(prescaler); //start counting

}

void PWM\_1\_disable(void){

cli();

*uint16\_t* prescaler = PWM\_1\_get\_prescaler(); //uzimas trenutni preskaler

PWM\_1\_set\_prescaler(0); //stop counting in TCNT1

RESET\_BIT(TIMSK, OCIE1B); //disable interrupt on compare match

TCNT1 = 0; //reset the counter

TCCR1A &= ~((1 << COM1B0) | (1 << COM1B1)); //enable normal pin operation mode (no more PWM)

SET\_BIT(TIFR, OCF1B);

RESET\_BIT(PORTD, D4);

PWM\_1\_set\_prescaler(prescaler); //vratio stari preskaler

enable\_first\_impulse = false;

sei();

}

void PWM\_1\_minimaly\_increase\_prescaler(void){

if((TCCR1B & PRESCALER\_MASC) < MAX\_PRESCALER\_MASC)TCCR1B++;

}

void PWM\_1\_minimaly\_decrease\_prescaler(void){

if((TCCR1B & PRESCALER\_MASC) > MIN\_PRESCALER\_MASC)TCCR1B--;

}

void PWM\_1\_set\_prescaler(*uint16\_t* prescaler){

switch(prescaler){

case 0: {

prescaler = PRESCALER\_NO\_CLOCK\_SOURCE;

break;

}

case 1: {

prescaler = PRESCALER\_IS\_1;

break;

}

case 8: {

prescaler = PRESCALER\_IS\_8;

break;

}

case 64: {

prescaler = PRESCALER\_IS\_64;

break;

}

case 256: {

prescaler = PRESCALER\_IS\_256;

break;

}

case 1024: {

prescaler = PRESCALER\_IS\_1024;

break;

}

default: prescaler = 0;

}

TCCR1B = (TCCR1B & (~PRESCALER\_MASC)) | prescaler;

}

*uint16\_t* PWM\_1\_get\_prescaler(void){

*uint16\_t* prescaler = TCCR1B & PRESCALER\_MASC;

switch(prescaler){

case PRESCALER\_IS\_1: {

prescaler = 1;

break;

}

case PRESCALER\_IS\_8: {

prescaler = 8;

break;

}

case PRESCALER\_IS\_64: {

prescaler = 64;

break;

}

case PRESCALER\_IS\_256: {

prescaler = 256;

break;

}

case PRESCALER\_IS\_1024: {

prescaler = 1024;

break;

}

default: {

prescaler = 0;

}

}

return prescaler;

}

void PWM\_1\_minimaly\_increase\_duty\_cycle\_percentage(void){

if(OCR1BH != OCR1AH || OCR1BL != OCR1AL){

if(OCR1BH < 0xFF && OCR1BL == 0xFF ){

OCR1BH++;

OCR1BL = 0x00;

}

else{

OCR1BH = OCR1BH;

OCR1BL++;

}

}

if(OCR1BL == OCR1AL && OCR1BH == OCR1AH){

OCR1BH = OCR1BH;

OCR1BL--;

}

}

void PWM\_1\_minimaly\_decrease\_duty\_cycle\_percentage(void){

if(OCR1BH || OCR1BL > MIN\_DUTY\_CYCLE){

if(OCR1BH > 0x00 && OCR1BL == 0x00 ){

OCR1BH --;

OCR1BL = 0xFF;

}

else{

OCR1BH = OCR1BH;

OCR1BL--;

}

}

}

float PWM\_1\_get\_duty\_cycle\_percentage(void){

*uint16\_t* duty\_cycle = ((*uint16\_t*)OCR1BL & 0x00FF) | ((*uint16\_t*)OCR1BH << 8);

*uint16\_t* period = ((*uint16\_t*)OCR1AL & 0x00FF) | ((*uint16\_t*)OCR1AH << 8);

return (((float)duty\_cycle \* 100.00f / (float)period));

}

void PWM\_1\_minimaly\_increase\_period(void){

if(OCR1AH != 0xFF || OCR1AL != 0xFF){

if(OCR1AH < 0xFF && OCR1AL == 0xFF ){

OCR1AH++;

OCR1AL = 0x00;

}

else{

OCR1AH = OCR1AH;

OCR1AL++;

}

}

}

void PWM\_1\_minimaly\_decrease\_period(void){

if(OCR1AH != OCR1BH || OCR1AL != OCR1BL){

if(OCR1AH > 0x00 && OCR1AL == 0x00 ){

OCR1AH --;

OCR1AL = 0xFF;

}

else{

OCR1AH = OCR1AH;

OCR1AL--;

}

}

if(OCR1AL == OCR1BL && OCR1AH == OCR1BH){

OCR1AH = OCR1AH;

OCR1AL++;

}

}

void PWM\_1\_minimaly\_increase\_frequency(void){

PWM\_1\_minimaly\_decrease\_period();

}

void PWM\_1\_minimaly\_decrease\_frequency(void){

PWM\_1\_minimaly\_increase\_period();

}

float PWM\_1\_get\_frequency(void){

*uint32\_t* TOP = ((*uint16\_t*)OCR1AL | (*uint16\_t*)OCR1AH << 8);

*uint32\_t* prescaler = PWM\_1\_get\_prescaler();

*uint32\_t* factor\_1 = 1;

*uint32\_t* denominator = (prescaler \* (factor\_1 + TOP));

return (CLOCK\_FREQUENCY / denominator);

}

void PWM\_1\_minimaly\_increase\_max\_number\_of\_impulses(void){

if(max\_number\_of\_impulses < MAX\_NUMBER\_OF\_IMPULSES){

max\_number\_of\_impulses++;

}

}

void PWM\_1\_minimaly\_decrease\_max\_number\_of\_impulses(void){

if(max\_number\_of\_impulses > MIN\_NUMBER\_OF\_IMPULSES){

max\_number\_of\_impulses--;

}

}

*uint16\_t* PWM\_1\_get\_OCR1A(void){

*uint16\_t* lower\_8\_bits = OCR1AL;

*uint16\_t* higher\_8\_bits = OCR1AH;

return ((higher\_8\_bits << 8) | lower\_8\_bits);

}

*uint16\_t* PWM\_1\_get\_OCR1B(void){

*uint16\_t* lower\_8\_bits = OCR1BL;

*uint16\_t* higher\_8\_bits = OCR1BH;

return ((higher\_8\_bits << 8) | lower\_8\_bits);

}

*uint16\_t* PWM\_1\_get\_max\_number\_of\_impulses(void){

return max\_number\_of\_impulses;

}

void PWM\_1\_set\_impulse\_counter(*uint16\_t* number\_of\_impulses){

impulse\_counter = number\_of\_impulses;

}

void PWM\_1\_reset\_impulse\_counter(void){

impulse\_counter = 0;

}

*uint16\_t* PWM\_1\_get\_impulse\_counter(void){

return impulse\_counter;

}

void PWM\_1\_set\_impulse\_counter\_to\_max\_number\_of\_impulses(void){

impulse\_counter = max\_number\_of\_impulses;

}

void PWM\_1\_reset\_sum\_of\_impulses(void){

sum\_of\_impulses = 0;

PWM\_1\_save\_the\_sum\_of\_impulses();

}

*uint32\_t* PWM\_1\_get\_sum\_of\_impulses(void){

return sum\_of\_impulses;

}

void PWM\_1\_save\_parameters(void){

eeprom\_update\_byte(PRESCALER\_ADDRESS, (TCCR1B & PRESCALER\_MASC));

eeprom\_update\_byte(OCR1BL\_ADDRESS, OCR1BL);

eeprom\_update\_byte(OCR1BH\_ADDRESS, OCR1BH);

eeprom\_update\_byte(OCR1AL\_ADDRESS, OCR1AL);

eeprom\_update\_byte(OCR1AH\_ADDRESS, OCR1AH);

eeprom\_update\_word(MAX\_NUMBER\_OF\_IMPULSES\_ADDRESS, max\_number\_of\_impulses);

}

void PWM\_1\_save\_the\_sum\_of\_impulses(void){

eeprom\_update\_dword(SUM\_OF\_IMPULSES\_ADDRESS, sum\_of\_impulses);

}

void PWM\_1\_load\_parameters(void){

TCCR1B |= eeprom\_read\_byte(PRESCALER\_ADDRESS);

OCR1BH = eeprom\_read\_byte(OCR1BH\_ADDRESS);

OCR1BL = eeprom\_read\_byte(OCR1BL\_ADDRESS);

OCR1AH = eeprom\_read\_byte(OCR1AH\_ADDRESS);

OCR1AL = eeprom\_read\_byte(OCR1AL\_ADDRESS);

max\_number\_of\_impulses = eeprom\_read\_word(MAX\_NUMBER\_OF\_IMPULSES\_ADDRESS);

sum\_of\_impulses = eeprom\_read\_dword(SUM\_OF\_IMPULSES\_ADDRESS);

}

/\*

Ako je COM1A1 resetovan i COM1A0 resetovan, diskonektovan je OC1A.

Ako je COM1B1 resetovan i COM1B0 resetovan, pin OC1B je diskonektovan i ima  
 standardnu I/O funkciju

FOC1A i FOC1B trebaju biti nula ako se koristi PWM. To su force output compare  
 bitovi, nicemu ne sluze za pwm

Ako su WGM13, WGM12, WGM11 i WGM10 setovani, onda je OCR1A vrednost do koje brojac  
 broji pre reseta

ICNC1 i ICES1 su za input noise capture, tako da su resetovani

CS12, CS11 i CS10 definisu preskaler

\*/

/\* PWM\_2.h \*/

#ifndef PWM\_2\_H\_

#define PWM\_2\_H\_

#define PRESCALER 8.0f

#define FAST\_PWM\_WITH\_PRESCALER\_8\_AND\_TURNED\_OFF 0b01001010

#define TCCR2\_INITIAL\_STATE FAST\_PWM\_WITH\_PRESCALER\_8\_AND\_TURNED\_OFF

void PWM\_2\_initialize(void);

void PWM\_2\_enable\_non\_inverted\_wave\_form(void);

void PWM\_2\_disable(void);

void PWM\_2\_set\_duty\_cycle\_percentage(float percantage);

float PWM\_2\_get\_duty\_cycle\_percentage(void);

void PWM\_2\_minimaly\_increase\_duty\_cycle\_percantage(void);

void PWM\_2\_minimaly\_decrease\_duty\_cycle\_percentage(void);

#endif /\* PWM\_2\_H\_ \*/

/\* PWM\_2.c \*/

#include <math.h>

#include "ports.h"

#include "PWM\_2.h"

#include "buzzer.h"

void PWM\_2\_initialize(void){

ports\_set\_OC2\_pin\_direction\_to\_output();

TCCR2 = TCCR2\_INITIAL\_STATE;

}

void PWM\_2\_enable\_non\_inverted\_wave\_form(void){

TCCR2 = (TCCR2 & (~(1 << COM20))) | (1 << COM21);

}

void PWM\_2\_disable(void){

TCCR2 &= ~((1 << COM20) | (1 << COM21));

RESET\_BIT(PORTD, D7);

}

void PWM\_2\_set\_duty\_cycle\_percentage(float percantage){

OCR2 = (*uint8\_t*)(*round*(percantage \* 255.0f / 100.0f));

}

float PWM\_2\_get\_duty\_cycle\_percentage(void){

return ((float)OCR2 \* 100.0f / 255.0f);

}

void PWM\_2\_minimaly\_increase\_duty\_cycle\_percantage(void){

if(OCR2 < 255)OCR2++;

}

void PWM\_2\_minimaly\_decrease\_duty\_cycle\_percantage(void){

if(OCR2 > 0)OCR2--;

}

/\* GLCD.h \*/

#ifndef GLCD\_H\_

#define GLCD\_H\_

#include <avr/io.h>

//ports

#define CS1\_PORT PORTA

#define CS2\_PORT PORTA

#define RS\_PORT PORTA

#define RW\_PORT PORTA

#define E\_PORT PORTA

#define RST\_PORT PORTA

#define CONTROL\_PORT PORTA

#define DATA\_PORT PORTC

//port directions

#define CONTROL\_PORT\_DIRECTION DDRA

#define DATA\_PORT\_DIRECTION DDRC

//pin numbers

#define CS1 0

#define CS2 1

#define RS 2

#define RW 3

#define EN 4

#define RST 5

#define DB0 0

#define DB1 1

#define DB2 2

#define DB3 3

#define DB4 4

#define DB5 5

#define DB6 6

#define DB7 7

#define RIGHT 0

#define LEFT 1

#define BUSY 0b10000000

#define PAGE\_ADDRESS 0b10111000

#define Y\_ADDRESS 0b01000000

#define STARTING\_LINE 0b11000000

#define DISPLAY\_ON 0b00111111

#define DISPLAY\_OFF 0b00111110

#define FONT\_LENGHT 5

#define COLUMN\_OFFSET 6

#define MAX\_CURSOR\_POSITION 3

#define MIN\_CURSOR\_POSITION 0

#define OCR1A\_CURSOR\_POSITION 0

#define OCR1B\_CURSOR\_POSITION 1

#define PRESCALER\_CURSOR\_POSITION 2

#define NUMBER\_OF\_IMPULSES\_CURSOR\_POSITION 3

#define STROBE\_DATA\_TIME\_IN\_MICRO\_SECONDS 5

extern volatile *uint8\_t* screen\_side;

extern volatile *uint8\_t* cursor\_x;

extern volatile *uint8\_t* cursor\_y;

extern volatile *uint8\_t* cursor\_position;

void GLCD\_initialize(void);

void GLCD\_select\_screen\_side(*uint8\_t* side);

void GLCD\_select\_left\_screen\_side(void);

void GLCD\_select\_right\_screen\_side(void);

void GLCD\_strobe\_data(void);

void GLCD\_instruction\_write(*uint8\_t* instruction);

void GLCD\_data\_write(*uint8\_t* data);

*uint8\_t* GLCD\_data\_read(void);

void GLCD\_set\_x(*uint8\_t* x);

void GLCD\_set\_y(*uint8\_t* y);

void GLCD\_set\_xy(*uint8\_t* x, *uint8\_t* y);

void GLCD\_set\_column\_and\_row(*uint8\_t* column, *uint8\_t* row);

void GLCD\_select\_starting\_line(*uint8\_t* starting\_line);

void GLCD\_clear\_screen(void);

void GLCD\_put\_character(unsigned char ascii\_code);

void GLCD\_print(char\* text);

void GLCD\_set\_dot(*uint8\_t* x\_axis, *uint8\_t* y\_axis);

void GLCD\_reset\_dot(*uint8\_t* x\_axis, *uint8\_t* y\_axis);

void GLCD\_increment\_cursor\_position(void);

void GLCD\_decrement\_cursor\_position(void);

*uint8\_t* GLCD\_get\_cursor\_position(void);

void GLCD\_print\_OCR1A(void);

void GLCD\_print\_OCR1B(void);

void GLCD\_print\_prescaler(void);

void GLCD\_print\_impulse\_counter(void);

void GLCD\_print\_max\_number\_of\_impulses(void);

void GLCD\_print\_frequency(void);

void GLCD\_print\_duty\_cycle(void);

void GLCD\_print\_sum\_of\_impulses(void);

void GLCD\_print\_sum\_of\_impulses\_time\_count(void);

void GLCD\_print\_number\_of\_counts(void);

void GLCD\_display\_parameters(void);

void GLCD\_print\_small\_ch(void);

void GLCD\_print\_small\_tj(void);

void GLCD\_print\_small\_zh(void);

void GLCD\_print\_small\_sh(void);

void GLCD\_print\_big\_dj(void);

void GLCD\_print\_decrement\_arrow(void);

void GLCD\_print\_increment\_arrow(void);

void GLCD\_print\_increment\_switch\_state(void);

void GLCD\_print\_decrement\_switch\_state(void);

void GLCD\_print\_slide\_switch\_state(void);

void GLCD\_remove\_slide\_switch\_state(void);

#endif /\* GLCD\_H\_ \*/

/\*

GLCD ima 21 kolonu i 8 redova (matrica piksela rezolucije 128 x 64).

SVI PINOVI KOJI SE KORISTE ZA GLCD TREBAJU DA BUDU PODESENI KAO FLOATING OUTPUT PINOVI!

\*/

/\* GLCD.c \*/

#include <stdlib.h>

#include <string.h>

#include "USART.h"

#include <util/delay.h>

#include "common.h"

#include "ports.h"

#include "PWM\_1.h"

#include "timer\_0.h"

#include "buttons\_and\_switches.h"

#include "GLCD.h"

const char font[1024] = {

0x00, 0x00, 0x00, 0x00, 0x00,// (space)

0x00, 0x00, 0x5F, 0x00, 0x00,// !

0x00, 0x07, 0x00, 0x07, 0x00,// "

0x14, 0x7F, 0x14, 0x7F, 0x14,// #

0x24, 0x2A, 0x7F, 0x2A, 0x12,// $

0x23, 0x13, 0x08, 0x64, 0x62,// %

0x36, 0x49, 0x55, 0x22, 0x50,// &

0x00, 0x05, 0x03, 0x00, 0x00,// '

0x00, 0x1C, 0x22, 0x41, 0x00,// (

0x00, 0x41, 0x22, 0x1C, 0x00,// )

0x08, 0x2A, 0x1C, 0x2A, 0x08,// \*

0x08, 0x08, 0x3E, 0x08, 0x08,// +

0x00, 0x50, 0x30, 0x00, 0x00,// ,

0x08, 0x08, 0x08, 0x08, 0x08,// -

0x00, 0x30, 0x30, 0x00, 0x00,// .

0x20, 0x10, 0x08, 0x04, 0x02,// /

0x3E, 0x51, 0x49, 0x45, 0x3E,// 0

0x00, 0x42, 0x7F, 0x40, 0x00,// 1

0x42, 0x61, 0x51, 0x49, 0x46,// 2

0x21, 0x41, 0x45, 0x4B, 0x31,// 3

0x18, 0x14, 0x12, 0x7F, 0x10,// 4

0x27, 0x45, 0x45, 0x45, 0x39,// 5

0x3C, 0x4A, 0x49, 0x49, 0x30,// 6

0x01, 0x71, 0x09, 0x05, 0x03,// 7

0x36, 0x49, 0x49, 0x49, 0x36,// 8

0x06, 0x49, 0x49, 0x29, 0x1E,// 9

0x00, 0x36, 0x36, 0x00, 0x00,// :

0x00, 0x56, 0x36, 0x00, 0x00,// ;

0x00, 0x08, 0x14, 0x22, 0x41,// <

0x14, 0x14, 0x14, 0x14, 0x14,// =

0x00, 0x7F, 0x3E, 0x1C, 0x08,// >

0x02, 0x01, 0x51, 0x09, 0x06,// ?

0x32, 0x49, 0x79, 0x41, 0x3E,// @

0x7E, 0x11, 0x11, 0x11, 0x7E,// A

0x7F, 0x49, 0x49, 0x49, 0x36,// B

0x3E, 0x41, 0x41, 0x41, 0x22,// C

0x7F, 0x41, 0x41, 0x22, 0x1C,// D

0x7F, 0x49, 0x49, 0x49, 0x41,// E

0x7F, 0x09, 0x09, 0x01, 0x01,// F

0x3E, 0x41, 0x41, 0x51, 0x32,// G

0x7F, 0x08, 0x08, 0x08, 0x7F,// H

0x00, 0x41, 0x7F, 0x41, 0x00,// I

0x20, 0x40, 0x41, 0x3F, 0x01,// J

0x7F, 0x08, 0x14, 0x22, 0x41,// K

0x7F, 0x40, 0x40, 0x40, 0x40,// L

0x7F, 0x02, 0x04, 0x02, 0x7F,// M

0x7F, 0x04, 0x08, 0x10, 0x7F,// N

0x3E, 0x41, 0x41, 0x41, 0x3E,// O

0x7F, 0x09, 0x09, 0x09, 0x06,// P

0x3E, 0x41, 0x51, 0x21, 0x5E,// Q

0x7F, 0x09, 0x19, 0x29, 0x46,// R

0x46, 0x49, 0x49, 0x49, 0x31,// S

0x01, 0x01, 0x7F, 0x01, 0x01,// T

0x3F, 0x40, 0x40, 0x40, 0x3F,// U

0x1F, 0x20, 0x40, 0x20, 0x1F,// V

0x7F, 0x20, 0x18, 0x20, 0x7F,// W

0x63, 0x14, 0x08, 0x14, 0x63,// X

0x03, 0x04, 0x78, 0x04, 0x03,// Y

0x61, 0x51, 0x49, 0x45, 0x43,// Z

0x00, 0x00, 0x7F, 0x41, 0x41,// [

0x02, 0x04, 0x08, 0x10, 0x20,// "\"

0x41, 0x41, 0x7F, 0x00, 0x00,// ]

0x04, 0x02, 0x01, 0x02, 0x04,// ^

0x40, 0x40, 0x40, 0x40, 0x40,// \_

0x00, 0x01, 0x02, 0x04, 0x00,// `

0x20, 0x54, 0x54, 0x54, 0x78,// a

0x7F, 0x48, 0x44, 0x44, 0x38,// b

0x38, 0x44, 0x44, 0x44, 0x20,// c

0x38, 0x44, 0x44, 0x48, 0x7F,// d

0x38, 0x54, 0x54, 0x54, 0x18,// e

0x08, 0x7E, 0x09, 0x01, 0x02,// f

0x08, 0x14, 0x54, 0x54, 0x3C,// g

0x7F, 0x08, 0x04, 0x04, 0x78,// h

0x00, 0x44, 0x7D, 0x40, 0x00,// i

0x20, 0x40, 0x44, 0x3D, 0x00,// j

0x00, 0x7F, 0x10, 0x28, 0x44,// k

0x00, 0x41, 0x7F, 0x40, 0x00,// l

0x7C, 0x04, 0x18, 0x04, 0x78,// m

0x7C, 0x08, 0x04, 0x04, 0x78,// n

0x38, 0x44, 0x44, 0x44, 0x38,// o

0x7C, 0x14, 0x14, 0x14, 0x08,// p

0x08, 0x14, 0x14, 0x18, 0x7C,// q

0x7C, 0x08, 0x04, 0x04, 0x08,// r

0x48, 0x54, 0x54, 0x54, 0x20,// s

0x04, 0x3F, 0x44, 0x40, 0x20,// t

0x3C, 0x40, 0x40, 0x20, 0x7C,// u

0x1C, 0x20, 0x40, 0x20, 0x1C,// v

0x3C, 0x40, 0x30, 0x40, 0x3C,// w

0x44, 0x28, 0x10, 0x28, 0x44,// x

0x0C, 0x50, 0x50, 0x50, 0x3C,// y

0x44, 0x64, 0x54, 0x4C, 0x44,// z

0x00, 0x08, 0x36, 0x41, 0x00,// {

0x00, 0x00, 0x7F, 0x00, 0x00,// |

0x00, 0x41, 0x36, 0x08, 0x00,// }

0x08, 0x08, 0x2A, 0x1C, 0x08,// ->

0x08, 0x1C, 0x2A, 0x08, 0x08,// <-

0x38, 0x45, 0x46, 0x45, 0x20,// ch

0x38, 0x44, 0x46, 0x45, 0x20,// tj

0x44, 0x65, 0x56, 0x4D, 0x44,// zh

0x48, 0x55, 0x56, 0x55, 0x20,// sh

0x7F, 0x49, 0x41, 0x22, 0x1C,// Dj

0x20, 0x7F, 0xFF, 0x7F, 0x20,// down arrow

0x04, 0xFE, 0xFF, 0xFE, 0x04 // up arrow

};

volatile *uint8\_t* screen\_side = 0;

volatile *uint8\_t* cursor\_x = 0;

volatile *uint8\_t* cursor\_y = 0;

volatile *uint8\_t* cursor\_position = 0;

void GLCD\_initialize(void){

RESET\_BIT(RS\_PORT, RS);

RESET\_BIT(RW\_PORT, RW);

RESET\_BIT(E\_PORT, EN);

RESET\_BIT(CS1\_PORT, CS1);

RESET\_BIT(CS2\_PORT, CS1);

SET\_BIT(RST\_PORT, RST);

*\_delay\_ms*(10);

RESET\_BIT(RST\_PORT, RST);

*\_delay\_ms*(10);

SET\_BIT(RST\_PORT, RST);

GLCD\_select\_left\_screen\_side();

GLCD\_instruction\_write(DISPLAY\_OFF);

GLCD\_instruction\_write(STARTING\_LINE);

GLCD\_instruction\_write(PAGE\_ADDRESS);

GLCD\_instruction\_write(Y\_ADDRESS);

GLCD\_instruction\_write(DISPLAY\_ON);

GLCD\_select\_right\_screen\_side();

GLCD\_instruction\_write(DISPLAY\_OFF);

GLCD\_instruction\_write(STARTING\_LINE);

GLCD\_instruction\_write(PAGE\_ADDRESS);

GLCD\_instruction\_write(Y\_ADDRESS);

GLCD\_instruction\_write(DISPLAY\_ON);

GLCD\_display\_parameters();

}

void GLCD\_select\_screen\_side(*uint8\_t* side){

RESET\_BIT(RS\_PORT, RS);

if(side == RIGHT){

RESET\_BIT(RW\_PORT, RW);

SET\_BIT(CS1\_PORT, CS1);

RESET\_BIT(CS2\_PORT, CS2);

GLCD\_instruction\_write(DISPLAY\_ON);

screen\_side = RIGHT;

}

else{

RESET\_BIT(RW\_PORT, RW);

RESET\_BIT(CS1\_PORT, CS1);

SET\_BIT(CS2\_PORT, CS2);

GLCD\_instruction\_write(DISPLAY\_ON);

screen\_side = LEFT;

}

}

void GLCD\_select\_left\_screen\_side(void){

GLCD\_select\_screen\_side(LEFT);

}

void GLCD\_select\_right\_screen\_side(void){

GLCD\_select\_screen\_side(RIGHT);

}

void GLCD\_strobe\_data(void){

SET\_BIT(E\_PORT, EN);

*\_delay\_us*(STROBE\_DATA\_TIME\_IN\_MICRO\_SECONDS);

RESET\_BIT(E\_PORT, EN);

}

void GLCD\_instruction\_write(*uint8\_t* instruction){

RESET\_BIT(RS\_PORT, RS);

RESET\_BIT(RW\_PORT, RW);

DATA\_PORT = instruction;

GLCD\_strobe\_data();

}

void GLCD\_data\_write(*uint8\_t* data){

SET\_BIT(RS\_PORT, RS);

RESET\_BIT(RW\_PORT, RW);

DATA\_PORT = data;

GLCD\_strobe\_data();

}

*uint8\_t* GLCD\_data\_read(void){

*uint8\_t* data = 0;

DATA\_PORT = 0x00;

SET\_BIT(RS\_PORT, RS);

SET\_BIT(RW\_PORT, RW);

GLCD\_strobe\_data();

SET\_BIT(E\_PORT, EN);

*\_delay\_ms*(1);

for(*uint8\_t* i = 0; i < 8; i++)data = ports\_read\_port\_C(); /\*ovo treba promeniti ako  
 port C nije data port\*/

RESET\_BIT(E\_PORT, EN);

DATA\_PORT = 0xFF;

*\_delay\_ms*(1);

RESET\_BIT(RS\_PORT, RS);

RESET\_BIT(RW\_PORT, RW);

return data;

}

void GLCD\_set\_x(*uint8\_t* x){

if(x < 64){

GLCD\_select\_left\_screen\_side();

GLCD\_instruction\_write(Y\_ADDRESS | x);

screen\_side = LEFT;

}

else if(x < 128) {

GLCD\_select\_right\_screen\_side();

GLCD\_instruction\_write(Y\_ADDRESS | (x - 64));

screen\_side = RIGHT;

}

cursor\_x = x;

}

void GLCD\_set\_y(*uint8\_t* y){

if(y < 8){

GLCD\_instruction\_write(PAGE\_ADDRESS | y);

cursor\_y = y;

}

}

void GLCD\_set\_xy(*uint8\_t* x, *uint8\_t* y){

GLCD\_set\_x(x);

GLCD\_set\_y(y);

}

void GLCD\_set\_column\_and\_row(*uint8\_t* column, *uint8\_t* row){

if(column >= 0 && column <= 20 && row >= 0 && row <=7){  
  
 GLCD\_set\_xy(column \* COLUMN\_OFFSET + 1, row);  
 }

}

void GLCD\_select\_starting\_line(*uint8\_t* starting\_line){

GLCD\_instruction\_write(STARTING\_LINE | starting\_line);

}

void GLCD\_clear\_screen(void){

*\_delay\_ms*(1);

unsigned char q,vert;

for (vert = 0; vert < 8; vert++){

GLCD\_set\_xy(0, vert);

for (q = 0; q < 64; q++)GLCD\_data\_write(0);

GLCD\_set\_xy(64, vert);

for (q = 0; q < 64; q++)GLCD\_data\_write(0);

}

}

void GLCD\_put\_character(unsigned char ascii\_code){

unsigned char trm;

if((cursor\_x + FONT\_LENGHT) > 127){

cursor\_y++;

if (cursor\_y>7)cursor\_y=0;

cursor\_x=0;

}

GLCD\_set\_xy(cursor\_x,cursor\_y);

for(trm = 0; trm < FONT\_LENGHT; trm++){

GLCD\_data\_write(font[((ascii\_code - 32) \* FONT\_LENGHT) + trm]);

cursor\_x++;

GLCD\_set\_xy(cursor\_x,cursor\_y);

}

GLCD\_data\_write(0);

cursor\_x++;

}

void GLCD\_print(char\* text){

while(\*text != 0){

GLCD\_put\_character(\*text);

text++;

}

}

void GLCD\_set\_dot(*uint8\_t* x\_axis, *uint8\_t* y\_axis){

*uint8\_t* data = 0;

GLCD\_set\_xy(x\_axis,(y\_axis / 8));

data = GLCD\_data\_read();

GLCD\_set\_xy(x\_axis,(y\_axis / 8));

GLCD\_data\_write(data | (1 << (y\_axis % 8)));

}

void GLCD\_reset\_dot(*uint8\_t* x\_axis, *uint8\_t* y\_axis){

*uint8\_t* data = 0;

GLCD\_set\_xy(x\_axis,(y\_axis / 8));

data = GLCD\_data\_read();

GLCD\_set\_xy(x\_axis,(y\_axis / 8));

GLCD\_data\_write(data & (0xff^(1 << (y\_axis % 8))) );

}

void GLCD\_increment\_cursor\_position(void){

GLCD\_set\_column\_and\_row(0, cursor\_position);

GLCD\_print(" ");

if(cursor\_position < MAX\_CURSOR\_POSITION)cursor\_position++;

GLCD\_set\_column\_and\_row(0, cursor\_position);

GLCD\_print(">>");

}

void GLCD\_decrement\_cursor\_position(void){

GLCD\_set\_column\_and\_row(0, cursor\_position);

GLCD\_print(" ");

if(cursor\_position > MIN\_CURSOR\_POSITION)cursor\_position--;

GLCD\_set\_column\_and\_row(0, cursor\_position);

GLCD\_print(">>");

}

*uint8\_t* GLCD\_get\_cursor\_position(void){

return cursor\_position;

}

void GLCD\_print\_OCR1A(void){

char OCR1A\_string[6] = {'0','0','0','0','0','\0'};

*uint16\_t* OCR1A\_value = PWM\_1\_get\_OCR1A();

OCR1A\_string[0] = common\_get\_fifth\_digit(OCR1A\_value);

OCR1A\_string[1] = common\_get\_fourth\_digit(OCR1A\_value);

OCR1A\_string[2] = common\_get\_third\_digit(OCR1A\_value);

OCR1A\_string[3] = common\_get\_second\_digit(OCR1A\_value);

OCR1A\_string[4] = common\_get\_first\_digit(OCR1A\_value);

GLCD\_set\_column\_and\_row(9, 0);

GLCD\_print(OCR1A\_string);

}

void GLCD\_print\_OCR1B(void){

char OCR1B\_string[6] = {'0','0','0','0','0','\0'};

*uint16\_t* OCR1B\_value = PWM\_1\_get\_OCR1B();

OCR1B\_string[0] = common\_get\_fifth\_digit(OCR1B\_value);

OCR1B\_string[1] = common\_get\_fourth\_digit(OCR1B\_value);

OCR1B\_string[2] = common\_get\_third\_digit(OCR1B\_value);

OCR1B\_string[3] = common\_get\_second\_digit(OCR1B\_value);

OCR1B\_string[4] = common\_get\_first\_digit(OCR1B\_value);

GLCD\_set\_column\_and\_row(9, 1);

GLCD\_print(OCR1B\_string);

}

void GLCD\_print\_prescaler(void){

char prescaler\_string[5] = {'0','0','0','0','\0'};

*uint16\_t* prescaler\_value = PWM\_1\_get\_prescaler();

prescaler\_string[0] = common\_get\_fourth\_digit(prescaler\_value);

prescaler\_string[1] = common\_get\_third\_digit(prescaler\_value);

prescaler\_string[2] = common\_get\_second\_digit(prescaler\_value);

prescaler\_string[3] = common\_get\_first\_digit(prescaler\_value);

GLCD\_set\_column\_and\_row(13, 2);

GLCD\_print(prescaler\_string);

}

void GLCD\_print\_impulse\_counter(void){

char impulse\_counter\_string[5] = {'0','0','0','0','\0'};

*uint16\_t* impulse\_counter\_value = PWM\_1\_get\_impulse\_counter();

impulse\_counter\_string[0] = common\_get\_fourth\_digit(impulse\_counter\_value);

impulse\_counter\_string[1] = common\_get\_third\_digit(impulse\_counter\_value);

impulse\_counter\_string[2] = common\_get\_second\_digit(impulse\_counter\_value);

impulse\_counter\_string[3] = common\_get\_first\_digit(impulse\_counter\_value);

GLCD\_set\_column\_and\_row(11, 3);

GLCD\_print(impulse\_counter\_string);

}

void GLCD\_print\_max\_number\_of\_impulses(void){

char max\_number\_of\_impulses\_string[5] = {'0','0','0','0','\0'};

*uint16\_t* max\_number\_of\_impulses\_value = PWM\_1\_get\_max\_number\_of\_impulses();

max\_number\_of\_impulses\_string[0] = common\_get\_fourth\_digit(max\_number\_of\_impulses\_value);

max\_number\_of\_impulses\_string[1] = common\_get\_third\_digit(max\_number\_of\_impulses\_value);

max\_number\_of\_impulses\_string[2] = common\_get\_second\_digit(max\_number\_of\_impulses\_value);

max\_number\_of\_impulses\_string[3] = common\_get\_first\_digit(max\_number\_of\_impulses\_value);

GLCD\_set\_column\_and\_row(16, 3);

GLCD\_print(max\_number\_of\_impulses\_string);

}

void GLCD\_print\_sum\_of\_impulses(void){

char sum\_of\_impulses\_string[11] = {'0','0','0','0','0','0','0','0','0','0','\0'};

*uint32\_t* sum\_of\_impulses\_value = PWM\_1\_get\_sum\_of\_impulses();

sum\_of\_impulses\_string[0] = common\_get\_tenth\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[1] = common\_get\_ninth\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[2] = common\_get\_eighth\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[3] = common\_get\_seventh\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[4] = common\_get\_sixth\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[5] = common\_get\_fifth\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[6] = common\_get\_fourth\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[7] = common\_get\_third\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[8] = common\_get\_second\_digit(sum\_of\_impulses\_value);

sum\_of\_impulses\_string[9] = common\_get\_first\_digit(sum\_of\_impulses\_value);

GLCD\_set\_column\_and\_row(10, 4);

GLCD\_print(sum\_of\_impulses\_string);

}

void GLCD\_print\_frequency(void){

float frequency = PWM\_1\_get\_frequency();

*uint16\_t* integer\_part = *trunc*(frequency);

float fraction\_part = frequency - integer\_part;

*uint16\_t* after\_point\_number = (*uint16\_t*)(fraction\_part \* 100);

char integer\_part\_string[4] = {'0','0','0','\0'};

char after\_point\_number\_string[3] = {'0','0','\0'};

integer\_part\_string[0] = common\_get\_third\_digit(integer\_part);

integer\_part\_string[1] = common\_get\_second\_digit(integer\_part);

integer\_part\_string[2] = common\_get\_first\_digit(integer\_part);

after\_point\_number\_string[0] = common\_get\_second\_digit(after\_point\_number);

after\_point\_number\_string[1] = common\_get\_first\_digit(after\_point\_number);

GLCD\_set\_column\_and\_row(12, 5);

GLCD\_print(integer\_part\_string);

GLCD\_set\_column\_and\_row(16, 5);

GLCD\_print(after\_point\_number\_string);

}

void GLCD\_print\_duty\_cycle(void){

float duty\_cycle = PWM\_1\_get\_duty\_cycle\_percentage();

if(duty\_cycle >= 100.0f)duty\_cycle = 99.99f;

*uint16\_t* integer\_part = *trunc*( duty\_cycle);

float fraction\_part = duty\_cycle - integer\_part;

*uint16\_t* after\_point\_number = (*uint16\_t*)(fraction\_part \* 100);

char integer\_part\_string[3] = {'0','0','\0'};

char after\_point\_number\_string[3] = {'0','0','\0'};

integer\_part\_string[0] = common\_get\_second\_digit(integer\_part);

integer\_part\_string[1] = common\_get\_first\_digit(integer\_part);

after\_point\_number\_string[0] = common\_get\_second\_digit(after\_point\_number);

after\_point\_number\_string[1] = common\_get\_first\_digit(after\_point\_number);

GLCD\_set\_column\_and\_row(14, 6);

GLCD\_print(integer\_part\_string);

GLCD\_set\_column\_and\_row(17, 6);

GLCD\_print(after\_point\_number\_string);

}

void GLCD\_print\_sum\_of\_impulses\_time\_count(void){

char reset\_sum\_of\_impulses\_time\_count\_string[2] = {'0','\0'};

*uint8\_t* reset\_sum\_of\_impulses\_time\_count\_value =  
 timer\_0\_get\_reset\_sum\_of\_impulses\_time\_count();

reset\_sum\_of\_impulses\_time\_count\_string[0] =  
 common\_get\_first\_digit(reset\_sum\_of\_impulses\_time\_count\_value);

GLCD\_set\_column\_and\_row(19, 7);

GLCD\_print(reset\_sum\_of\_impulses\_time\_count\_string);

}

void GLCD\_print\_number\_of\_counts(void){

char number\_of\_counts\_string[2] = {'0','\0'};

*uint8\_t* number\_of\_counts\_value = timer\_0\_get\_number\_of\_counts();

number\_of\_counts\_string[0] = common\_get\_first\_digit(number\_of\_counts\_value);

GLCD\_set\_column\_and\_row(17, 7);

GLCD\_print(number\_of\_counts\_string);

}

void GLCD\_display\_parameters(void){

GLCD\_clear\_screen();

GLCD\_set\_column\_and\_row(0, cursor\_position);

GLCD\_print(">>");

GLCD\_set\_column\_and\_row(3, 0);

GLCD\_print("OCR1A ");

GLCD\_print\_OCR1A();

GLCD\_set\_column\_and\_row(3, 1);

GLCD\_print("OCR1B ");

GLCD\_print\_OCR1B();

GLCD\_set\_column\_and\_row(3, 2);

GLCD\_print("PRESKALER ");

GLCD\_print\_prescaler();

GLCD\_set\_column\_and\_row(3, 3);

GLCD\_print("IMPULSI /");

GLCD\_print\_impulse\_counter();

GLCD\_print\_max\_number\_of\_impulses();

GLCD\_set\_column\_and\_row(0, 4);

GLCD\_print("SUMA IMP. ");

GLCD\_print\_sum\_of\_impulses();

GLCD\_set\_column\_and\_row(0, 5);

GLCD\_print("FREKVENCIJA . Hz");

GLCD\_print\_frequency();

GLCD\_set\_column\_and\_row(0, 6);

GLCD\_print("FAKTOR ISPUNE . %");

GLCD\_print\_duty\_cycle();

GLCD\_print\_slide\_switch\_state();

}

void GLCD\_print\_small\_ch(void){

GLCD\_put\_character(128);

}

void GLCD\_print\_small\_tj(void){

GLCD\_put\_character(129);

}

void GLCD\_print\_small\_zh(void){

GLCD\_put\_character(130);

}

void GLCD\_print\_small\_sh(void){

GLCD\_put\_character(131);

}

void GLCD\_print\_big\_dj(void){

GLCD\_put\_character(132);

}

void GLCD\_print\_decrement\_arrow(void){

GLCD\_put\_character(133);

}

void GLCD\_print\_increment\_arrow(void){

GLCD\_put\_character(134);

}

void GLCD\_print\_increment\_switch\_state(void){

GLCD\_set\_column\_and\_row(20 , 0);

GLCD\_print\_increment\_arrow();

}

void GLCD\_print\_decrement\_switch\_state(void){

GLCD\_set\_column\_and\_row(20 , 0);

GLCD\_print\_decrement\_arrow();

}

void GLCD\_print\_slide\_switch\_state(void){

if(!(READ\_PIN(PINB, INCREMENT\_SIGN\_SWITCH\_PIN)))GLCD\_print\_increment\_switch\_state();

else GLCD\_print\_decrement\_switch\_state();

}

void GLCD\_remove\_slide\_switch\_state(void){

GLCD\_set\_column\_and\_row(20 , 0);

GLCD\_print(" ");

}

/\* buzzer.h \*/

#ifndef BUZZER\_H\_

#define BUZZER\_H\_

#include "ports.h"

#define BUZZER\_PORT PORTD

#define BUZZER\_PIN 7

void buzzer\_initialize(void);

void buzzer\_activate\_tone(*uint8\_t* duty\_cycle\_percentage, *uint32\_t* time\_of\_enabled\_tone);

void buzzer\_activate\_button\_has\_been\_pushed\_tone(void);

void buzzer\_activate\_countdown\_tone(void);

void buzzer\_activate\_treatment\_is\_finished\_tone(void);

#endif /\* BUZZER\_H\_ \*/

/\* buzzer.c \*/

#include "buzzer.h"

#include "timer\_0.h"

#include "PWM\_2.h"

void buzzer\_initialize(void){

buzzer\_activate\_tone(255, 50);

}

void buzzer\_activate\_tone(*uint8\_t* duty\_cycle\_percentage, *uint32\_t* time\_of\_enabled\_tone){

PWM\_2\_set\_duty\_cycle\_percentage(duty\_cycle\_percentage);

PWM\_2\_enable\_non\_inverted\_wave\_form();

timer\_0\_delay\_in\_milliseconds(time\_of\_enabled\_tone);

PWM\_2\_disable();

}

void buzzer\_activate\_button\_has\_been\_pushed\_tone(void){

buzzer\_activate\_tone(196, 50);

}

void buzzer\_activate\_countdown\_tone(void){

buzzer\_activate\_tone(50, 50);

}

void buzzer\_activate\_treatment\_is\_finished\_tone(void){

PWM\_2\_set\_duty\_cycle\_percentage(50);

for(*uint8\_t* i = 0; i < 3; i++){

for(*uint8\_t* j = 0; j < 3; j++){

PWM\_2\_enable\_non\_inverted\_wave\_form();

timer\_0\_delay\_in\_milliseconds(50);

PWM\_2\_disable();

timer\_0\_delay\_in\_milliseconds(50);

}

timer\_0\_delay\_in\_milliseconds(400);

}

}

/\* buttons\_and\_switches.h \*/

#ifndef BUTTONS\_AND\_SWITCHES\_H\_

#define BUTTONS\_AND\_SWITCHES\_H\_

#include "ports.h"

#define SAVE\_PARAMETERS\_BUTTON\_PORT PORTA

#define CURSOR\_UP\_BUTTON\_PORT PORTB

#define CURSOR\_DOWN\_BUTTON\_PORT PORTB

#define INCREMENT\_SIGN\_SWITCH\_PORT PORTB

#define INCREMENT\_BY\_1\_BUTTON\_PORT PORTB

#define INCREMENT\_BY\_10\_BUTTON\_PORT PORTB

#define INCREMENT\_BY\_100\_BUTTON\_PORT PORTB

#define INCREMENT\_BY\_1000\_BUTTON\_PORT PORTB

#define INCREMENT\_BY\_10000\_BUTTON\_PORT PORTB

#define RESTART\_BUTTON\_PORT PORTD

#define START\_BUTTON\_PORT PORTD

#define PAUSE\_BUTTON\_PORT PORTD

#define RESET\_BUTTON\_PORT PORTD

#define SAVE\_PARAMETERS\_BUTTON\_PIN 7

#define CURSOR\_UP\_BUTTON\_PIN 0

#define CURSOR\_DOWN\_BUTTON\_PIN 1

#define INCREMENT\_SIGN\_SWITCH\_PIN 2

#define INCREMENT\_BY\_1\_BUTTON\_PIN 3

#define INCREMENT\_BY\_10\_BUTTON\_PIN 4

#define INCREMENT\_BY\_100\_BUTTON\_PIN 5

#define INCREMENT\_BY\_1000\_BUTTON\_PIN 6

#define INCREMENT\_BY\_10000\_BUTTON\_PIN 7

#define RESTART\_BUTTON\_PIN 0

#define START\_BUTTON\_PIN 2

#define PAUSE\_BUTTON\_PIN 3

#define RESET\_BUTTON\_PIN 6

#define NUMBER\_OF\_BUTTONS 12

#define NUMBER\_OF\_NO\_BOUNCES 25

#define SAVE\_BUTTON\_ARRAY\_INDEX 0

#define CURSOR\_UP\_BUTTON\_ARRAY\_INDEX 1

#define CURSOR\_DOWN\_BUTTON\_ARRAY\_INDEX 2

#define INCREMENT\_BY\_1\_BUTTON\_ARRAY\_INDEX 3

#define INCREMENT\_BY\_10\_BUTTON\_ARRAY\_INDEX 4

#define INCREMENT\_BY\_100\_BUTTON\_ARRAY\_INDEX 5

#define INCREMENT\_BY\_1000\_BUTTON\_ARRAY\_INDEX 6

#define INCREMENT\_BY\_10000\_BUTTON\_ARRAY\_INDEX 7

#define RESTART\_BUTTON\_ARRAY\_INDEX 8

#define START\_BUTTON\_ARRAY\_INDEX 9

#define PAUSE\_BUTTON\_ARRAY\_INDEX 10

#define RESET\_BUTTON\_ARRAY\_INDEX 11

extern volatile *uint8\_t* bounce[NUMBER\_OF\_BUTTONS];

extern volatile *uint8\_t* increment\_switch\_number\_of\_bounces;

extern volatile *uint8\_t* increment\_switch\_state;

extern volatile *uint8\_t* increment\_switch\_state\_current;

extern volatile *uint8\_t* increment\_switch\_state\_previous;

void buttons\_and\_switches\_initialize(void);

*uint8\_t* buttons\_and\_switches\_port\_B\_get\_button\_state(*uint8\_t* button\_pin,  
 *uint8\_t* button\_array\_index);

*uint8\_t* buttons\_and\_switches\_port\_D\_get\_button\_state(*uint8\_t* button\_pin,  
 *uint8\_t* button\_array\_index);

*uint8\_t* buttons\_and\_switches\_save\_parameters\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_cursor\_up\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_cursor\_down\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_increment\_sign\_switch\_changed\_state(void);

*uint8\_t* buttons\_and\_switches\_increment\_by\_1\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_increment\_by\_10\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_increment\_by\_100\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_increment\_by\_1000\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_increment\_by\_10000\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_restart\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_start\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_pause\_button\_is\_pushed(void);

*uint8\_t* buttons\_and\_switches\_reset\_button\_is\_pushed(void);

#endif /\* BUTTONS\_AND\_SWITCHES\_H\_ \*/

/\*

PORTA

PA7 - INPUT (BUTTON SAVE PARAMETERS)

PORTB

PB0 - INPUT (BUTTON GLCD CURSOR UP)

PB1 - INPUT (BUTTON GLCD CURSOR DOWN)

PB2 - INPUT (INCREMENT / DECREMENT)

PB3 - INPUT (BY 1)

PB4 - INPUT (BY 10)

PB5 - INPUT (BY 100)

PB6 - INPUT (BY 1000)

PB7 - INPUT (BY 10000)

PORTD

PD0 - INPUT (BUTTON RESTART)

PD2 - INPUT (BUTTON START)

PD3 - INPUT (BUTTON PAUSE)

PD6 - INPUT (BUTTON RESET)

\*/

/\* buttons\_and\_switches.c \*/

#include <stdbool.h>

#include "buttons\_and\_switches.h"

#include "GLCD.h"

#include "USART.h"

volatile *uint8\_t* bounce[NUMBER\_OF\_BUTTONS];

volatile *uint8\_t* increment\_switch\_number\_of\_bounces = NUMBER\_OF\_NO\_BOUNCES;

volatile *uint8\_t* increment\_switch\_state = false;

volatile *uint8\_t* increment\_switch\_state\_current = false;

volatile *uint8\_t* increment\_switch\_state\_previous = false;

volatile *uint8\_t* current\_increment\_switch\_state = false;

volatile *uint8\_t* previous\_increment\_switch\_state = false;

void buttons\_and\_switches\_initialize(void){

for(*uint8\_t* button\_array\_index = 0; button\_array\_index < NUMBER\_OF\_BUTTONS;  
 button\_array\_index++)bounce[button\_array\_index] = 0;

increment\_switch\_number\_of\_bounces = NUMBER\_OF\_NO\_BOUNCES;

increment\_switch\_state = false;

increment\_switch\_state\_current = false;

increment\_switch\_state\_previous = false;

current\_increment\_switch\_state = false;

previous\_increment\_switch\_state = false;

}

*uint8\_t* buttons\_and\_switches\_port\_B\_get\_button\_state(*uint8\_t* button\_pin,  
 *uint8\_t* button\_array\_index){

*uint8\_t* button\_state = false;

if(!(READ\_PIN(PINB, button\_pin))){

if(bounce[button\_array\_index] < NUMBER\_OF\_NO\_BOUNCES)bounce[button\_array\_index]++;

}

else bounce[button\_array\_index] = 0;

if(bounce[button\_array\_index] == NUMBER\_OF\_NO\_BOUNCES)button\_state = true;

if(bounce[button\_array\_index] >= NUMBER\_OF\_NO\_BOUNCES)bounce[button\_array\_index] =   
 NUMBER\_OF\_NO\_BOUNCES;

return button\_state;

}

*uint8\_t* buttons\_and\_switches\_port\_D\_get\_button\_state(*uint8\_t* button\_pin,   
 *uint8\_t* button\_array\_index){

*uint8\_t* button\_state = false;

if(!(READ\_PIN(PIND, button\_pin))){

if(bounce[button\_array\_index] < NUMBER\_OF\_NO\_BOUNCES)bounce[button\_array\_index]++;

}

else bounce[button\_array\_index] = 0;

if(bounce[button\_array\_index] == NUMBER\_OF\_NO\_BOUNCES)button\_state = true;

if(bounce[button\_array\_index] >= NUMBER\_OF\_NO\_BOUNCES)  
 bounce[button\_array\_index] = NUMBER\_OF\_NO\_BOUNCES;

return button\_state;

}

*uint8\_t* buttons\_and\_switches\_save\_parameters\_button\_is\_pushed(void){

*uint8\_t* button\_state = false;

if(!(READ\_PIN(PINA, SAVE\_PARAMETERS\_BUTTON\_PIN))){

if(bounce[SAVE\_BUTTON\_ARRAY\_INDEX] < NUMBER\_OF\_NO\_BOUNCES)  
 bounce[SAVE\_BUTTON\_ARRAY\_INDEX]++;

}

else bounce[SAVE\_BUTTON\_ARRAY\_INDEX] = 0;

if(bounce[SAVE\_BUTTON\_ARRAY\_INDEX] == NUMBER\_OF\_NO\_BOUNCES)button\_state = true;

if(bounce[SAVE\_BUTTON\_ARRAY\_INDEX] >= NUMBER\_OF\_NO\_BOUNCES)  
 bounce[SAVE\_BUTTON\_ARRAY\_INDEX] = NUMBER\_OF\_NO\_BOUNCES;

return button\_state;

}

*uint8\_t* buttons\_and\_switches\_cursor\_up\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_B\_get\_button\_state(CURSOR\_UP\_BUTTON\_PIN,  
 CURSOR\_UP\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_cursor\_down\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_B\_get\_button\_state(CURSOR\_DOWN\_BUTTON\_PIN,  
 CURSOR\_DOWN\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed(void){

increment\_switch\_state\_previous = increment\_switch\_state\_current;

increment\_switch\_state\_current = READ\_PIN(PINB, INCREMENT\_SIGN\_SWITCH\_PIN);

if(increment\_switch\_state\_previous != increment\_switch\_state\_current){

increment\_switch\_number\_of\_bounces = (NUMBER\_OF\_NO\_BOUNCES + 1) / 2 + NUMBER\_OF\_NO\_BOUNCES;

}

if(increment\_switch\_state\_current){

if(increment\_switch\_number\_of\_bounces > NUMBER\_OF\_NO\_BOUNCES)  
 increment\_switch\_number\_of\_bounces--;

}

else{

if(increment\_switch\_number\_of\_bounces < (2 \* NUMBER\_OF\_NO\_BOUNCES))  
 increment\_switch\_number\_of\_bounces++;

}

if(increment\_switch\_number\_of\_bounces == NUMBER\_OF\_NO\_BOUNCES)  
 increment\_switch\_state = false;

if(increment\_switch\_number\_of\_bounces == (2 \* NUMBER\_OF\_NO\_BOUNCES))  
 increment\_switch\_state = true;

return increment\_switch\_state;

}

*uint8\_t* buttons\_and\_switches\_increment\_sign\_switch\_changed\_state(void){

previous\_increment\_switch\_state = current\_increment\_switch\_state;

current\_increment\_switch\_state = buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed();

if(previous\_increment\_switch\_state != current\_increment\_switch\_state){

return true;

}

else{

return false;

}

}

*uint8\_t* buttons\_and\_switches\_increment\_by\_1\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_B\_get\_button\_state(INCREMENT\_BY\_1\_BUTTON\_PIN,  
 INCREMENT\_BY\_1\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_increment\_by\_10\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_B\_get\_button\_state(INCREMENT\_BY\_10\_BUTTON\_PIN,  
 INCREMENT\_BY\_10\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_increment\_by\_100\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_B\_get\_button\_state(INCREMENT\_BY\_100\_BUTTON\_PIN,  
 INCREMENT\_BY\_100\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_increment\_by\_1000\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_B\_get\_button\_state(INCREMENT\_BY\_1000\_BUTTON\_PIN,  
 INCREMENT\_BY\_1000\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_increment\_by\_10000\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_B\_get\_button\_state(INCREMENT\_BY\_10000\_BUTTON\_PIN,  
 INCREMENT\_BY\_10000\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_restart\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_D\_get\_button\_state(RESTART\_BUTTON\_PIN,  
 RESTART\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_start\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_D\_get\_button\_state(START\_BUTTON\_PIN,  
 START\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_pause\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_D\_get\_button\_state(PAUSE\_BUTTON\_PIN,  
 PAUSE\_BUTTON\_ARRAY\_INDEX);

}

*uint8\_t* buttons\_and\_switches\_reset\_button\_is\_pushed(void){

return buttons\_and\_switches\_port\_D\_get\_button\_state(RESET\_BUTTON\_PIN,  
 RESET\_BUTTON\_ARRAY\_INDEX);

}

/\* RGB\_LED.h \*/

#ifndef RGB\_LED\_H\_

#define RGB\_LED\_H\_

#include <avr/io.h>

#include <avr/interrupt.h>

#define ws2812\_resettime 300

///////////////////////////////////////////////////////////////////////

// Define Reset time in µs.

//

// This is the time the library spends waiting after writing the data.

//

// WS2813 needs 300 µs reset time

// WS2812 and clones only need 50 µs

//

///////////////////////////////////////////////////////////////////////

#if !defined(ws2812\_resettime)

#define ws2812\_resettime 300

#endif

///////////////////////////////////////////////////////////////////////

// Define I/O pin

///////////////////////////////////////////////////////////////////////

#if !defined(ws2812\_port)

#define ws2812\_port A // Data port

#endif

#if !defined(ws2812\_pin)

#define ws2812\_pin 6 // Data out pin

#endif

#define SHINE 100

/\*

\* Structure of the LED array

\*

\* cRGB: RGB for WS2812S/B/C/D, SK6812, SK6812Mini, SK6812WWA, APA104, APA106

\* cRGBW: RGBW for SK6812RGBW

\*/

struct cRGB { *uint8\_t* g; *uint8\_t* r; *uint8\_t* b; };

struct cRGBW { *uint8\_t* g; *uint8\_t* r; *uint8\_t* b; *uint8\_t* w;};

/\* User Interface

\*

\* Input:

\* ledarray: An array of GRB data describing the LED colors

\* number\_of\_leds: The number of LEDs to write

\* pinmask (optional): Bitmask describing the output bin. e.g. \_BV(PB0)

\*

\* The functions will perform the following actions:

\* - Set the data-out pin as output

\* - Send out the LED data

\* - Wait 50µs to reset the LEDs

\*/

void ws2812\_setleds (struct cRGB \*ledarray, *uint16\_t* number\_of\_leds);

void ws2812\_setleds\_pin (struct cRGB \*ledarray, *uint16\_t* number\_of\_leds,  
 *uint8\_t* pinmask);

void ws2812\_setleds\_rgbw(struct cRGBW \*ledarray, *uint16\_t* number\_of\_leds);

/\*

\* Old interface / Internal functions

\*

\* The functions take a byte-array and send to the data output as WS2812 bitstream.

\* The length is the number of bytes to send - three per LED.

\*/

void ws2812\_sendarray (*uint8\_t* \*array,*uint16\_t* length);

void ws2812\_sendarray\_mask(*uint8\_t* \*array,*uint16\_t* length, *uint8\_t* pinmask);

void RGB\_LED\_initialize (void);

void RGB\_LED\_set\_red (void);

void RGB\_LED\_set\_green (void);

void RGB\_LED\_set\_dark\_blue (void);

void RGB\_LED\_set\_yellow (void);

void RGB\_LED\_set\_light\_blue (void);

void RGB\_LED\_set\_pink (void);

void RGB\_LED\_set\_white (void);

void RGB\_LED\_disable (void);

/\*

\* Internal defines

\*/

#if !defined(CONCAT)

#define CONCAT(a, b) a ## b

#endif

#if !defined(CONCAT\_EXP)

#define CONCAT\_EXP(a, b) CONCAT(a, b)

#endif

#define ws2812\_PORTREG CONCAT\_EXP(PORT,ws2812\_port)

#define ws2812\_DDRREG CONCAT\_EXP(DDR,ws2812\_port)

#endif /\* LIGHT\_WS2812\_H\_ \*/

/\* RGB\_LED.c \*/

#include <avr/interrupt.h>

#include <avr/io.h>

#include "RGB\_LED.h"

#include "USART.h"

#include <util/delay.h>

// Setleds for standard RGB

void inline ws2812\_setleds(struct cRGB \*ledarray, *uint16\_t* leds)

{

ws2812\_setleds\_pin(ledarray,leds, \_BV(ws2812\_pin));

}

void inline ws2812\_setleds\_pin(struct cRGB \*ledarray, *uint16\_t* leds, *uint8\_t* pinmask)

{

ws2812\_sendarray\_mask((*uint8\_t*\*)ledarray,leds+leds+leds,pinmask);

*\_delay\_us*(ws2812\_resettime);

}

// Setleds for SK6812RGBW

void inline ws2812\_setleds\_rgbw(struct cRGBW \*ledarray, *uint16\_t* leds)

{

ws2812\_sendarray\_mask((*uint8\_t*\*)ledarray,leds<<2,\_BV(ws2812\_pin));

*\_delay\_us*(ws2812\_resettime);

}

void ws2812\_sendarray(*uint8\_t* \*data,*uint16\_t* datlen)

{

ws2812\_sendarray\_mask(data,datlen,\_BV(ws2812\_pin));

}

void RGB\_LED\_initialize(void){

for(*uint8\_t* i = 0; i < 2; i++){

RGB\_LED\_set\_white();

*\_delay\_ms*(130);

RGB\_LED\_disable();

*\_delay\_ms*(130);

}

RGB\_LED\_set\_red();

}

void RGB\_LED\_set\_green(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=SHINE;led[0].g=00;led[0].b=0;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

void RGB\_LED\_set\_red(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=0;led[0].g=SHINE;led[0].b=0;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

void RGB\_LED\_set\_dark\_blue(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=0;led[0].g=00;led[0].b=SHINE;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

void RGB\_LED\_set\_yellow(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=SHINE;led[0].g=SHINE;led[0].b=0;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

void RGB\_LED\_set\_light\_blue(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=SHINE;led[0].g=0;led[0].b=SHINE;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

void RGB\_LED\_set\_pink(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=00;led[0].g=SHINE;led[0].b=SHINE;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

void RGB\_LED\_set\_white(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=SHINE;led[0].g=SHINE;led[0].b=SHINE;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

void RGB\_LED\_disable(void){

for(*uint8\_t* i = 0; i < 3; i++){

struct cRGB led[1];

led[0].r=00;led[0].g=00;led[0].b=00;

ws2812\_setleds(led,1);

*\_delay\_ms*(1);

}

}

//calculator boja: http://kourikustoms.com/tools/NeoPixelColorSelector.html

/\*

This routine writes an array of bytes with RGB values to the Dataout pin

using the fast 800kHz clockless WS2811/2812 protocol.

\*/

// Timing in ns

#define w\_zeropulse 350/\*350\*/

#define w\_onepulse 1360/\*900\*/

#define w\_totalperiod 1710/\*1250\*/

// Fixed cycles used by the inner loop

#define w\_fixedlow 2

#define w\_fixedhigh 4

#define w\_fixedtotal 8

// Insert NOPs to match the timing, if possible

#define w\_zerocycles (((F\_CPU/1000)\*w\_zeropulse )/1000000)

#define w\_onecycles (((F\_CPU/1000)\*w\_onepulse +500000)/1000000)

#define w\_totalcycles (((F\_CPU/1000)\*w\_totalperiod +500000)/1000000)

// w1 - nops between rising edge and falling edge - low

#define w1 (w\_zerocycles-w\_fixedlow)

// w2 nops between fe low and fe high

#define w2 (w\_onecycles-w\_fixedhigh-w1)

// w3 nops to complete loop

#define w3 (w\_totalcycles-w\_fixedtotal-w1-w2)

#if w1>0

#define w1\_nops w1

#else

#define w1\_nops 0

#endif

// The only critical timing parameter is the minimum pulse length of the "0"

// Warn or throw error if this timing can not be met with current F\_CPU settings.

#define w\_lowtime ((w1\_nops+w\_fixedlow)\*1000000)/(F\_CPU/1000)

#if w\_lowtime>550

#error "Light\_ws2812: Sorry, the clock speed is too low. Did you set F\_CPU correctly?"

#elif w\_lowtime>450

#warning "Light\_ws2812: The timing is critical and may only work on WS2812B, not on WS2812(S)."

#warning "Please consider a higher clockspeed, if possible"

#endif

#if w2>0

#define w2\_nops w2

#else

#define w2\_nops 0

#endif

#if w3>0

#define w3\_nops w3

#else

#define w3\_nops 0

#endif

#define w\_nop1 "nop \n\t"

#define w\_nop2 "rjmp .+0 \n\t"

#define w\_nop4 w\_nop2 w\_nop2

#define w\_nop8 w\_nop4 w\_nop4

#define w\_nop16 w\_nop8 w\_nop8

void inline ws2812\_sendarray\_mask(*uint8\_t* \*data,*uint16\_t* datlen,*uint8\_t* maskhi)

{

*uint8\_t* curbyte,ctr,masklo;

*uint8\_t* sreg\_prev;

ws2812\_DDRREG |= maskhi; // Enable output

masklo =~maskhi&ws2812\_PORTREG;

maskhi |= ws2812\_PORTREG;

sreg\_prev=SREG;

cli();

while (datlen--) {

curbyte=\*data++;

asm volatile(

" ldi %0,8 \n\t"

"loop%=: \n\t"

" out %2,%3 \n\t" // '1' [01] '0' [01] - re

#if (w1\_nops&1)

w\_nop1

#endif

#if (w1\_nops&2)

w\_nop2

#endif

#if (w1\_nops&4)

w\_nop4

#endif

#if (w1\_nops&8)

w\_nop8

#endif

#if (w1\_nops&16)

w\_nop16

#endif

" sbrs %1,7 \n\t" // '1' [03] '0' [02]

" out %2,%4 \n\t" // '1' [--] '0' [03] - fe-low

" lsl %1 \n\t" // '1' [04] '0' [04]

#if (w2\_nops&1)

w\_nop1

#endif

#if (w2\_nops&2)

w\_nop2

#endif

#if (w2\_nops&4)

w\_nop4

#endif

#if (w2\_nops&8)

w\_nop8

#endif

#if (w2\_nops&16)

w\_nop16

#endif

" out %2,%4 \n\t" // '1' [+1] '0' [+1] - fe-high

#if (w3\_nops&1)

w\_nop1

#endif

#if (w3\_nops&2)

w\_nop2

#endif

#if (w3\_nops&4)

w\_nop4

#endif

#if (w3\_nops&8)

w\_nop8

#endif

#if (w3\_nops&16)

w\_nop16

#endif

" dec %0 \n\t" // '1' [+2] '0' [+2]

" brne loop%=\n\t" // '1' [+3] '0' [+4]

: "=&d" (ctr)

: "r" (curbyte), "I" (\_SFR\_IO\_ADDR(ws2812\_PORTREG)), "r" (maskhi), "r" (masklo)

);

}

SREG = sreg\_prev;

}

/\* state\_machine.h \*/

#ifndef STATE\_MACHINE\_H\_\_

#define STATE\_MACHINE\_H\_

// STATES

#define STAND\_BY 1

#define COUNTDOWN 2

#define THERAPEUTIC\_TREATMENT 3

// EVENTS

#define NOTHING\_IS\_HAPPENING 0

#define RESET 1

#define PAUSE 2

#define SAVE 3

#define CURSOR\_UP 4

#define CURSOR\_DOWN 5

#define INCREMENT\_BY\_1 6

#define INCREMENT\_BY\_10 7

#define INCREMENT\_BY\_100 8

#define INCREMENT\_BY\_1000 9

#define INCREMENT\_BY\_10000 10

#define START 11

#define RESTART 12

#define SLIDE\_SWITCH 13

#define WAITING\_FOR\_RESTART 1

extern volatile *uint8\_t* state;

extern volatile *uint8\_t* event;

extern volatile *uint16\_t* current\_number\_of\_impulses;

void state\_machine\_initialize(void);

*uint8\_t* state\_machine\_get\_event(void);

void state\_machine\_reset(void);

void state\_machine\_pause(void);

void state\_machine\_save(void);

void state\_machine\_cursor\_up(void);

void state\_machine\_cursor\_down(void);

void increment\_by\_number(*uint16\_t* number);

void state\_machine\_increment\_by\_1(void);

void state\_machine\_increment\_by\_10(void);

void state\_machine\_increment\_by\_100(void);

void state\_machine\_increment\_by\_1000(void);

void state\_machine\_increment\_by\_10000(void);

void state\_machine\_start(void);

void state\_machine\_restart(void);

void state\_machine\_countdown(void);

void state\_machine\_therapeutic\_treatment(void);

void state\_machine\_slide\_switch(void);

void state\_machine\_routine(void);

#endif /\* STATE\_MACHINE\_H\_ \*/

/\* state\_machine.c \*/  
  
#include <avr/io.h>

#include "USART.h"

#include "timer\_0.h"

#include "PWM\_1.h"

#include "PWM\_2.h"

#include "GLCD.h"

#include "watch\_dog\_timer.h"

#include "buzzer.h"

#include "buttons\_and\_switches.h"

#include "RGB\_LED.h"

#include "state\_machine.h"

volatile *uint8\_t* state = 0;

volatile *uint8\_t* event = 0;

volatile *uint16\_t* current\_number\_of\_impulses = 0;

void state\_machine\_initialize(void){

state = STAND\_BY;

event = NOTHING\_IS\_HAPPENING;

}

/////////////////////////////////////////////////////////////////////////////////////////

*uint8\_t* state\_machine\_get\_event(void){

if(buttons\_and\_switches\_increment\_sign\_switch\_changed\_state())return SLIDE\_SWITCH;

else if(buttons\_and\_switches\_reset\_button\_is\_pushed())return RESET;

else if(buttons\_and\_switches\_pause\_button\_is\_pushed())return PAUSE;

else if(buttons\_and\_switches\_save\_parameters\_button\_is\_pushed())return SAVE;

else if(buttons\_and\_switches\_cursor\_up\_button\_is\_pushed())return CURSOR\_UP;

else if(buttons\_and\_switches\_cursor\_down\_button\_is\_pushed())return CURSOR\_DOWN;

else if(buttons\_and\_switches\_increment\_by\_1\_button\_is\_pushed())return INCREMENT\_BY\_1;

else if(buttons\_and\_switches\_increment\_by\_10\_button\_is\_pushed())return INCREMENT\_BY\_10;

else if(buttons\_and\_switches\_increment\_by\_100\_button\_is\_pushed())return INCREMENT\_BY\_100;

else if(buttons\_and\_switches\_increment\_by\_1000\_button\_is\_pushed())return INCREMENT\_BY\_1000;

else if(buttons\_and\_switches\_increment\_by\_10000\_button\_is\_pushed())return INCREMENT\_BY\_10000;

else if(buttons\_and\_switches\_start\_button\_is\_pushed())return START;

else if(buttons\_and\_switches\_restart\_button\_is\_pushed())return RESTART;

else return NOTHING\_IS\_HAPPENING;

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_reset(void){

*uint16\_t* ocr1b = OCR1B;

*uint16\_t* prescaler = PWM\_1\_get\_prescaler();

while(state == THERAPEUTIC\_TREATMENT){

if(TCNT1 > ocr1b)TCCR1B &= 0b11111000;

if(PWM\_1\_get\_prescaler() == 0){

PWM\_1\_set\_prescaler(prescaler);

break;

}

}

PWM\_1\_disable();

PWM\_1\_set\_prescaler(prescaler);

PWM\_1\_save\_the\_sum\_of\_impulses();

PWM\_1\_reset\_impulse\_counter();

GLCD\_print\_impulse\_counter();

GLCD\_print\_sum\_of\_impulses();

GLCD\_print\_slide\_switch\_state();

RGB\_LED\_set\_light\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

timer\_0\_reset\_sum\_of\_impulses\_time();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print("RESET SUME IMP. ZA ");

GLCD\_print\_sum\_of\_impulses\_time\_count();

while(buttons\_and\_switches\_reset\_button\_is\_pushed()){

if(timer\_0\_reset\_sum\_of\_impulses\_time\_has\_passed()){

PWM\_1\_reset\_sum\_of\_impulses();

GLCD\_print\_sum\_of\_impulses();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print("SUMA IMP. RESETOVANA");

break;

}

}

RGB\_LED\_set\_dark\_blue();

while(buttons\_and\_switches\_reset\_button\_is\_pushed());

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print(" ");

RGB\_LED\_set\_red();

state = STAND\_BY;

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_pause(void){

*uint16\_t* ocr1b = OCR1B;

*uint16\_t* prescaler = PWM\_1\_get\_prescaler();

while(state == THERAPEUTIC\_TREATMENT){

if(TCNT1 > ocr1b)TCCR1B &= 0b11111000;

if(PWM\_1\_get\_prescaler() == 0){

PWM\_1\_set\_prescaler(prescaler);

break;

}

}

PWM\_1\_disable();

PWM\_1\_set\_prescaler(prescaler);

PWM\_1\_save\_the\_sum\_of\_impulses();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print(" ");

GLCD\_print\_impulse\_counter();

GLCD\_print\_sum\_of\_impulses();

GLCD\_print\_slide\_switch\_state();

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_pause\_button\_is\_pushed());

RGB\_LED\_set\_red();

state = STAND\_BY;

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_save(void){

PWM\_1\_save\_parameters();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print("PARAMETRI UMEMORISANI");

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_save\_parameters\_button\_is\_pushed());

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print(" ");

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_cursor\_up(void){

GLCD\_decrement\_cursor\_position();

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_cursor\_up\_button\_is\_pushed());

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_cursor\_down(void){

GLCD\_increment\_cursor\_position();

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_cursor\_down\_button\_is\_pushed());

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void increment\_by\_number(*uint16\_t* number){

*uint8\_t* cursor\_position = GLCD\_get\_cursor\_position();

switch(cursor\_position){

case OCR1A\_CURSOR\_POSITION:{

if(buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed()){

for(*uint16\_t* number\_of\_increments = 0; number\_of\_increments < number;  
 number\_of\_increments++)PWM\_1\_minimaly\_increase\_period();

}

else{

for(*uint16\_t* number\_of\_decrements = 0; number\_of\_decrements < number;  
 number\_of\_decrements++)PWM\_1\_minimaly\_decrease\_period();

}

GLCD\_print\_OCR1A();

GLCD\_print\_frequency();

GLCD\_print\_duty\_cycle();

break;

}

case OCR1B\_CURSOR\_POSITION:{

if(buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed()){

for(*uint16\_t* number\_of\_increments = 0; number\_of\_increments < number;  
 number\_of\_increments++)PWM\_1\_minimaly\_increase\_duty\_cycle\_percentage();

}

else{

for(*uint16\_t* number\_of\_decrements = 0; number\_of\_decrements < number;   
 number\_of\_decrements++)PWM\_1\_minimaly\_decrease\_duty\_cycle\_percentage();

}

GLCD\_print\_OCR1B();

GLCD\_print\_duty\_cycle();

break;

}

case PRESCALER\_CURSOR\_POSITION:{

if(buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed() && number == 1)  
 PWM\_1\_minimaly\_increase\_prescaler();

else if(!buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed() && number == 1)  
 PWM\_1\_minimaly\_decrease\_prescaler();

GLCD\_print\_prescaler();

GLCD\_print\_frequency();

GLCD\_print\_duty\_cycle();

break;

}

case NUMBER\_OF\_IMPULSES\_CURSOR\_POSITION:{

if(buttons\_and\_switches\_increment\_sign\_switch\_is\_pushed()){

for(*uint16\_t* number\_of\_increments = 0; number\_of\_increments < number;  
 number\_of\_increments++)PWM\_1\_minimaly\_increase\_max\_number\_of\_impulses();

}

else{

for(*uint16\_t* number\_of\_decrements = 0; number\_of\_decrements < number;  
 number\_of\_decrements++)PWM\_1\_minimaly\_decrease\_max\_number\_of\_impulses();

if(PWM\_1\_get\_impulse\_counter() > PWM\_1\_get\_max\_number\_of\_impulses()){

PWM\_1\_set\_impulse\_counter\_to\_max\_number\_of\_impulses();

GLCD\_print\_impulse\_counter();

}

}

GLCD\_print\_max\_number\_of\_impulses();

break;

}

}

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_increment\_by\_1(void){

increment\_by\_number(1);

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_increment\_by\_1\_button\_is\_pushed());

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_increment\_by\_10(void){

increment\_by\_number(10);

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_increment\_by\_10\_button\_is\_pushed());

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_increment\_by\_100(void){

increment\_by\_number(100);

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_increment\_by\_100\_button\_is\_pushed());

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_increment\_by\_1000(void){

increment\_by\_number(1000);

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_increment\_by\_1000\_button\_is\_pushed());

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_increment\_by\_10000(void){

increment\_by\_number(10000);

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_increment\_by\_10000\_button\_is\_pushed());

RGB\_LED\_set\_red();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_start(void){

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_start\_button\_is\_pushed());

timer\_0\_reset\_number\_of\_counts();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print("ODBROJAVANJE -> ");

GLCD\_print\_number\_of\_counts();

state = COUNTDOWN;

GLCD\_remove\_slide\_switch\_state();

RGB\_LED\_set\_yellow();

timer\_0\_reset\_countdown\_timer();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_restart(void){

*uint16\_t* ocr1b = OCR1B;

*uint16\_t* prescaler = PWM\_1\_get\_prescaler();

while(state == THERAPEUTIC\_TREATMENT){

if(TCNT1 > ocr1b)TCCR1B &= 0b11111000;

if(PWM\_1\_get\_prescaler() == 0){

PWM\_1\_set\_prescaler(prescaler);

break;

}

}

PWM\_1\_disable();

PWM\_1\_set\_prescaler(prescaler);

PWM\_1\_save\_the\_sum\_of\_impulses();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print(" ");

GLCD\_print\_impulse\_counter();

GLCD\_print\_sum\_of\_impulses();

GLCD\_print\_slide\_switch\_state();

RGB\_LED\_set\_dark\_blue();

buzzer\_activate\_button\_has\_been\_pushed\_tone();

while(buttons\_and\_switches\_restart\_button\_is\_pushed());

RGB\_LED\_set\_red();

watch\_dog\_timer\_on();

state = STAND\_BY;

while(WAITING\_FOR\_RESTART);

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_countdown(void){

if(timer\_0\_countdown\_is\_over()){

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print(" ");

state = THERAPEUTIC\_TREATMENT;

current\_number\_of\_impulses = PWM\_1\_get\_impulse\_counter();

RGB\_LED\_set\_green();

if(PWM\_1\_get\_impulse\_counter() < PWM\_1\_get\_max\_number\_of\_impulses())PWM\_1\_enable\_non\_inverted\_wave\_form();

}

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_therapeutic\_treatment(void){

if(current\_number\_of\_impulses != PWM\_1\_get\_impulse\_counter()){

GLCD\_print\_impulse\_counter();

GLCD\_print\_sum\_of\_impulses();

current\_number\_of\_impulses = PWM\_1\_get\_impulse\_counter();

}

if(PWM\_1\_get\_impulse\_counter() >= PWM\_1\_get\_max\_number\_of\_impulses()){

PWM\_1\_disable();

PWM\_1\_save\_the\_sum\_of\_impulses();

RGB\_LED\_set\_pink();

GLCD\_print\_impulse\_counter();

GLCD\_print\_sum\_of\_impulses();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print("TERAPIJA JE GOTOVA");

GLCD\_print\_slide\_switch\_state();

buzzer\_activate\_treatment\_is\_finished\_tone();

GLCD\_set\_column\_and\_row(0, 7);

GLCD\_print(" ");

RGB\_LED\_set\_red();

state = STAND\_BY;

}

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_slide\_switch(void){

GLCD\_print\_slide\_switch\_state();

}

/////////////////////////////////////////////////////////////////////////////////////////

void state\_machine\_routine(void){

event = state\_machine\_get\_event();

switch(state){

case STAND\_BY: {

switch(event){

case SLIDE\_SWITCH: state\_machine\_slide\_switch();break;

case RESET: state\_machine\_reset();break;

case SAVE: state\_machine\_save();break;

case CURSOR\_UP: state\_machine\_cursor\_up();break;

case CURSOR\_DOWN: state\_machine\_cursor\_down();break;

case INCREMENT\_BY\_1: state\_machine\_increment\_by\_1();break;

case INCREMENT\_BY\_10: state\_machine\_increment\_by\_10();break;

case INCREMENT\_BY\_100: state\_machine\_increment\_by\_100();break;

case INCREMENT\_BY\_1000: state\_machine\_increment\_by\_1000();break;

case INCREMENT\_BY\_10000: state\_machine\_increment\_by\_10000();break;

case START: state\_machine\_start();break;

case RESTART: state\_machine\_restart();break;

}

break;

}

case COUNTDOWN: {

switch(event){

case RESET: state\_machine\_reset();break;

case PAUSE: state\_machine\_pause();break;

case RESTART: state\_machine\_restart();break;

case NOTHING\_IS\_HAPPENING: state\_machine\_countdown();break;

}

break;

}

case THERAPEUTIC\_TREATMENT: {

switch(event){

case RESET: state\_machine\_reset();break;

case PAUSE: state\_machine\_pause();break;

case RESTART: state\_machine\_restart();break;

case NOTHING\_IS\_HAPPENING: state\_machine\_therapeutic\_treatment();break;

}

break;

}

}

}

/////////////////////////////////////////////////////////////////////////////////////////

/\*

PORTA

PA7 - INPUT (BUTTON SAVE PARAMETERS)

PORTB

PB0 - INPUT (BUTTON GLCD CURSOR UP)

PB1 - INPUT (BUTTON GLCD CURSOR DOWN)

PB2 - INPUT (INCREMENT / DECREMENT)

PB3 - INPUT (BY 1)

PB4 - INPUT (BY 10)

PB5 - INPUT (BY 100)

PB6 - INPUT (BY 1000)

PB7 - INPUT (BY 10000)

PORTD  
 PD0 - INPUT (BUTTON RESTART)  
 PD2 - INPUT (BUTTON START)  
 PD3 - INPUT (BUTTON PAUSE)  
 PD6 - INPUT (BUTTON RESET)

\*/