



UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA



Nikola Ristić

Elektropneumatski generator kompresionih mehaničkih talasa

ommented [N1]: Promeniti pri završetku rada

- MASTER RAD -

Novi Sad, 2020.



UNIVERZITET U NOVOM SADU • FAKULTET TEHNIČKIH
NAUKA

21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNE DOKUMENTACIJSKE INFORMACIJE


1

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 :	Elektropneumatski generator ultrazvuka	
Jezik publikacije, JP :	Srpski	
Jezik izvoda, Jl :	Srpski	
Zemlja publikovanja, ZP :	Srbija	
Uže geografsko područje, UGP :	AP Vojvodina	
Godina, GO :	2020.	
Izdavač, IZ :		
Mesto i adresa, MA :	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 Srđan Tegeltija	
	Član: Dr Dragan Rajnović	Potpis mentora
	Član, mentor: Dr Dragan Šešlija	

ommented [N2]: Promeniti kad rad bude gotov.

ommented [N3]: ss Ovo je potrebno promeniti posle izvršetka rada

ommented [N4]: . Ovo je potrebno promeniti kad se utvrdi datum odbrane


	UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES 21000 NOVI SAD, Trg Dositeja Obradovića 6	
	KEY WORDS DOCUMENTATION	

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 ultrasound		
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. Srđan Tegeltija	
	Member:	Ph.D. Dragan Rajnović	Menthor's signature
	Member, Mentor:	Ph.D. Dragan Šešlija	

ommented [N5]: Promeniti posle završetka rada

ommented [N6]: s Ovo je potrebno promeniti posle završetka da

ommented [N7]: Ovo je potrebno promeniti kada se utvrdi datum odbrane

	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:	<input type="checkbox"/> 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:

ELEKREOPNEUMATSKI GENERATOR ULTRAZVUKA

ommented [N8]: Promeniti posle završetka rada

TEKST ZADATKA:

<ul style="list-style-type: none"> - Upoznati se sa teorijskim osnovama generisanja zvučnog polja primenom sudara čvrstih tela. - Projektovati elektropneumatski aktuator koji generiše ultrazvučno polje. - Komentarisati dobijene rezultate i ustanoviti potrebna testiranja u cilju optimizacije sistema.

ommented [N9]: Promeniti posle završetka rada

Rukovodilac studijskog programa:	Mentor rada:

Primerak za: <input type="checkbox"/> - Studenta; <input type="checkbox"/> - Mentora
--

Sadržaj

ommented [N10]: Dodati na kraju zbog stalnog
svežavanja

Spisak slika

ommented [N11]: Dodati na kraju zbog stalnog
svežavanja

Spisak tabela

ommented [N12]: Dodati na kraju zbog stalnog
svežavanja

1. Uvod

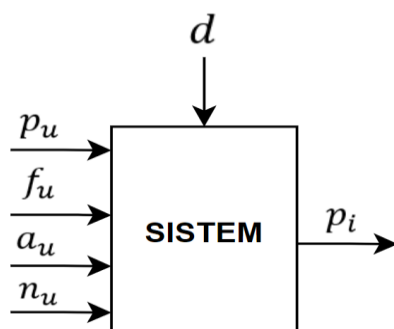
Elektropneumatska automatizacija procesa predstavlja jedan od najprimenjenijih načina za ostvarivanje željenog rada industrijskih sistema. U opštem slučaju, izvršni organ koristi vazduh pod pritiskom kao energetski izvor za postizanje potrebnog funkcionisanja, dok se njegovo upravljanje ostvaruje upotrebom 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 elektropneumatski izvršni organ koji generiše kompresione mehaničke talase, kao i korisnički interfejs koji je namenjen za njegovo upravljanje. Postoji mogućnost da se ovakav sistem može primeniti u domenu medicine koji se bavi fizikalnom terapijom telesnih povreda [1], ali su za to potrebna dodatna ispitivanja koja nisu obavljena u okviru 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 pojavljuje unutar procesa u vidu 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:

ommented [N13]: Uvod promeniti posle završetka rada.
odati i pasus
koji opisuje šta je rađeno po poglavljima



Slika 1 - Šematski prikaz sistema

gde je p_u ulazni vazdušni pritisak, f_u je frekvencija impulsno širinske modulacije, a_u predstavlja faktor ispune *PWM* signala, a n_u je broj perioda istog. Izlazna veličina p_i je pritisak talasnog polja koje generiše izvršni organ, dok je d 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.

2. Energija sistema

Commented [N14]: Možda promeni naziv

2.1. Zakon održanja ukupne energije

2.1.1. 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:

$$A = \vec{F} \vec{s} \quad (1)$$

odnosno:

$$A = F s \cos \alpha \quad (2)$$

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

$$A = \sum_{j=1}^n \Delta A_j = \sum_{j=1}^n \left(F_j \Delta s_j \cos(\angle(\vec{F}_j, \Delta \vec{s}_j)) \right) \quad (3)$$

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

$$A = \lim_{\Delta s_j \rightarrow 0} \sum_{j=1}^n (\vec{F}_j \Delta \vec{s}_j) = \int_{s_0}^{s_1} \vec{F} d\vec{s} \quad (4)$$

gde je s_0 početna takča putanje \vec{s}_j , a s_1 je krajnja.

S obzirom da se dejstvo sila manifestuje kao promena impulsa \vec{p} u obliku:

$$\vec{F} = \frac{d\vec{p}}{dt} = m \frac{d\vec{v}}{dt} \quad (5)$$

jednačina (4) može se predstaviti u formi:

$$A = \int_{s_0}^{s_1} \vec{F} d\vec{s} = m \int_{v_0}^{v_1} \vec{v} d\vec{v} \quad (6)$$

pri čemu je v_0 početna brzina tela, a v_1 je krajnja. Rešavanjem integralne jednačine (6) dobija se izraz za promenu kinetičke energije kao:

$$m \int_{v_0}^{v_1} \vec{v} d\vec{v} = \frac{m(v_1)^2}{2} - \frac{m(v_0)^2}{2} = E_{k_1} - E_{k_0} = \Delta E_k \quad (7)$$

što znači da je ukupan rad sistema:

$$A = \Delta E_k \quad (8)$$

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:

$$\Delta E_k = E_{k_1} \quad (9)$$

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

$$A = E_{k_1} = \frac{mv_1^2}{2} \quad (10)$$

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

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

$$dA = \vec{F} d\vec{s} \quad (11)$$

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

$$dA = dE_k \quad (12)$$

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

2.1.2. 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 \vec{r}_0) na drugo (definisano radijus vektorom \vec{r}_1), tj. jednaka je negativnom radu konzervativnih sila:

$$\Delta E_p = -A_k \quad (13)$$

ili drugačije zapisano:

$$E_p(\vec{r}_1) - E_p(\vec{r}_0) = - \int_{r_0}^{r_1} \vec{F}_k d\vec{r} \quad (14)$$

gde ΔE_p predstavlja razliku potencijalnih energija $E_p(\vec{r}_1)$ i $E_p(\vec{r}_0)$, a A_k je rad geometrijskog zbira svih konzervativnih sila koje deluju na sistem \vec{F}_k .

Ukoliko se potencijalna energija u tački \vec{r}_0 usvoji kao referentna (što znači da je potencijalna energija u toj tački jednaka nuli), jednačina (13) može se prikazati u obliku:

$$E_p(\vec{r}_1) = -A_k \quad (15)$$

ili u opštem slučaju:

$$-E_p(\vec{r}) = A_k \quad (16)$$

pri čemu \vec{r} predstavlja vektor položaja tela u odnosu na radijus vektor \vec{r}_0 . Diferenciranjem jednačine (16) dobija se:

$$dA_k = -dE_p \quad (17)$$

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

2.1.3. Mehanička energija

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

$$dA = dA_k \quad (18)$$

odakle sledi:

$$dA - dA_k = 0 \quad (19)$$

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

$$d(E_k + E_p) = 0 \quad (20)$$

što znači da je:

$$E_k + E_p = \text{const} \quad (21)$$

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

$$E_m = \text{const} \quad (22)$$

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:

$$dA = dA_k + dA_n \quad (23)$$

odnosno:

$$dA - dA_k = dA_n \quad (24)$$

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

$$d(E_k + E_p) = dA_n \quad (25)$$

odakle sledi:

$$dE_m = dA_n \quad (26)$$

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:

$$E_m = A_n + E_{m_0} \quad (27)$$

pri čemu je E_{m_0} 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.

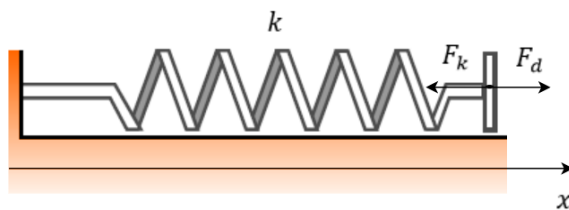
U opštem slučaju, često se početna mehanička energija sistema usvaja kao referentna. To znači da je njena vrednost jednaka nuli, a jednačina (27) se može napisati u obliku:

$$E_m = A_n \quad (28)$$

2.2. 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 analzi potencijalne energije elastične deformacije modelira oprugom. Veza između opterećenja i deformacije tela opisuje se Hukovim zakonom [10].

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



Slika 2 - Deformisana opruga

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

$$F_k = -kx \quad (29)$$

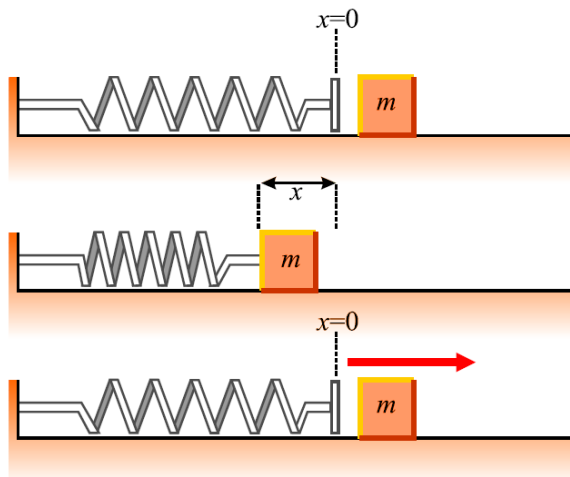
Uvrštavanjem jednačine (29) u integralnu jednačinu (4), dobija se da je rad sile koja teži da vrati oprugu u prvobitni oblik:

$$A_k = -\frac{kx^2}{2} \quad (30)$$

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

$$E_p = \frac{kx^2}{2} \quad (31)$$

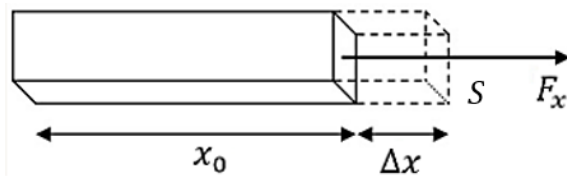
Potencijalna energija elastične deformacije može se shvatiti kao energija koja je akumulirana u deformisanoj opruzi, bilo da je sabijena ili istegnuta. Kada bi sila sabijene opruge delovala na telo mase m kao na slici 3, telo bi se pomerilo u smeru dejstva sile opruge (pod pretpostavkom da ne postoji 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 F_x kao na što je prikazano na slici 4, njena deformacija Δx može se izračunati kao:

$$\Delta x = \frac{F_x}{k} \quad (32)$$



Slika 4 - Deformacija tela pri dejstvu sile zatezanja

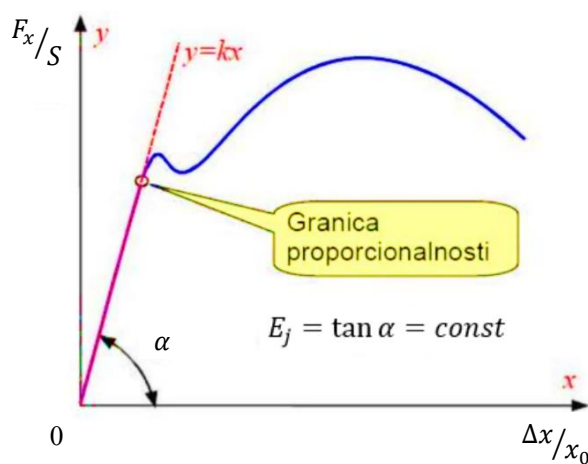
Jednačina (32) može se napisati u formi:

$$\Delta x = F_x \frac{x_0}{SE_j} \quad (33)$$

gde je x_0 dužina tela pre deformacije, S je površina poprečnog preseka tela, a E_j je Jangov modul elastičnosti. Oдавde sledi:

$$k = \frac{SE_j}{x_0} \quad (34)$$

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 naprežanja koji je prikazan na slici 5.



Slika 5 - Oblast važenja linearne deformacije

3. Mehanički talasi

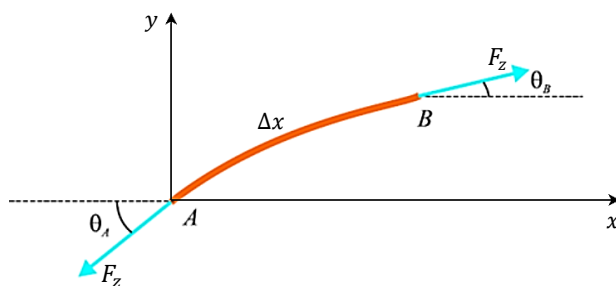
Oscilovanje čestice oko svog ravnotežnog položaja u neprekidnoj elastičnoj sredini, pri čemu se energija oscilovanja prenosi sa jedne na drugu česticu, naziva se mehanički talas. Da bi postojao takav talas, neophodno je da postoji:

- izvor poremećaja koji premešta česticu iz svog ravnotežnog položaja
- medijum u kojem može da se izvede prostiranje talasa
- 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 [3]. Takav slučaj postoji pri radu izvršnog organa koji je tema ove teze.

3.1. Talasna jednačina

Jednačina koja opisuje vezu brzine mehaničkog talasa sa vremenskom i prostornom promenom njegove elongacije, naziva se talasna jednačina [4]. Kako bi je odredili, posmatrajmo prostiranje talasa na zategnutoj žici koji se dešava usled pobuđivanja oscilacijama jednog njenog kraja. Na slici 3, prikazan je jedan delić dužine žice Δx koji je zategnut silom zatezanja F_z , pri čemu krajevi tog delića zaklapaju uglove θ_A i θ_B sa x osom. Pretpostavimo da se oscilacije dešavaju samo u vertikalnom pravcu y .



Slika 6 - Delić žice pobuđen na oscilacije

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

$$F_y = F_z(\sin \theta_B - \sin \theta_A) \quad (35)$$

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

$$F_y = F_z(\tan \theta_B - \tan \theta_A) \quad (36)$$

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

$$F_y = F_z \left(\left. \frac{\partial y}{\partial x} \right|_B - \left. \frac{\partial y}{\partial x} \right|_A \right) \quad (37)$$

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

$$F_y = \Delta m \frac{\partial^2 y}{\partial t^2} \quad (38)$$

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

$$\Delta m = \gamma \Delta x \quad (39)$$

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

$$\gamma \Delta x \frac{\partial^2 y}{\partial t^2} = F_z \left(\left. \frac{\partial y}{\partial x} \right|_B - \left. \frac{\partial y}{\partial x} \right|_A \right) \quad (40)$$

odnosno:

$$\frac{\gamma}{F_z} \frac{\partial^2 y}{\partial t^2} = \frac{\left. \frac{\partial y}{\partial x} \right|_B - \left. \frac{\partial y}{\partial x} \right|_A}{\Delta x} \quad (41)$$

Pošto je parcijalni izvod bilo koje funkcije po definiciji:

$$\frac{\partial y}{\partial x} = \lim_{\Delta x \rightarrow 0} \frac{y(x + \Delta x) - y(x)}{\Delta x} \quad (42)$$

jednačina (41) može se zapisati kao:

$$\frac{\gamma}{F_z} \frac{\partial^2 y}{\partial t^2} = \frac{\partial^2 y}{\partial x^2} \quad (43)$$

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

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

$$y = A \sin(kx - \omega t) \quad (44)$$

gde je A amplituda, ω je kružna frekvencija, a k je talasni broj koji se definiše kao:

$$k = \frac{2\pi}{\lambda} \quad (45)$$

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

$$\frac{\partial^2 y}{\partial x^2} = -\omega^2 A \sin(kx - \omega t) \quad (46)$$

$$\frac{\partial^2 y}{\partial t^2} = -k^2 A \sin(kx - \omega t) \quad (47)$$

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

$$-\frac{\gamma \omega^2}{F_z} \sin(kx - \omega t) = -k^2 \sin(kx - \omega t) \quad (48)$$

odakle sledi:

$$k^2 = \frac{\gamma \omega^2}{F_z} \quad (49)$$

Kako se brzina prostiranja talasa u računa kao:

$$u = \frac{\omega}{k} \quad (50)$$

talasni broj se može izraziti kao:

$$k = \frac{\omega}{u} \quad (51)$$

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

$$u = \sqrt{\frac{F_z}{\gamma}} \quad (52)$$

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

$$u^2 \frac{\partial^2 y(x, t)}{\partial x^2} = \frac{\partial^2 y(x, t)}{\partial t^2} \quad (53)$$

Jednačina talasa 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:

$$c^2 \left(\frac{\partial^2 \Pi}{\partial x^2} + \frac{\partial^2 \Pi}{\partial y^2} + \frac{\partial^2 \Pi}{\partial z^2} \right) = \frac{\partial^2 \Pi}{\partial t^2} \quad (54)$$

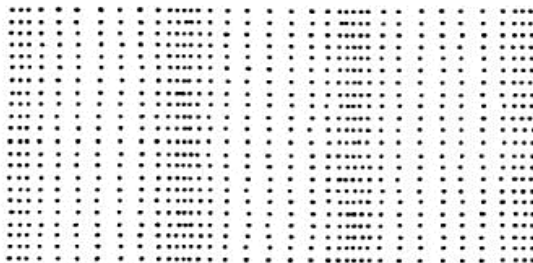
pri čemu je $\Pi = \Pi(x, y, z, t)$ funkcija pomeraja žice, a c je faktor brzine prostiranja talasa. Na sličan način može se odrediti talasna jednačina različitih tipova talasa u zavisnosti od medijuma kroz koji se prostiru. Ako se

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 talasna fronta, a to su ravanski i sferni [3]. 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.

3.2. 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 [3]. 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.

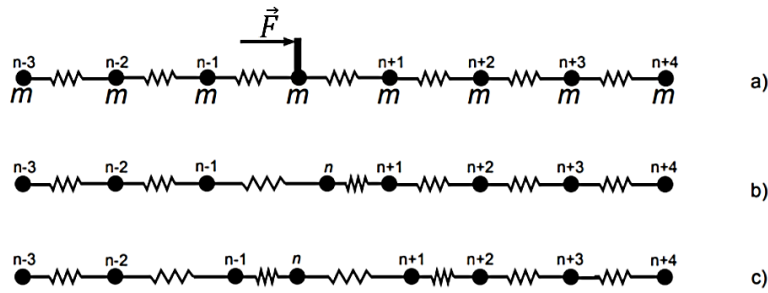
U čvrstim telima mehanički talasi se klasifikuju u zavisnosti od pravca njihovog prostiranja u odnosu na pravac oscilovanja čestica. Talasi kod kojih se oscilacije dešavaju u pravcu koji je paralelan sa pravcem prostiranja talasa nazivaju se kompresioni (longitudinalni). Primer takvih talasa predstavljen 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 m 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 b). Poremećaj se dalje prenosi na susedne čestice kao što se može videti na segmentu slike pod c). 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 odnosu na promenu sile, ali bi elongacije čestice 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:

$$\frac{d\vec{F}}{dt} = m\vec{\varepsilon} \neq 0 \quad (55)$$

pri čemu mora biti ispunjen uslov:

$$\varepsilon \gg 0 \quad (56)$$

Veličina $\vec{\varepsilon}$ naziva se trzaj tela i definiše se kao:

$$\vec{\varepsilon} = \frac{d\vec{a}}{dt} \quad (57)$$

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

$$u_l = \sqrt{\frac{E_j}{\rho}} \quad (58)$$

gde je u_l brzina kompresionog talsa, a ρ je gustina medijuma. Talasna jednačina pri elongaciji koja se opisuje funkcijom $\Psi(x, t)$ je tada:

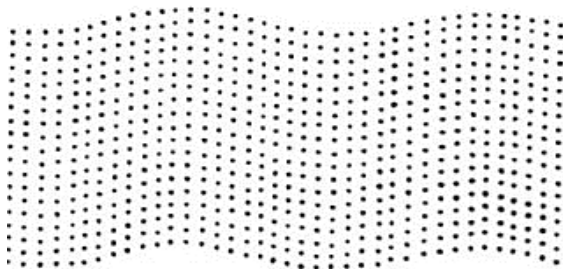
$$u_l^2 \frac{\partial^2 \Psi(x, t)}{\partial x^2} = \frac{\partial^2 \Psi(x, t)}{\partial t^2} \quad (59)$$

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 E_s :

$$u_t = \sqrt{\frac{E_s}{\rho}} \quad (60)$$

pri čemu je u_t brzina transferzalnog talasa. Talasna jednačina savijajućih talasa pri elongaciji koja se opisuje funkcijom $\Phi(x, t)$ je:

$$u_t^2 \frac{\partial^2 \Phi(x, t)}{\partial x^2} = \frac{\partial^2 \Phi(x, t)}{\partial t^2} \quad (61)$$



Slika 9 - Transferzalni talasi

3.3. Kompreioni talasi

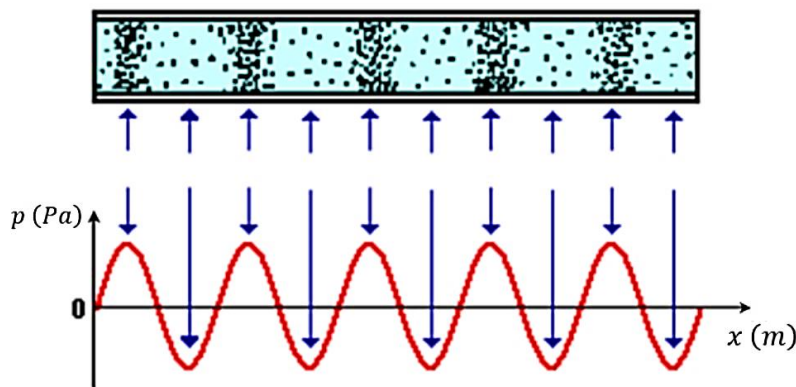
Longitudinalni talasi mogu se prostirati kroz čvrsta tela, ali za razliku od transferzalnih, mogu se kretati i u sredinama koje su tečnog ili gasovitog agregatnog stanja. Njihova talasna jednačina može se predstaviti izrazom (59), ali će njihova brzina zavisiti od medijuma u kojem se nalaze. Međutim, 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):

$$u^2 \frac{\partial^2 p(x, t)}{\partial x^2} = \frac{\partial^2 p(x, t)}{\partial t^2} \quad (62)$$

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

$$c^2 \left(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} \right) = \frac{\partial^2 P}{\partial t^2} \quad (63)$$

gde je $P = P(x, y, z, t)$ 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 ilustriramo 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:

$$u_f = \sqrt{\frac{\beta}{\rho}} \quad (64)$$

pri čemu je u_f brzina longitudinalnog talasa u fluidu, a β je modul stišljivosti istog. Opšta veza između elongacije i pritiska u fluidu može se izraziti jednačinama [17]:

$$p(x, t) = -\beta \frac{\partial \Psi(x, t)}{\partial x} \quad (65)$$

$$\frac{\partial p(x, t)}{\partial x} = -\rho \frac{\partial^2 \Psi(x, t)}{\partial t^2} \quad (66)$$

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

$$\Psi(x, t) = \Psi_0 \cos(kx - \omega t) \quad (67)$$

gde je Ψ_0 amplituda elongacije. U tom slučaju, promena pritiska može se opisati jednačinom [16]:

$$p(x, t) = \Psi_0 k \beta \sin(kx - \omega t) = p_0 \sin(kx - \omega t) \quad (68)$$

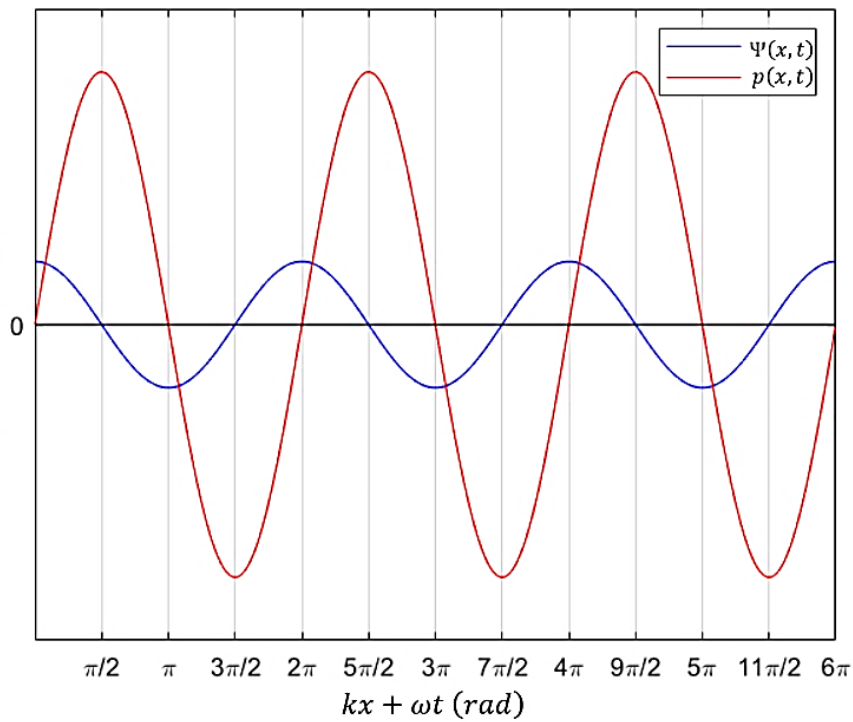
pri čemu je p_0 amplituda pritiska. Odavde sledi da je veza između amplitude elongacije i amplitude pritiska:

$$p_0 = \Psi_0 k \beta \quad (69)$$

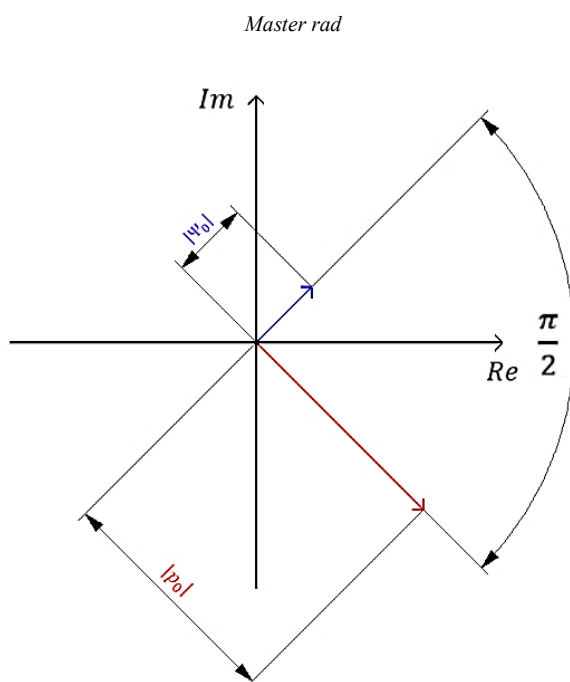
Na slici 11 predstavljen je grafik promene signala pritiska i elongacije, a na slici 12 se može videti fazorski dijagram istih. Na njima se može primetiti da

Master rad

je talasna dužina, frekvencija i talasna brzina ista za oba signala, ali da fazor elongacije prednjači u odnosu na fazor pritiska za $\pi/2$ (*rad*). To znači da je apsolutna vrednost amplitude pritiska maksimalna kad je elongacija jednaka nuli i obratno.



Slika 11 - Signali pritiska i elongacije

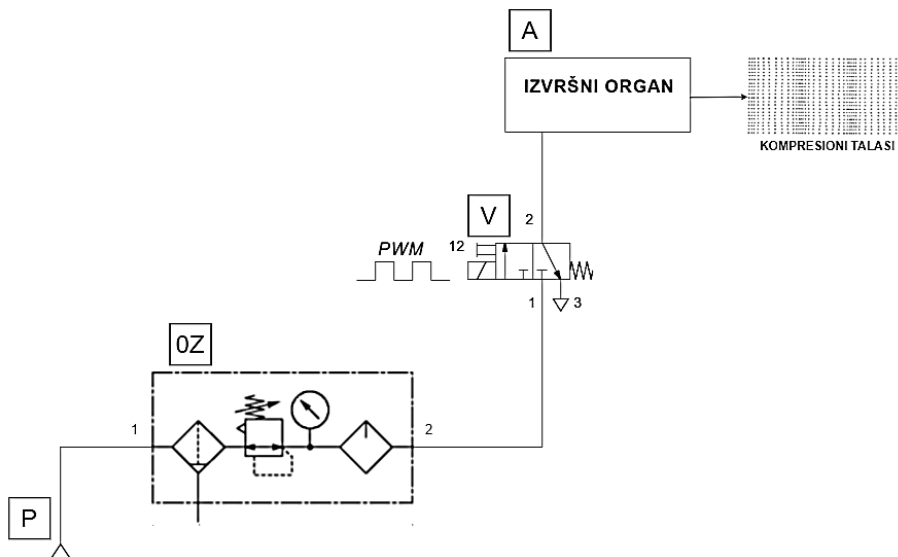


Slika 12 - Fazorski dijagram pritiska i elongacije

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 $p(t)$. Merni instrument koji je namenjen za merenje takve promene pritiska je mikrofoni.

4. 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.



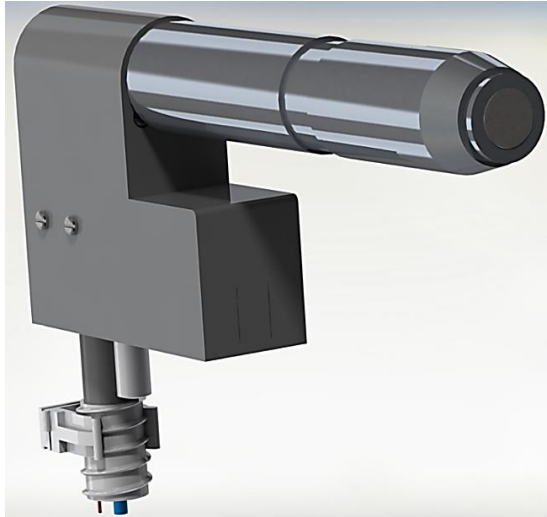
Slika 13 – Elektropneumatska šema sistema

4.1. 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.

ommented [N15]: Ne znam kako mogu staviti nestandardni simbol

Master rad



Slika 14 - CAD model izvršnog organa

Slika 15 - Realan model izvršnog organa

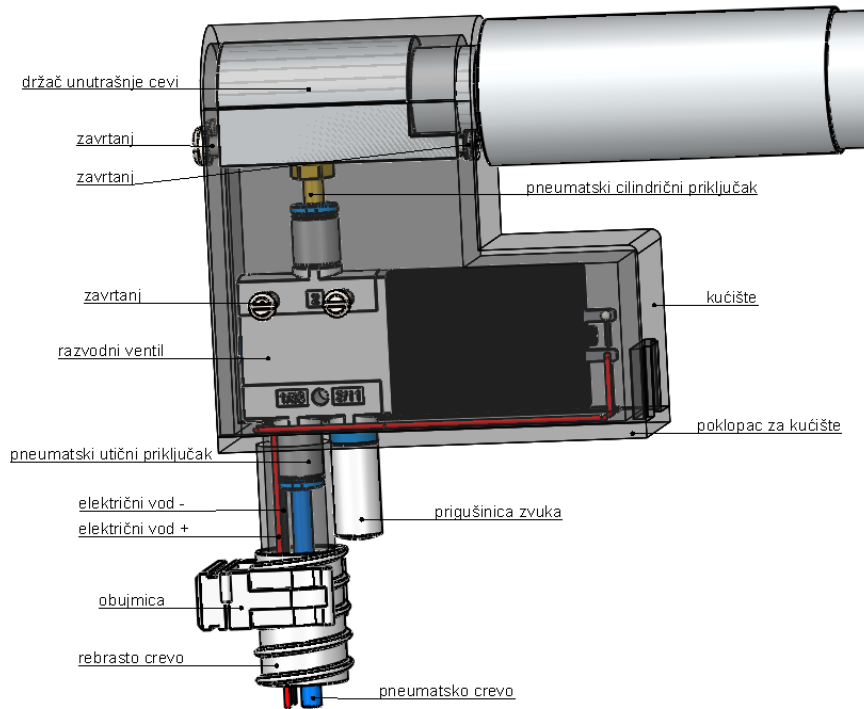
4.1.1. Integracija razvodnog ventila u izvršni organ

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 [18] (oko 2 ms). Takođe, kompatibilan je za rad aktuatora pošto je namenjen za protok **zauljenog** vazduha pod pritiskom. 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 obavlja. Ono je stegnuto za cilindrični segment poklopca kućišta dejstvom objumice.

ommented [N16]: Ubaciti realnu sliku posle 3D štampanja kućišta

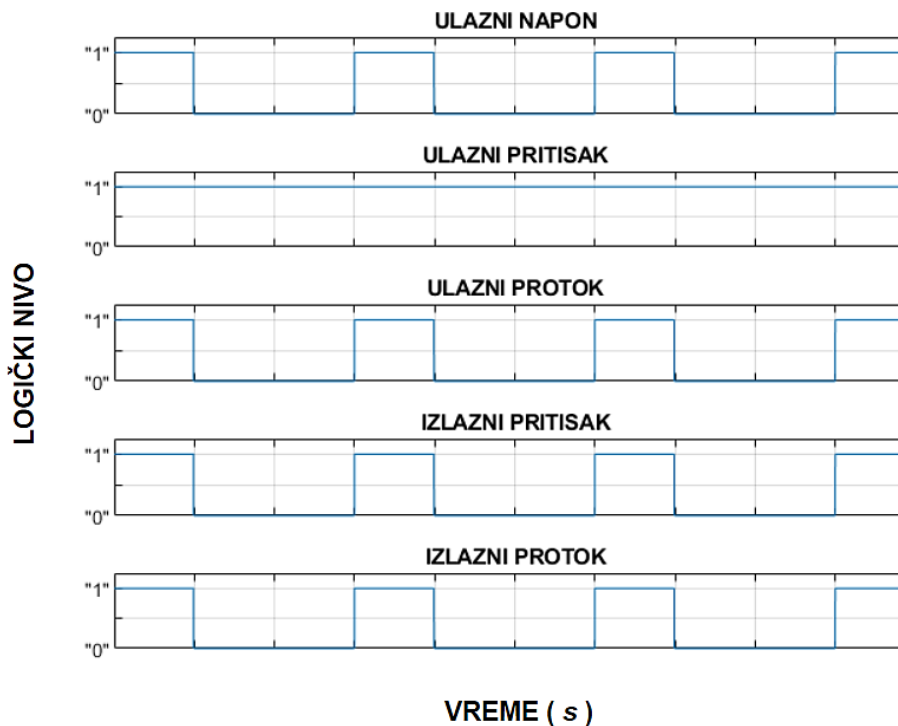
ommented [N17]: U tehničkim specifikacija piše da je imenljiv i u slučaju zauljenog vazduha pod pritiskom



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.

Master rad

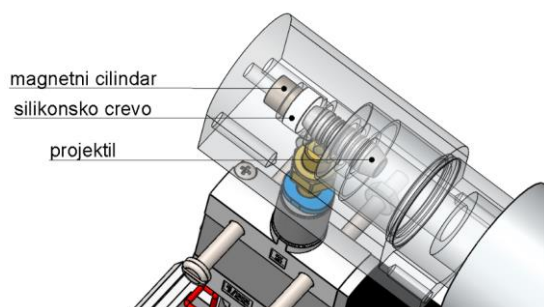


Slika 17 - Signali razvodnog ventila za vreme radnog režima izvršnog organa

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 ispunje upravljačkog signala.

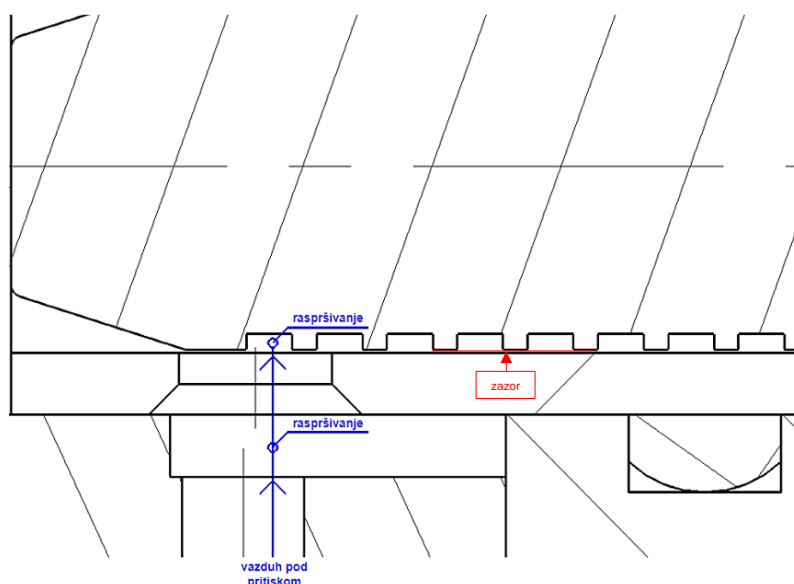
4.1.2. Pogon projektila

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.



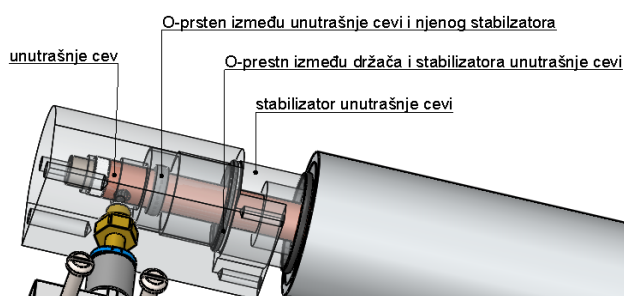
Slika 18 - Početni položaj projektila

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.



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

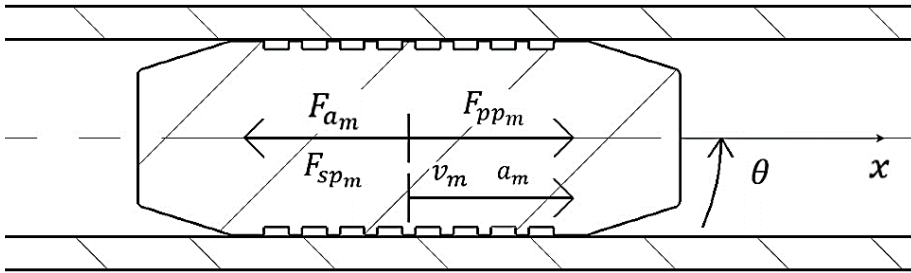
O-prsten između unutrašnje cevi i njenog stabilizatora ima dve funkcije u sistemu. Jedna je sprečavanje vibracije i zakretanje unutrašnje cevi za vreme rada izvršnog organa, a druga je zaptivanje unutrašnje sredine stabilizatora kako ne bi došlo do povećanja pritiska u njoj.

4.1.3. 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 [5]. Eksperimentalno je utvrđeno da privlačno dejstvo cilindra ne deluje na projektil ukoliko je distanca između njih veća 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 θ . Osa x je kolinearna sa uzdužnom osom unutrašnje cevi, a v_m i a_m 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. F_{a_m} – aktivna komponenta sile zemljine teže
2. F_{pp_m} – sila koja postoji usled pogonskog pneumatskog pritiska koji ubrzava projektil prema membrani (u nastavku pneumatska pogonska sila)
3. F_{sp_m} – 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:

$$ma_m = -F_{a_m} + F_{pp_m} - F_{sp_m} \quad (70)$$

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

$$F_{a_m} = mg \sin \theta \quad (71)$$

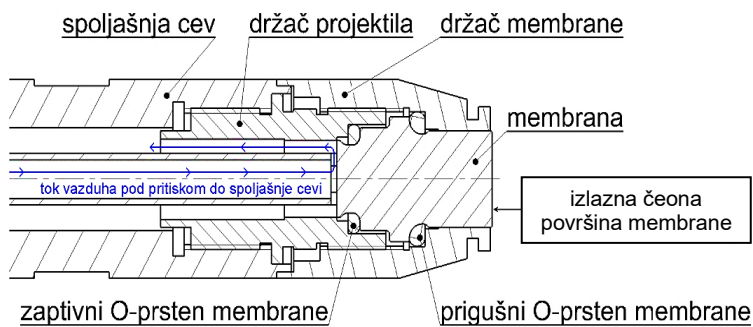
gde je g konstanta gravitacionog ubrzanja, može se izračunati intenzitet aktivne komponente sile zemljine teže [6]. U skladu sa time mogu se napisati izrazi za brzinu i ubrzanje projektila dok se kreće prema membrani kao:

$$v_m = v_{0_m} - gt \sin \theta + v_{pp_m} - v_{sp_m} \quad (72)$$

$$a_m = a_{0_m} - g \sin \theta + a_{pp_m} - a_{sp_m} \quad (73)$$

pri čemu je t vremenski interval između početnog i posmatranog trenutka kretanja, v_{pp_m} i a_{pp_m} su brzina i ubrzanje koje metak dobija usled uticaja pneumatske pogonske sile, a v_{sp_m} i a_{sp_m} su brzina i ubrzanje projektila zbog dejstva sile suprotstavljajućeg pritiska. Početna brzina v_{0_m} i ubrzanje a_{0_m} 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, zapremine komora u aktuatoru, kao i od pritiska i protoka ulaznog vazduha pod pritiskom

4.1.4. Sudar projektila i membrane

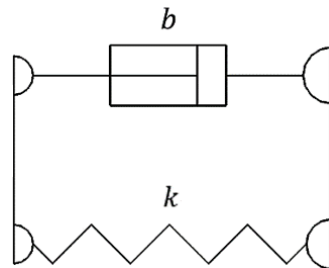
Sudar projektila i membrane (sonde, talasovoda) predstavlja ključni potproces koji se dešava za vreme rada sistema. Ovakav tip sudara naziva se centralni čeon sudar dva tela [2][8]. Za takav događaj važi:

- postoji prava linija koja je paralelna sa pravcem kretanja tela koja učestvuju u sudaru i prolazi kroz centre masa istih (centralni sudar)
- oba tela pre i posle sudara se kreću duž istog pravca (čeon 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 [7]. 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 [9]. 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 k i prigušnice (amortizera) faktora prigušenja b na način koji je prikazan na slici 23.



Slika 23 - Kelvin-Vojtovo telo

Dinamičko svojstvo Kelvin-Vojtovog tela opisuje se jednačinom:

$$F_{KV} = b\dot{d} + kd \quad (74)$$

gde je F_{KV} sila Kelvin-Vojtovog tela, a d je elongacija istog. Eksperimentalno je utvrđeno da Kelvin-Vojtovo telo može dovoljno blizu da aproksimira visko-elastične karakteristike O-prstena [12] i postoje razvijene metode kojima se mogu ustanoviti vrednost parametara b i k . 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 [11]. 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.

Master rad

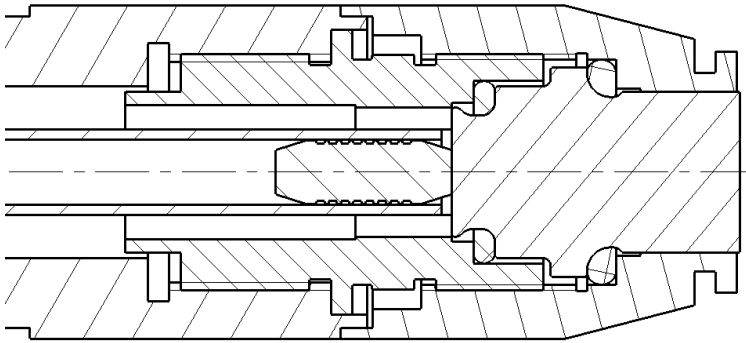


Slika 25 - Hukovo telo

U slučaju redne veze dva Hukova tela krutosti k_1 i k_2 , ekvivalentna krutost opruga k_{12} može se izračunati kao:

$$k_{12} = \frac{k_1 k_2}{k_1 + k_2} \quad (75)$$

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 D i posmatra se u pravcu koji je paralelan sa njima. Takođe, ose membrane i projektila su pod nagibom koji je definisan uglom θ 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:

$$m\ddot{x} = -\frac{k_m k_M}{k_m + k_M}(x - y) - F_{a_m} + F_{pp_m} - F_{sp_m} \quad (76)$$

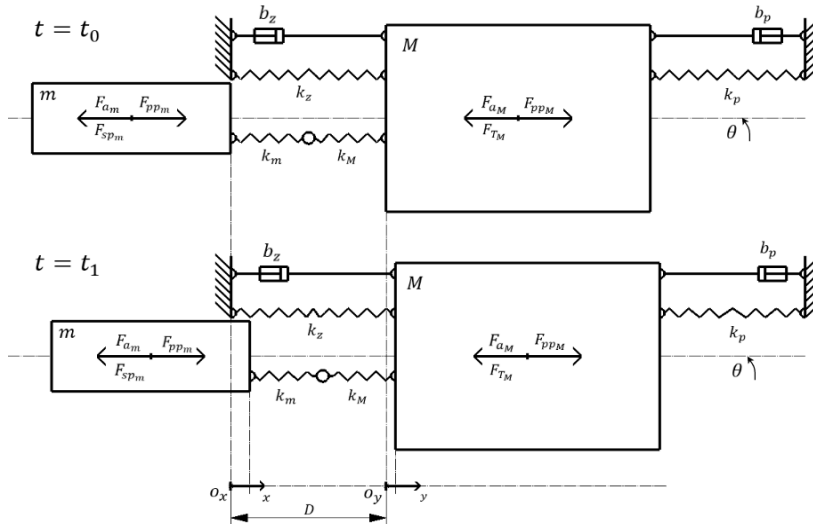
$$M\ddot{y} = \frac{k_m k_M}{k_m + k_M}(x - y) + k_z y + k_z \dot{y} - k_p y - b_p \dot{y} - F_{a_M} + F_{pp_M} - F_{T_M} \quad (77)$$

pri čemu su poznata dva početna uslova:

$$x(t_0) = 0 \quad (78)$$

$$y(t_0) = 0 \quad (79)$$

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.	t_0	početni trenutak sudara
2.	t_1	proizvoljni trenutak sudara
3.	m	masa projektila
4.	M	masa membrane
5.	b_z	faktor prigušenja zaptivnog O-prstena membrane
6.	b_p	faktor prigušenja prigušnog O-prstena membrane
7.	k_z	koeficijent krutosti zaptivnog O-prstena membrane
8.	k_p	koeficijent krutosti prigušnog O-prstena membrane
9.	k_m	koeficijent krutosti projektila
10.	k_M	koeficijent krutosti membrane
11.	F_{a_m}	aktivna komponenta sile zemljine teže koja deluje na projektil
12.	F_{a_M}	aktivna komponenta sile zemljine teže koja deluje na membranu
13.	F_{pp_m}	pneumatska pogonska sila koja deluje na projektil
14.	F_{pp_M}	sila vazduha pod pritiskom koja deluje na membranu
15.	F_{sp_m}	sila suprotstavljajućeg pritiska koja deluje na projektil
16.	F_{T_M}	sila trenja koja deluje na membranu
17.	O_x	koordinatni početak ose x
18.	O_y	koordinatni početak ose y
19.	D	distanca između koordinatnih početaka O_x i O_y
20.	θ	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 F_{a_m} može se odrediti i intenzitet sile aktivne komponente zemljine težee koja deluje na membranu. S obzirom da su masa membrane M i ugao nagiba θ poznate veličine, intenzitet sile F_{a_M} se može izračunati primenom obrasca:

$$F_{a_M} = Mg \sin \theta \quad (80)$$

Takođe, može se odrediti i sila trenja koja deluje na talasovod kao [6]:

$$F_{T_M} = \mu Mg \cos \theta \quad (81)$$

gde je μ koeficijent trenja klizanja.

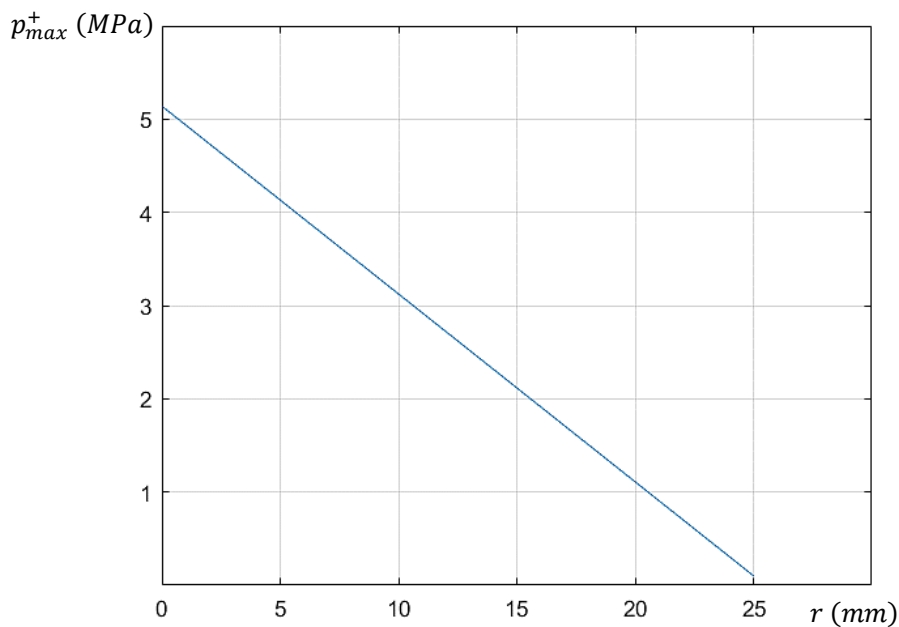
4.1.5. Izlazni signal

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 [3].

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 [19]. 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 slična talasnoj impedansi tkiva jer se ono pretežno sastoji od vode [1]).

Ciljana maksimalna pozitivna promena pritiska duž ose talasovda 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 $76,5^\circ$. Na slici 29 prikazan je geometrijski oblik talasnog polja.

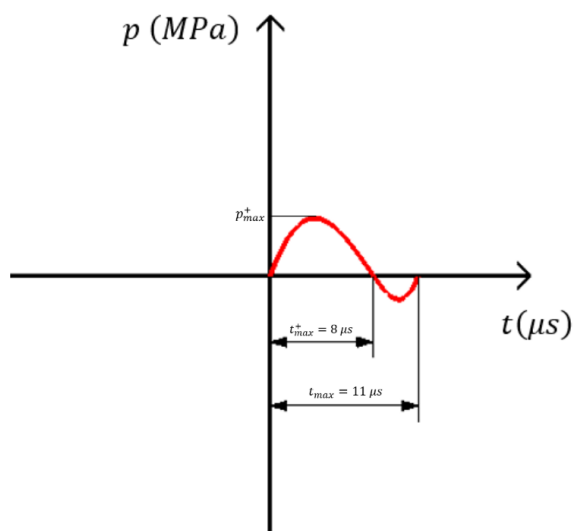


Slika 28 - Promena maksimalne pozitivne vrednosti pritiska duž ose membrane u odnosu na centar njene izlazne čeone površine



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 $11 \mu\text{s}$, pri čemu vreme trajanja pozitivne promene pritiska treba da iznosi maksimalno $8 \mu\text{s}$. Grafička ilustracija navedenih vremenskih karakteristika data je na slici 30.



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

4.1.6. 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 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 dospao 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.

4.1.7. 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 izvršnog 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].

4.2. Upravljačka kutija

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

Master rad



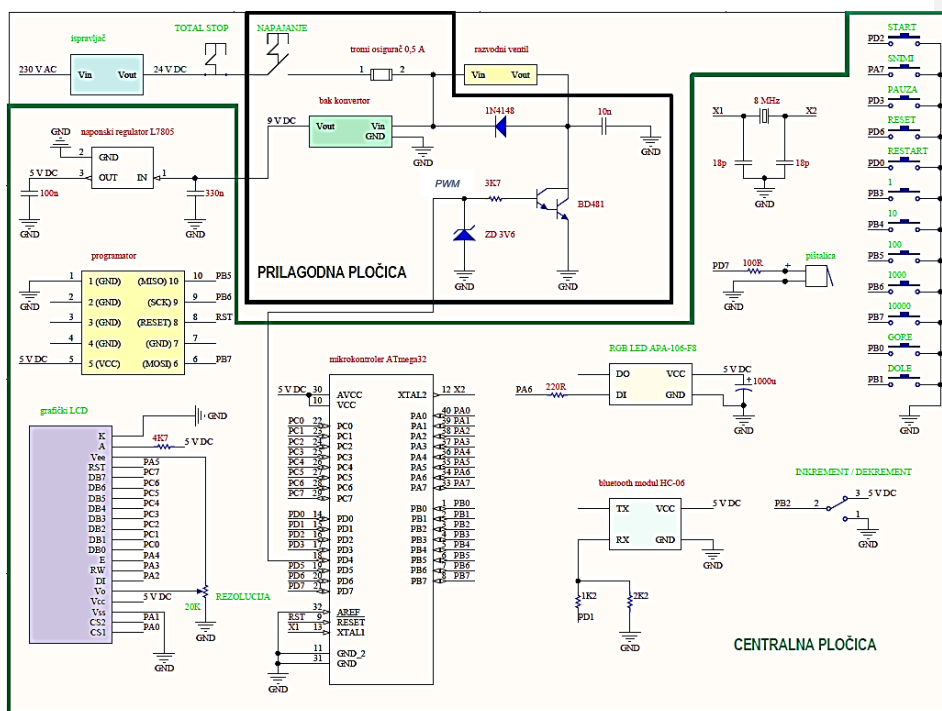
Slika 31 - Upravljačka kutija



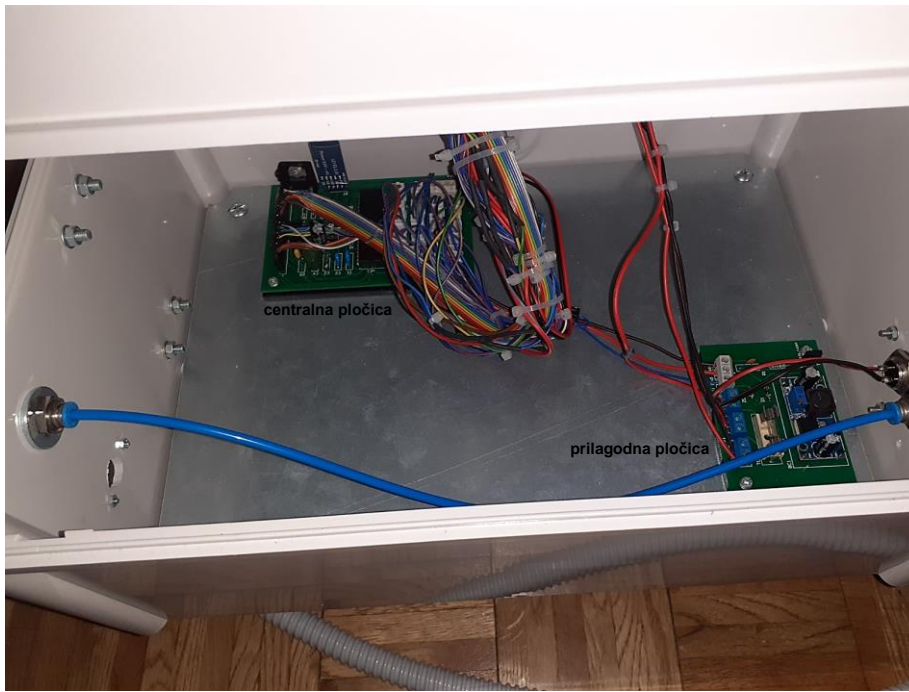
Slika 32 - Tasteri

4.2.1. 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 komponenta (prilagodna pločica). Na slici 33 može se videti šematski prikaz upravljačke elektronike sistema. Zelenom bojom naznačeni se nazivi komponenta koji su dostupni korisniku za direktnu interakciju. Način na koji su štampane pločice montirane unutar upravljačke kutije prikazan je na slici 34.



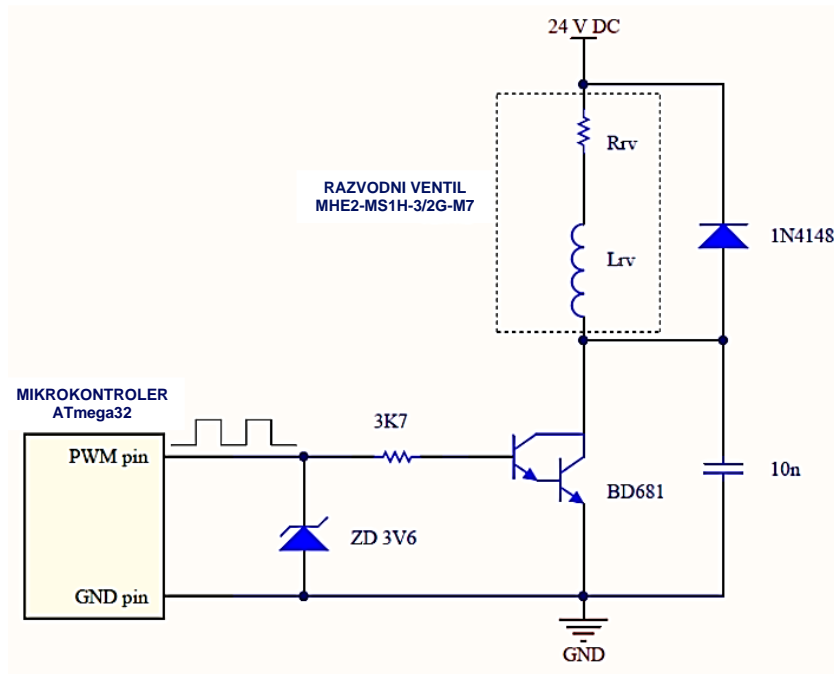
Slika 33 - Šematski prikaz upravljačke elektronike sistema



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 upravlje razvodnim ventilom izdvojeno je na slici 34. Električni model razvodnika predstavljen je rednom vezom otpornika i kalema [18]. 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* singala na izlaznom pinu mikrokontrolera su približno ista kao i izlazna pneumatska impulsno širinska modulacija razvodnog ventila.

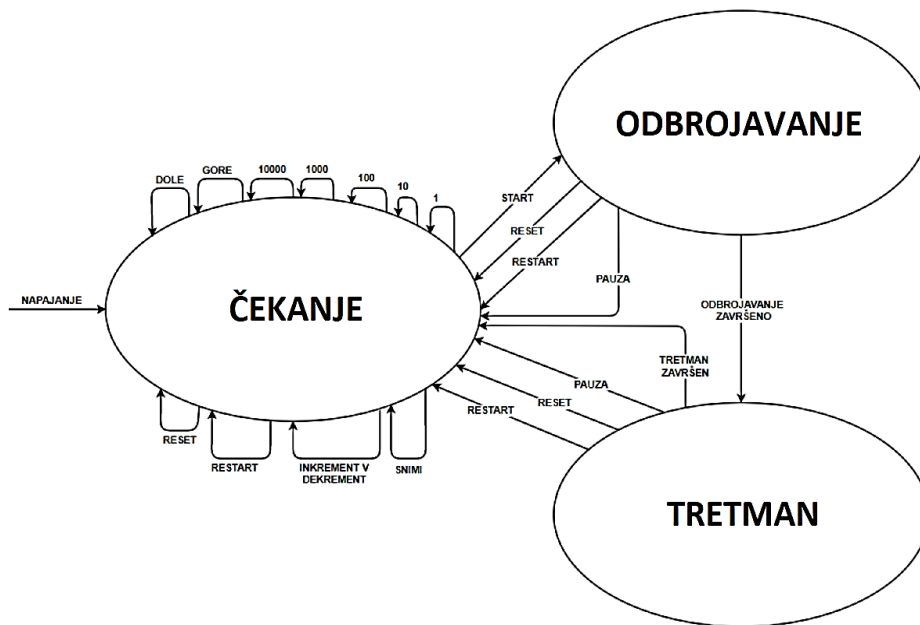


Slika 35 - Električno potkolo namenjeno za upravljanje razvodnim ventilom

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 komponenata 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.

4.2.2. Firmverski algoritam

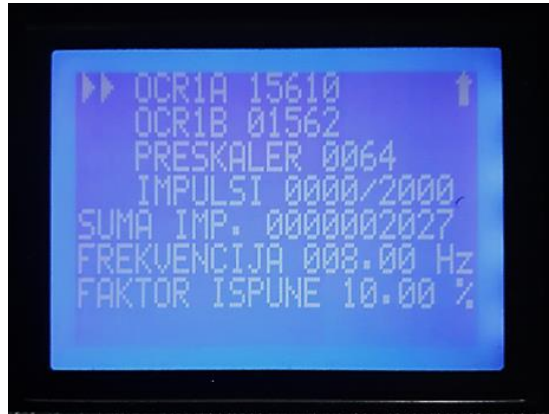
Firmverski algoritam zasnovan je na principu rada konačne mašine stanja. Na slici 36 predstavljen je blok dijagram algoritma na kojem se mogu videti mogući događaji i stanja (režimi) automata. Kompletan programski kod dat je u prilogu 3.



Slika 36 - Konačna mašina stanja

Kao što se može primetiti na slici 36, mašina sadrži 3 stanja. Takođe, postoje 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 intereaguje sadrži naziv događaja koji mikrokontroler registruje ukoliko je ta komponenta okinuta.

Uspostavljanjem napajanja sistema, podešavaju su početne vrednosti internih registra mikrokontrolera i automat se postavlja u stanje **ČEKANJE**. Taj režim namenjen je za podešavanje karakteristika 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 formata 8x21. Svakom elementu matrice dodeljen je jedan karater koji se može menjajti za vreme izvršavanja programa.



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 registra mikrokontrolera. Učestalost naponske impulsno širinske modulacije f_{PWM} prikazuje se u šestom redu displeja i određena je izrazom:

$$f_{PWM} = \frac{f_{clk}}{PRESKALER \cdot (1 + OCR1A)} \quad (82)$$

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

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

$$a = \frac{OCR1B}{OCR1A} \cdot 100 \quad (83)$$

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. Ta vrednost 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 od plavih 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 plavog tastera. 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 nalepticama 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.

redni broj	parametar	opseg vrednosti
1.	OCR1A	[6, 65535]
2.	OCR1B	[5, 65534]
3.	PRESKALER	{1, 8, 64, 256, 1024}
4.	IMPULSI	[0, 9999]

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 koji definišu vrednost delitelja učestalosti taktnog singla. 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 vrednosti faktora ispune na vrednost koja je jednaka ili veća od 100 %. Bez takvog ograničenja postojala bi mogućnost nepredviđenog ponašanja automata.

Posle podešavanja vrednosti 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 odbrojavanja moguće 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* mikrokontrolera 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.

Osim 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 stanjem mašine.

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

4.2.3. Korisnički interfejs

4.3. Interna upravljačka elektronika

4.4. Firmverski algoritam

5. Eksperimenti

Nastaviti pisanje ovde

6. Zaključak

7. Zahvalnica

8. Literatura

- [1] Klisurić O., Tanasijan L., *Terapija udarnim talasima*, 2011.
- [2] Belić S. Dragoljub, *FIZIKA I* (poglavlje 5), 1994.
- [3] Pavlović Š. Dragana, Mijić M., *ELEKTROAKUSTIKA* (poglavlje 2, poglavlje 4 i poglavlje 7), 2017.
- [4] Cvetić J., *Talasi* (poglavlje 7), 2003.
- [5] J. M. Camacho, V. Sosa, *Alternative method to calculate the magnetic field of permanent magnets with azimuthal symmetry*, 2013.
- [6] Obradović D., Šimonji J., *Kretanje tela na strmoj ravni*, 2008.
- [7] Friedrich U., Abtin J. Rad., *Ballistic Pain Therapy Devices: Measurement of Pressure Pulse Parameters*, 2012.
- [8] Nikolić B., *Fizička mehanika* (poglavlje 4), 2018.
- [9] Spasić D., *Mehanika* (poglavlje 3), 2015.
- [10] Lubrada V., *OTPORNOST MATERIJALA (UVOD U MEHANIKU DEFORMABILNOG TIJELA)* (poglavlje 3), 1989.
- [11] Miltenović V., *MAŠINSKI ELEMENTI oblici, proračun, primena* (poglavlje 2), 2009.
- [12] Al-Bender F., Colombo F., Reynaerts D, Villavicencio R., Waumans T., *Dynamic Characterization of Rubber O-Rings: Squeeze and Size Effects*, 2017.
- [13] https://web.archive.org/web/20190201171526/http://www.roymech.co.uk/Useful_Tables/Tribology/co_of_frict.htm#method (datum pristupa: 29.07.2020.)

- [14] Kauffman P., Vondracek M., *The Effect Surface Temperature Has on Kinetic Friction*, 2005.
- [15] <https://www.labthinkinternational.com/literatures/influence-of-temperature-on-coefficient-of-friction.html>
(datum pristupa: 30.07.2020.)
- [16] https://sites.ualberta.ca/~pogosyan/teaching/PHYS_130/FALL_2010/lectures/lect15/lecture15.html
(datum pristupa: 31.07.2020.)
- [17] <https://www.animations.physics.unsw.edu.au/jw/sound-wave-equation.htm>
(datum pristupa: 31.07.2020.)
- [18] Šešlija D., Čajetinac S., *PRILOG RAZVOJU MODELA ODLUČIVANJA ZA IZBOR ELEKTROPNEUMATSKOG UPRAVLJANJA* (poglavlje 2 i poglavlje 3), 2012.
- [19] Liu Y., Chen X., Guo A., Liu S., Hu G., *Quantitative Assessments of Mechanical Responses upon Radial Extracorporeal Shock Wave Therapy*, 2018.

9. Prilog

Prilog 1:

ommented [N18]: Dodati mašinske crteže pri konverziji u
if

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Master rad

Prilog 2:

Master rad

Master rad

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;
}
```

Master rad

```
/* 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);
```

Master rad

```
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)
```

Master rad

```
PORTD
PD0 - INPUT (BUTTON RESTART)
PD1 - OUTPUT (TXD PIN OF USART)
PD2 - INPUT (BUTTON START)
PD3 - INPUT (BUTTON PAUSE)
PD4 - OUTPUT (PWM SIGNAL FOR CONTROLLING THE ELECTROPNEUMATIC SOLENOID VALVE)
PD5 - OUTPUT (TOP VALUE FOR PWM 1)
PD6 - INPUT (BUTTON RESET)
PD7 - OUTPUT (PWM SIGNAL FOR CONTROLLING 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);
}
```

Master rad

```
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);
}
```

Master rad

```
/* 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);
}
```

Master rad

```
/* 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 TCCR0_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_milliseconds);

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 = TCCR0_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){
```

Master rad

```
        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;
    }
}
```

Master rad

```
/* 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_minimally_increase_prescaler(void);

void PWM_1_minimally_decrease_prescaler(void);

void PWM_1_set_prescaler(uint16_t prescaler);

uint16_t PWM_1_get_prescaler(void);

void PWM_1_minimally_increase_duty_cycle_percentage(void);

void PWM_1_minimally_decrease_duty_cycle_percentage(void);

float PWM_1_get_duty_cycle_percentage(void);
```

Master rad

```
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_OC1A(void);
uint16_t PWM_1_get_OC1B(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;

}

```

Master rad

```
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 prescaler
    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 prescaler
    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: {
```


Master rad

```
        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;
        }
    }
}
```

Master rad

```
        break;
    }

    case PRESCALER_IS_1024: {
        prescaler = 1024;
        break;
    }

    default: {
        prescaler = 0;
    }
}

return prescaler;
}

void PWM_1_minimally_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_minimally_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){
```

Master rad

```
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_minimally_increase_period(void){
    if(OCR1AH != 0xFF || OCR1AL != 0xFF){
        if(OCR1AH < 0xFF && OCR1AL == 0xFF ){
            OCR1AH++;
            OCR1AL = 0x00;
        }
        else{
            OCR1AH = OCR1AH;
            OCR1AL++;
        }
    }
}

void PWM_1_minimally_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_minimally_increase_frequency(void){
    PWM_1_minimally_decrease_period();
}

void PWM_1_minimally_decrease_frequency(void){
    PWM_1_minimally_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();
```

Master rad

```
uint32_t factor_1 = 1;
uint32_t denominator = (prescaler * (factor_1 + TOP));

return (CLOCK_FREQUENCY / denominator);
}

void PWM_1_minimally_increase_max_number_of_impulses(void){
    if(max_number_of_impulses < MAX_NUMBER_OF_IMPULSES){
        max_number_of_impulses++;
    }
}

void PWM_1_minimally_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;
}
```

Master rad

```
}

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
*/
```

Master rad

```
/* 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 percentage);

float PWM_2_get_duty_cycle_percentage(void);

void PWM_2_minimaly_increase_duty_cycle_percentage(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 percentage){
    OCR2 = (uint8_t)(round(percentage * 255.0f / 100.0f));
}

float PWM_2_get_duty_cycle_percentage(void){
    return ((float)OCR2 * 100.0f / 255.0f);
}

void PWM_2_minimally_increase_duty_cycle_percentage(void){
    if(OCR2 < 255)OCR2++;
}

void PWM_2_minimally_decrease_duty_cycle_percentage(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);
```

Master rad

```
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

```

Master rad

```
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_size = 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, CS2);
    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){
```

Master rad

```
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_LENGTH) > 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_LENGTH; trm++){

    GLCD_data_write(font[((ascii_code - 32) * FONT_LENGTH) + 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){

```

Master rad

```
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','\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);
}
```

Master rad

```
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(" ");
}
```

Master rad

```
/* 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);
```


Master rad

```
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;
}

```

Master rad

```
    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);
}
```

Master rad

```
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);  
}
```

Master rad

```
/* RGB_LED.h */

#ifndef RGB_LED_H_
#define RGB_LED_H_

#include <avr/io.h>
#include <avr/interrupt.h>

#define ws2812_resetime 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_resetime)
// #define ws2812_resetime 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);
```

Master rad

```
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_resetime);
}

// 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_resetime);
}

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];

```


Master rad

```
        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){
```

Master rad

```
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
```

Master rad

```
#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
        );
    }
}
```

Master rad

```
#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);

```

Master rad

```
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;

```

Master rad

```
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){
```


Master rad

```
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();
```

Master rad

```
    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_minimally_increase_period();

            }

            else{

                for(uint16_t number_of_decrements = 0; number_of_decrements < number;
                    number_of_decrements++)PWM_1_minimally_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_minimally_increase_duty_cycle_percentage();

            }

            else{


```

Master rad

```
        for(uint16_t number_of_decrements = 0; number_of_decrements < number;
            number_of_decrements++)PWM_1_minimally_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_minimally_increase_prescaler();

    else if(!buttons_and_switches_increment_sign_switch_is_pushed() && number == 1)
        PWM_1_minimally_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_minimally_increase_max_number_of_impulses();
    }

    else{

        for(uint16_t number_of_decrements = 0; number_of_decrements < number;
            number_of_decrements++)PWM_1_minimally_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());
}
```

Master rad

```
    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)TCR1B &= 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();
}
```

Master rad

```
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;
    }
}

////////////////////////////////////
```

Master rad

```
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;
        }
    }
}
```

Master rad

```
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)
*/
```