Tetris igra - ESP32 platforma s matričnim RGB LED pokaznikom

Pavić, Robert

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

TETRIS IGRA - ESP32 PLATFORMA S MATRIČNIM RGB LED POKAZNIKOM

Završni rad

Robert Pavić

Osijek, 2023.

SADRŽAJ

1.UVOD 2
1.1.Zadatak završnog rada 3
2. IZRADA TETRIS IGRE POMOĆU ESP32 PLATFORME S MATRIČNIM RGB LED POKAZNIKOM 4
2.1 Teorijski osvrt na projekt 4
2.2 Prijedlog sklopovskog rješenja 10
2.3 Prijedlog algoritamskog rješenja 11
3. REALIZACIJA TETRIS IGRE POMOĆU ESP32 PLATFORME S MATRIČNIM RGB LED POKAZNIKOM 13
3.1 Korišteni alati i razvojna okruženja 13
3.2 Realizacija sklopovskog rješenja 18
3.3 Realizacija algoritamskog rješenja 20
4.TESTIRANJE I REZULTATI 23
4.1 Metodologija testiranja 23
4.2 Rezultati testiranja 23
5. ZAKLJUČAK 31
LITERATURA 32
SAŽETAK 34
ABSTRACT 35
ŽIVOTOPIS 36
PRILOZI 37

1.UVOD

U današnjem svijetu definiranom tehnologijom, mikroupravljači su postali osnova brojnih tehnoloških primjena. Pojava mikroupravljača revolucionirala je svijet elektronike i računarstva te otvorila vrata novim tehnološkim rješenjima. Ugradnjom procesora, memorije i ulazno/izlazne periferije na istom čipu, omogućena je jednostavnost, učinkovitost i svestranost njihove primjene. Mikroupravljači se primjenjuju u mnogim područjima, od kojih su glavna automatizacija i robotika. Koriste se pri automatizaciju industrijskih postrojenja unutar raznih električnih mjernih sustava, kamera i senzorskih mreža. Primjenu su pronašli i u kućnoj automatizaciji pri izradi pametnih domova gdje mogu kontrolirati razne senzore, temperaturu, rasvjetu, sigurnosne sustave, itd. U polju robotike mikrokontroleri se često koriste za izradu strojeva, gdje omogućavanju upravljanje senzorima, motorima i ostalim dijelovima robota te obradu podataka vezanu uz otkrivanje objekata i kontrolu pokreta. Osim robotike i automatizacije primjenu su pronašli i u komercijalnim uređajima i Internetu stvari (engl. *Internet of Things, IoT*) te su time značajno utjecali na naš način komunikacije sa okruženjem te oblikovali budućnost tehnologije.

Još jedno od područja koje oblikuje budućnost tehnologije i potiče njezin napredak su računalne igre. U posljednjih nekoliko desetljeća računalne igre su ostvarile značaj porast u popularnosti, ali i u kompleksnosti. Rast u popularnosti računalnih igara također je uzrokovao rast tržišta računalnih igara koje postaje značajan ekonomski čimbenik u mnogim zemljama. Potražnja za boljim komponentama, računalima i konzolama te zahtjevima za boljim performansama igara glavni su faktori za napredak i daljni razvoj sklopovlja.

Cilj ovoga rada je povezivanje ta dva područja primjenom ESP32 mikrokontrolera u izradi Tetris računalne igre. Vizualni prikaz igre je ostvaren upotrebom dva matrična RGB LED pokaznika, dok se upravljačke naredbe primaju sa lokalnog upravljačkog uređaja koji se sastoji od gumb modula i analogne upravljačke palice (engl. *joystick)*. Zadaća ESP32 mikrokontrolera je primanje i obrada podataka od strane perifernih uređaja te daljnje upravljanje njima. ESP32 se programira uz pomoć integriranog programskog okruženja Arduino (engl. *Arduino Integrated Development Enviroment, Arduino IDE*) koje treba sadržavati potrebne dodatke koji omogućavanju rad sa ESP32 sustavima.

1.1.Zadatak završnog rada

Zadatak završnog rada je projektirati, izraditi i testirati računalnu igru Tetris na ESP32 mikroupravljačkoj platformi (u izvedbi WROOM) s prikazom na matričnom LED pokazniku u punom spektru boja, razlučivosti 64x128 piksela. Potrebno je omogućiti upravljanje predmetnom igrom koristeći lokalne upravljačke komande te prikaz iste.

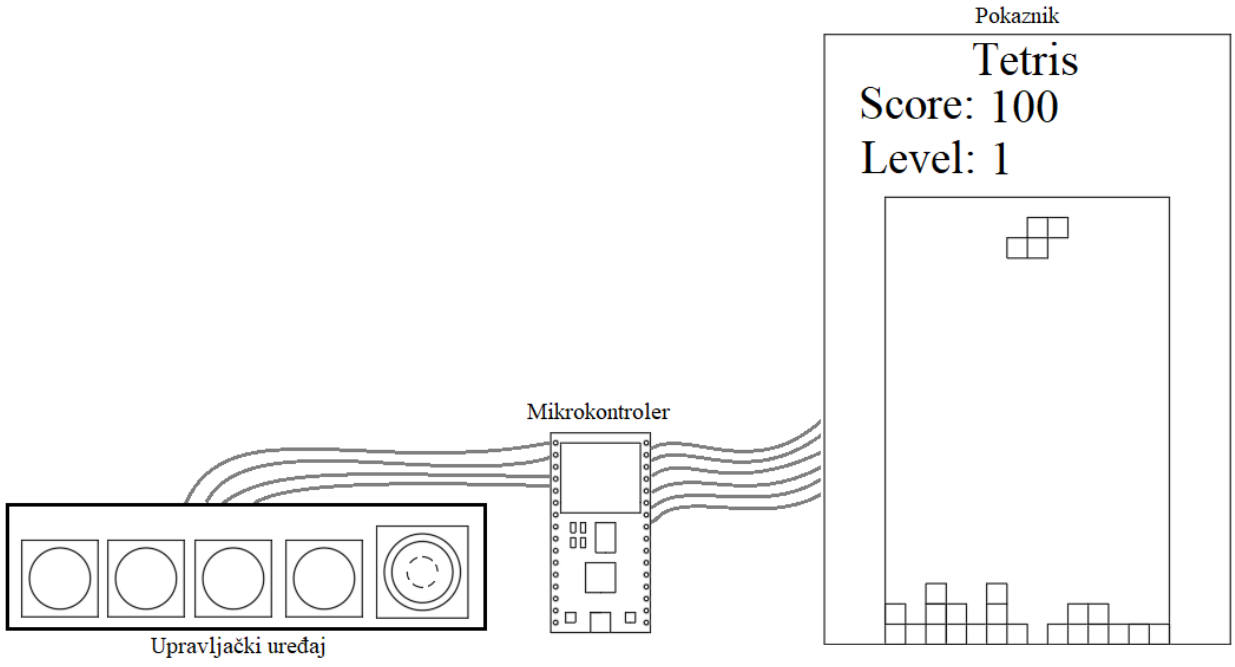
Na početku rada nalazi se kratak uvod u područja obuhvaćena ovim radom te zadatak završnog rada. U drugom poglavlju naveden je teorijski osvrt na projekt, te su obrađena i predložena programska i sklopovska rješenja. U trećem poglavlju ostvaren je detaljan pogled na realizaciju projekta, pojedine dijelove korištene u projektu, sklopovsko rješenje i programsko rješenje. Četvrto poglavlje opisuje testiranje i postignute rezultate testiranja. U petom poglavlju naveden je zaključak i kratak osvrt na rad i njegovu uspješnost.

2. IZRADA TETRIS IGRE POMOĆU ESP32 PLATFORME S MATRIČNIM RGB LED POKAZNIKOM

2.1 Teorijski osvrt na projekt

Ljudi svoje slobodno vrijeme provode na razne načine no sa napretkom igara i sklopovlja u posljednjh nekoliko godina sve više ljudi ga odlučuje provesti igrajući računalne igre. Poput napretka računalnih igara napredak su ostvarili i mikrokontroleri što je omogućilo njihovu inkorporaciju u mnogim novim sustavima. U ovome radu će se povezati računalne igre i mikrokontroleri kako bi se pomoću njih kreirao sustav koji implementira oboje.

Rad je konceptualno zamišljen tako da se sastoji od mikrokontrolera koji će biti mozak cijelog sustava te obavljati sve funkcije i operacije potrebne za uspješno provođenje toka igre i njen ispis na pokaznike. Na mikrokontroler će se slati upravljačke naredbe generirane na temelju pritiska gumba, tipkala ili upravljačke palice. Na temelju primljenih naredbi mikroupravljač mora provesti željene operacije unutar igre. Cijeli tok igre mora biti jasno prikazan na nekoj vrsti pokaznika. Na slici 2.1 prikazana je skica sustava.



Slika 2.1 Prikaz skice sustava

Kako bi projekt bio uspješno izrađen u obzir se mora uzeti nekoliko stvari. Kao prvo u projektu je potrebno koristiti pokaznike dovoljno velike rezolucije kako bi bio omogućen jasan prikaz igre, te prikaz igre mora biti u punom spektru boja. Potrebno je koristiti naprednije verzije ESP32

mikrokontrolera koje nude bolje karakteristike i funkcionalnosti što će omogućiti realizaciju sustava. Također će biti potrebno proučiti i implementirati modulacije za kontrolu razine osvjetljenosti kako bi se dodatno smanjio teret postavljen na mikrokontroler.

2.1.1 Teorija prikaza igre

Prikaz igre i podataka u igri mora biti izravan i jasan kako bi korisnik imao što bolje iskustvo. Glavni cilj prikaza podataka je omogućavanje uvida u trenutno stanje igre kako bi se korisniku omogućilo donošenje ispravnih odluka. Na pokazniku se moraju prikazati svi podatci koji utječu na igru poput pozicija, broja bodova, razine na kojoj se nalazi, itd., [1].

Kako bi prikaz igre bio moguć potrebno je odabrati odgovarajući pokaznik na koji će se igra ispisivati. Za vizualizaciju podataka pri radu sa mikrokontrolerima dostupno je nekoliko vrsta pokaznika koji su široko dostupni. Glavne tri vrste koje se proizvode za uporabu sa mikrokontrolerima su:

- Zasloni temeljeni na tehnologiji tekućih kristala (engl. Liquid Crystal Display, LCD)
- Zasloni temeljeni na tehnologiji organskih svjetlećih dioda (engl. Organic Light-Emitting Diode, OLED)
- RGB LED matrični pokaznici

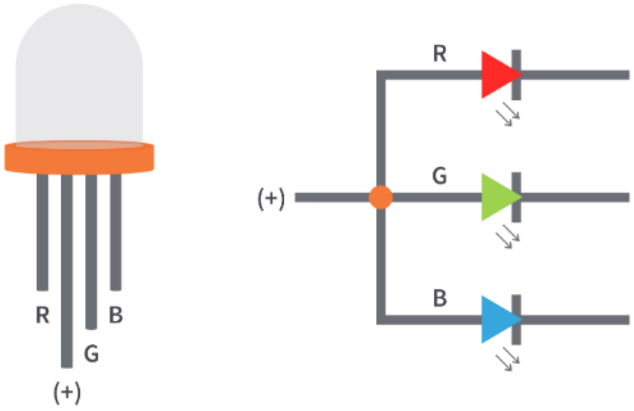
Prema [2] LCD zaslon je vrsta zaslona koja u svome primarnom obliku rada koristi tehnologiju tekućih kristala. LCD zasloni su mnogo tanji od prijašnih verzija zaslona i za rad koriste puno manje energije. Rade na principu blokiranja svjetlosti, a ne emisije svjetlosti gdje za prikaz slike koriste pozadinsko osvjetljenje. U posljednjih nekoliko godina sve su ih više počele zaminjenjivati novije tehnologije poput OLED-a.

Prema [3] OLED zasloni su napravljeni od serije tankih organskih filmova između dva vodiča. Prilikom protjecanja struje kroz vodiče dolazi do emisije svjetlosti. OLED zasloni su puno tanji i efikasniji od LCD zaslona i za rad im nije potrebno pozadinsko osvjetljenje.

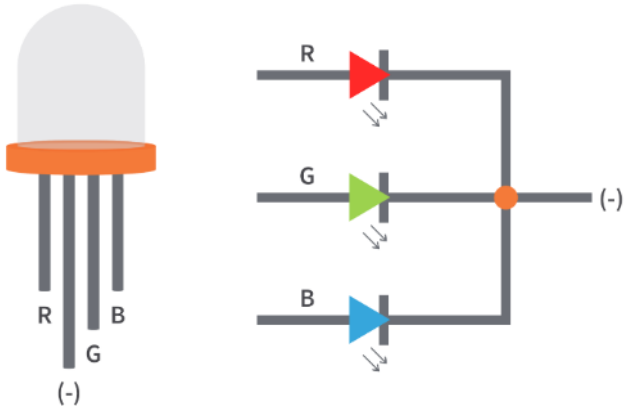
RGB LED matrični pokaznici se sastoje od RGB LED dioda koje su povezane u redove i stupce te na taj način kontrolirane. Karakteristično su puno većih dimenzija i koriste se za izradu velikih zaslona. U ovome radu će se koristiti RGB LED zasloni kako bi se dobio velik i jasan prikaz igre.

2.1.2 Prikaz punog spektra boja

Prikazivanje punog spektra boja na matričnom pokazniku postignuto je upotrebom RGB LED dioda. Prema [4] RGB LED dioda je paket koji se sastoji od tri LED diode: crvene, zelene i plave. Intenzitet svake od tri LED diode kontrolira se zasebno, ali pri gledanju ljudsko oko ne raspoznaje svaku od tih boja već vidi njihovu kombinaciju. Postoje dvije glavne izvedbe RGB LED dioda: RGB LED sa zajedničkom anodom (Slika 2.2) i RGB LED sa zajedničkom katodom(Slika 2.3).

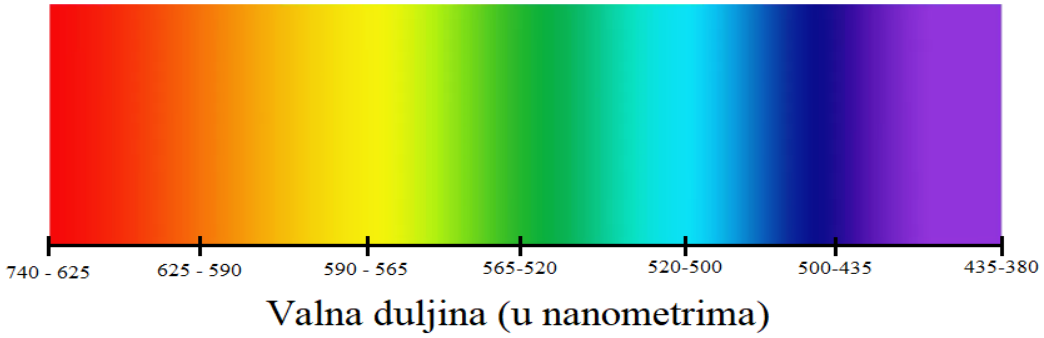


Slika 2.2 Izgled RGB LED diode sa zajedničkom anodom prema izvoru [4].



Slika 2.3 Izgled RGB LED diode sa zajedničkom katodom prema izvoru [4].

RBG LED diode napravljene su od različitih poluvodičkih materijala koji emitiraju fotone različite valne duljine. Ti fotoni različite valne duljine predstavljaju razne boje. Ljudskom oku vidljiv je spektar valne duljine od 400 nanometara do 700 nanometara,[5]. Na slici 2.4 prikazan je spektar boja vidljiv ljudskom oku.



Slika 2.4 Spektar boja vidljiv ljudskom oku prema izvoru [6].

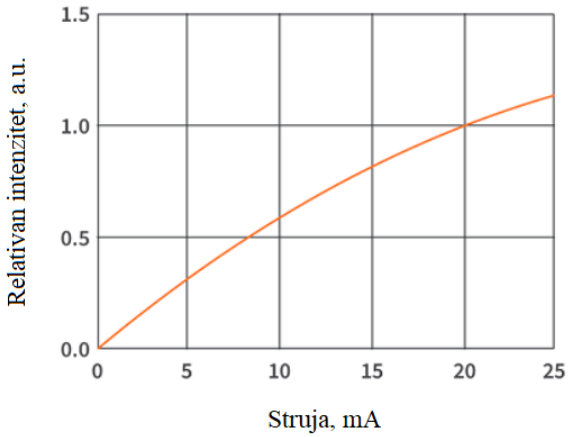
Povezivanjem velikog broja RGB LED dioda u redove i stupce može se izraditi RGB LED matrični pokaznik.

2.1.3 Metode kontrole razine osvjetljenosti

Najčešće se koriste PWM modulacija(engl. *Pulse-width modulation*) i BCM modulacija(engl. *Binary code modulation*). Postoji nekoliko glavnih metoda koje se primjenjuju za kontrolu razine osvjetljenosti RGB LED dioda i matričnih pokaznika, a to su:

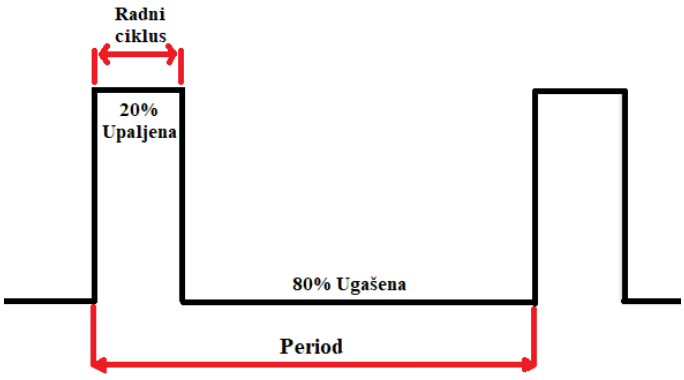
- Redukcija konstantne struje (engl. Constant Current Reduction, CCR)
- PWM modulacija (engl. *Pulse-width modulation*)
- BCM modulacija (engl. *Binary code modulation*)

Prema [7] prilikom upotrebe CCR metode kroz LED diodu kontinuirano teče struja za razliku od ostalih metoda gdje se dioda uključuje i isključuje u određenim vremenskim periodima. Razina osvjetljenosti LED diode određena količinom struje koja kroz nju teče. Slika 2.5 prikazuje graf ovisnosti razine osvjetljenosti o količini struje koja teče kroz LED diodu. Ova metoda je prigodna za korištenje u sustavima gdje je potrebno izbjeći stvaranje elektromagnetskih smetnji, ali je nepogodna za slučaje kada razina osvjetljenosti treba biti manja od 10% maksimalne vrijednosti te prikazane boje mogu biti nekonzistentne.



Slika 2.5 Graf ovisnosti relativnog intenziteta svjetlosti o količini struje koja protječe kroz LED dioda prema izvoru [7]

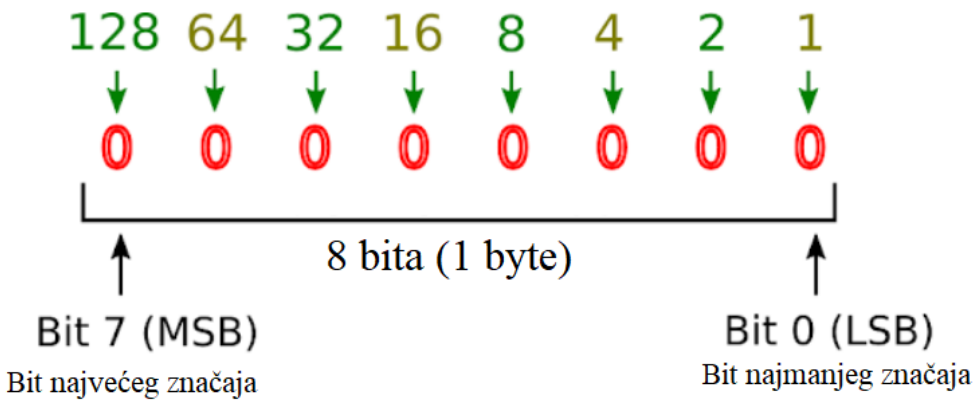
Prema [8] PWM modulacija je modulacija u kojoj je jačina osvjetljena određena radnim ciklusom koji predstavlja dio perioda za koji na diodu dovodimo napon. Odnosno LED dioda je dio perioda uključena, a dio perioda isključena. Uključivanje i isključivanje je potrebno izvoditi dovoljno velikom brzinom kako ljudsko oko nebi vidjelo trepenjenje već kontinuiranu emisiju svjetlosti željenog intenziteta. Ukoliko je potrebno postaviti ledicu da svijetli sa 20% svoje maksimalne jačine tada trajanje dovođenja struje u odnosu na cijeli period treba biti 20% kao što je vidljivo na slici 2.6. Na taj način se mijenja srednja vrijednost signala.



Slika 2.6 Primjer PWM modulacije prema izvoru [8].

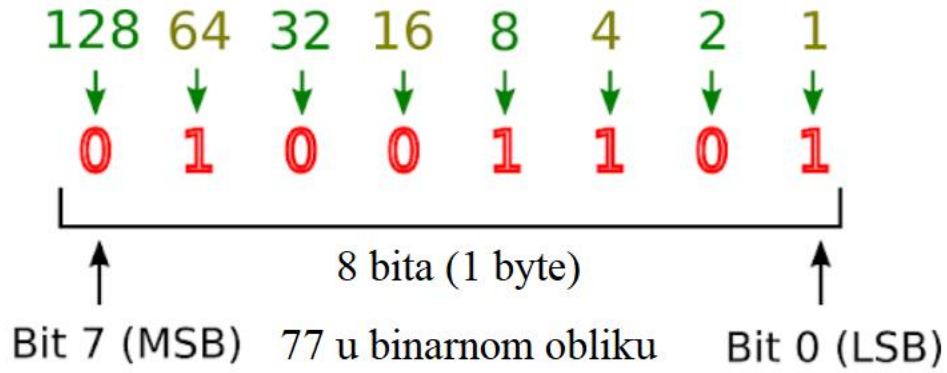
Prema [8] BCM modulacija za određivanje osvjetljenosti ledica koristi značaj bitova unutar binarnog broja. Ako se broj sastoji od naprimjer 8 bita tada svakom bitu možemo dodijeliti broj od 0 do 7 koji će predstavljati njegov značaj. Bit označen sa brojem 7 ima najveći značaj te traje

najveći dio perioda dok bit sa najmanjim značajem označen sa brojem 0 traje najkraći dio perioda (slika 2.7). Svaki bit ovisno o značaju nosi i određenu težinu pa prema tome bit 0 ima težinu $2^0 = 1$, bit 1 težinu $2^1 = 2$, bit 2 težinu $2^2 = 4$, itd.



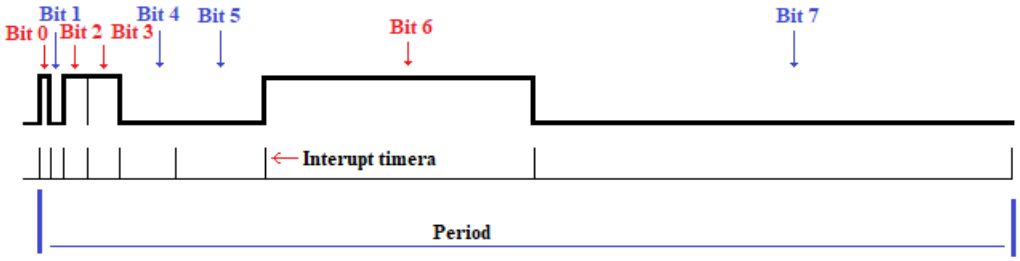
Slika 2.7. Prikaz značaja bita unutar 8 bitnog broja prema izvoru [8].

Pomoću 8-bitne BCM rezolucije moguće je prikazati $2^8 = 256$ različitih razina osvjetljenosti sa vrijednostima od 0 do 255 gdje nula predstavlja najmanju razinu osvjetljenosti tj. 0% radnog ciklusa, a 255 najveću razinu osvjetljenosti tj. 100% radnog ciklusa. Kako bi se dobila BCM vrijednost potrebno je pomnožiti BCM rezoluciju, u ovom slučaju 256 sa postotkom radnog ciklusa. Naprimjer BCM vrijednost za razinu osvjetljenosti od 30% će iznositi 256 * 0.3 = 77. Broj 77 je prikazan na slici 2.8 i predstavlja 30% radnog ciklusa.



Slika 2.8. Prikaz broja 77 u binarnom obliku što predstavlja 30% radnog ciklusa prema izvoru [8].

Jedan ciklus perioda se sastoji od 256 dijelova. Svaki bit binarnog broja traje onoliko dijelova ciklusa kolika je njegova težina. Pa tako bit 0 traje 1 dio ciklusa, bit 1 traje 2 dijela ciklusa, bit 2 traje 4 dijela ciklusa, dok bit 7 traje 128 dijelova ciklusa. Kako bi razina osvjetljenosti iznosila 30% potrebno je bitove 0, 2, 3 i 6 postaviti u stanje 1 kao što je prikazano na slici 2.9 što kada zbrojimo njihove težine iznosi 77 odnosno 30% radnog ciklusa. Na slici 2.9 se vidi kako dio perioda predstavljen bitom raste sa značajem toga bita pa tako bit 7 ima najveće trajanje a bit 0 najkraće trajanje.



Slika 2.9. Primjer BCM modulacije za postizanje razine osvjetljenosti 30% prema izvoru [8].

U radu je implementirana BCM modulacija jer je ona puno ekonomičnija u smislu crpljenja procesorskih resursa mikrokontrolera.

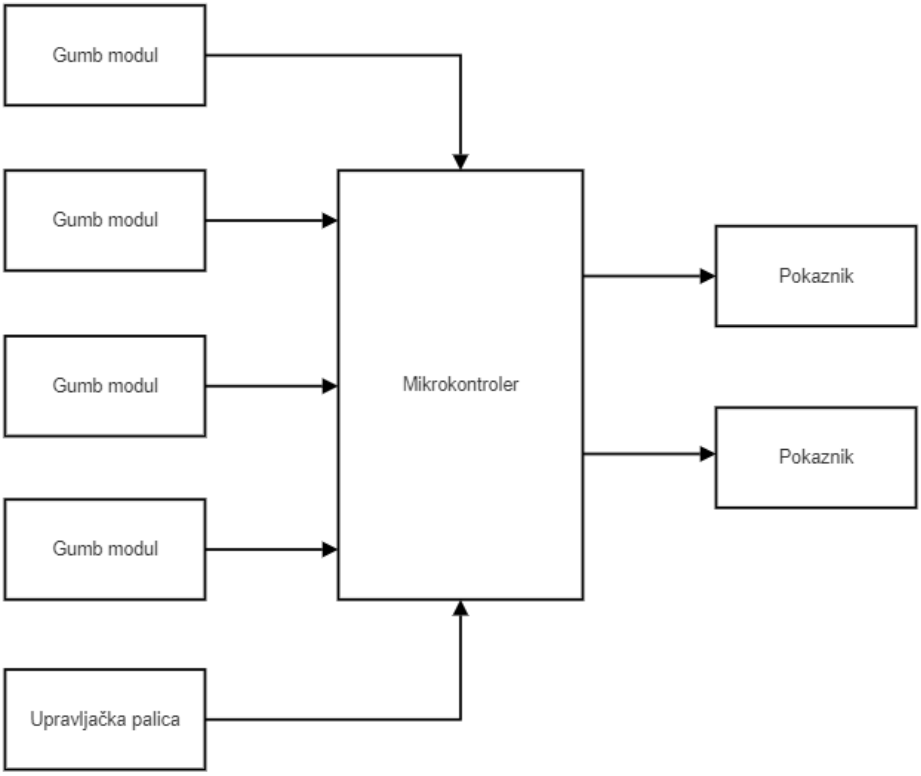
2.1.4 Dizajn kontrole igre

Prema [9] kontrole unutar igre trebaju biti jasne i jednostavne za korištenje i svladavanje. Moraju komunicirati korisnikovu namjeru na način koji je predvidljiv i intuitivan te korisniku pružiti punu kontrolu nad igrom. Prilikom dizajna upravljačkog uređaja u obzir je potrebno uzeti fizikalne i kognitivne limitacije korisnika. Prema tome korisnika se nesmije preplaviti potrebom za unosom velikog broja upravljačkih naredbi te je potrebno gumbove i ostale dijelove uređaja postaviti na prirodne pozicije koje su u dosegu prstiju korisnika.

2.2 Prijedlog sklopovskog rješenja

Na slici 2.10 vidi se da se sustav sastoji od logumb modula, upravljačke palice, mikrokontrolera i pokaznika. Zadatak gumb modula i upravljačke palice je slanje upravljačkih naredbi na mikrokontroler. Mikrokontroler zatim mora primiti upravljačke naredbe sa upravljačke palice i

gumb modula te na temelju njih izvršiti odgovarajuće funkcije i procese u sklopu igre. Mikrokontroler mora sadržavati sve funkcije potrebne za realizaciju igre. Nakon procesiranja mikrokontroler vrši komunikaciju upravljačkih naredbi na pokaznik koji ih mora primiti i na temelju njih prikazati igru. Pokaznik također mora biti zadovoljavajuće rezolucije kako bi prikaz igre bio jasan.

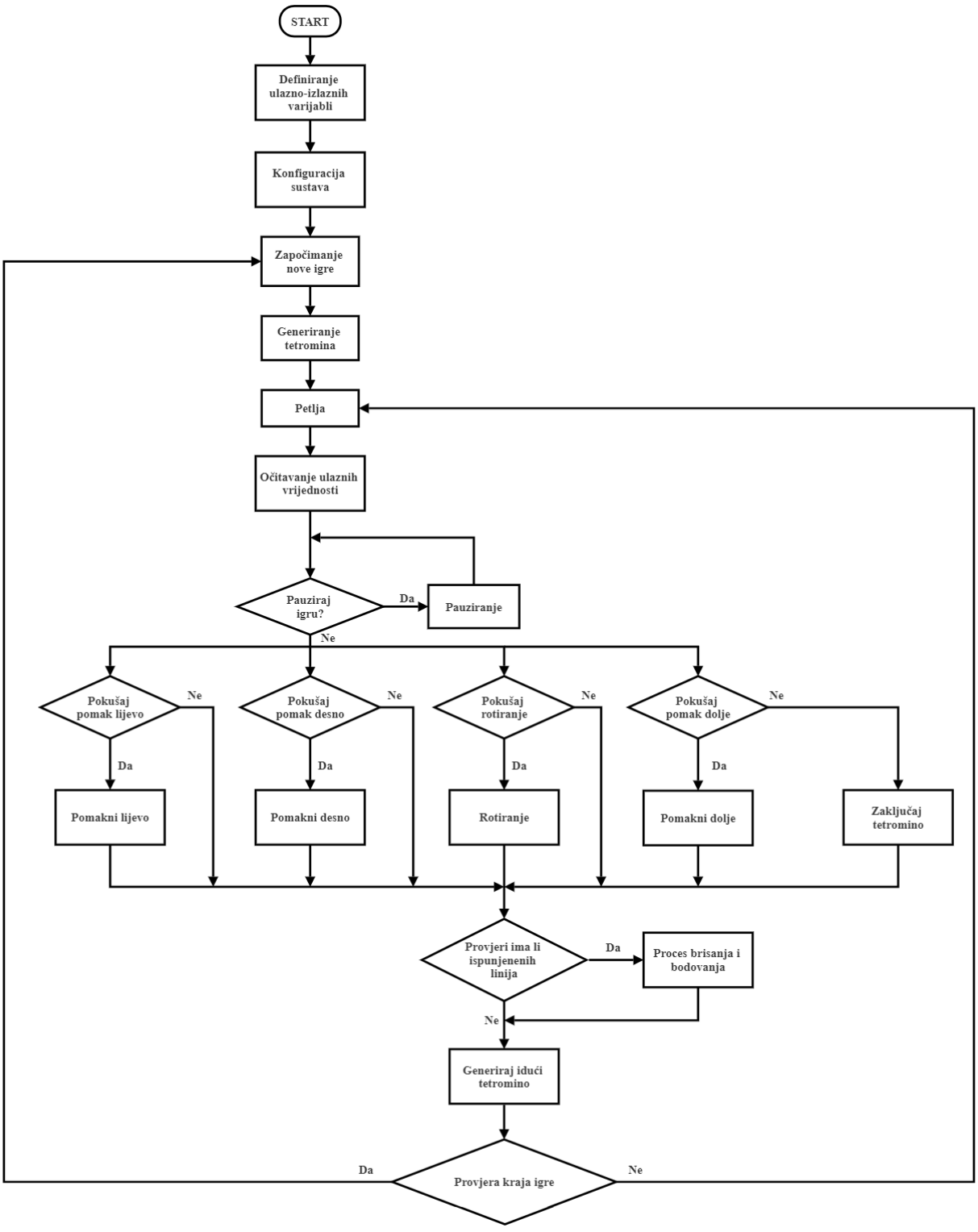


Slika 2.10 Blok dijagram strukture.

2.3 Prijedlog algoritamskog rješenja

Mikrokontroler predstavlja mozak cijelog sustava i njegova zadaća je procesiranje svih funkcija i podataka potrebnih za uspješan rad igre. Slika 2.11 prikazuje kako je zamišljeno algoritamsko rješenje tetris igre. Za početak potrebno je pravilno definirati ulazno-izlazne varijable te izvesti pravilnu konfiguraciju sustava. Nakon što počne nova igra na pinovima se očitavaju ulazne vrijednosti sa lokalnog upravljačkog uređaja. Ovisno o očitanim vrijednostima izvode se tražene operacije pomicanja, rotacije ili pomaka. U trenutku kada tetromino ostvari kontakt sa dnom igrače ploče ili prethodno postavljenim tetrominom on se zaključava. Nakon zakjučavanja tetromina

provodi se provjera redova te brisanje i dodjela bodova. Nakon što se generira sljedeći tetromino vrši se provjera kraja igre. Ako je mjesto na vrhu ploče gdje je idući tetromino potrebno generirati već zauzeto dolazi do kraja igre. U suprotnom se ponovno pokreće petlja te se ciklus očitavanja ulaznih vrijednosti i pomicanja tetromina ponovno pokreće.



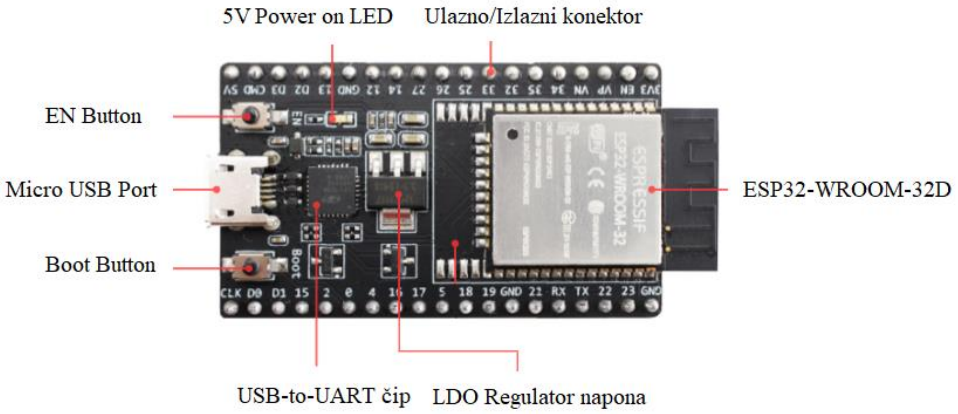
Slika 2.11 Blok dijagram algoritamskog rješenja.

3. REALIZACIJA TETRIS IGRE POMOĆU ESP32 PLATFORME S MATRIČNIM RGB LED POKAZNIKOM

3.1 Korišteni alati i razvojna okruženja

3.1.1 ESP32-DevKitC V4

ESP32-DevKitC V4 je razvojna pločica temeljena na ESP32 mikrokontroleru. Za potrebe ovoga rada koristiti će se verzija sa ESP32-WROOM-32D mikrokontrolerom. ESP32-WROOM-32D je moćan mikrokontroler koji u sebi sadrži ESP-D0WD čip sa dvije procesorske jezgre velike brzine, te se sa njega izvodi 39 pinova, [10]. Na slici 3.1 prikazana je pločica sa mikrokontrolerom i perifernim dijelovima. Tehničke karakteristike ESP32-WROOM-32D vidljive su u prilogu: **P.1.** Funkcionalnosti pojedinih pinova u prilogu: **P.2.**



Slika 3.1. ESP32-DevKitC V4 sa ESP32-WROOM-32D mikrokontrolerom prema izvoru [11].

3.1.4 Arduino integrirano razvojno okruženje

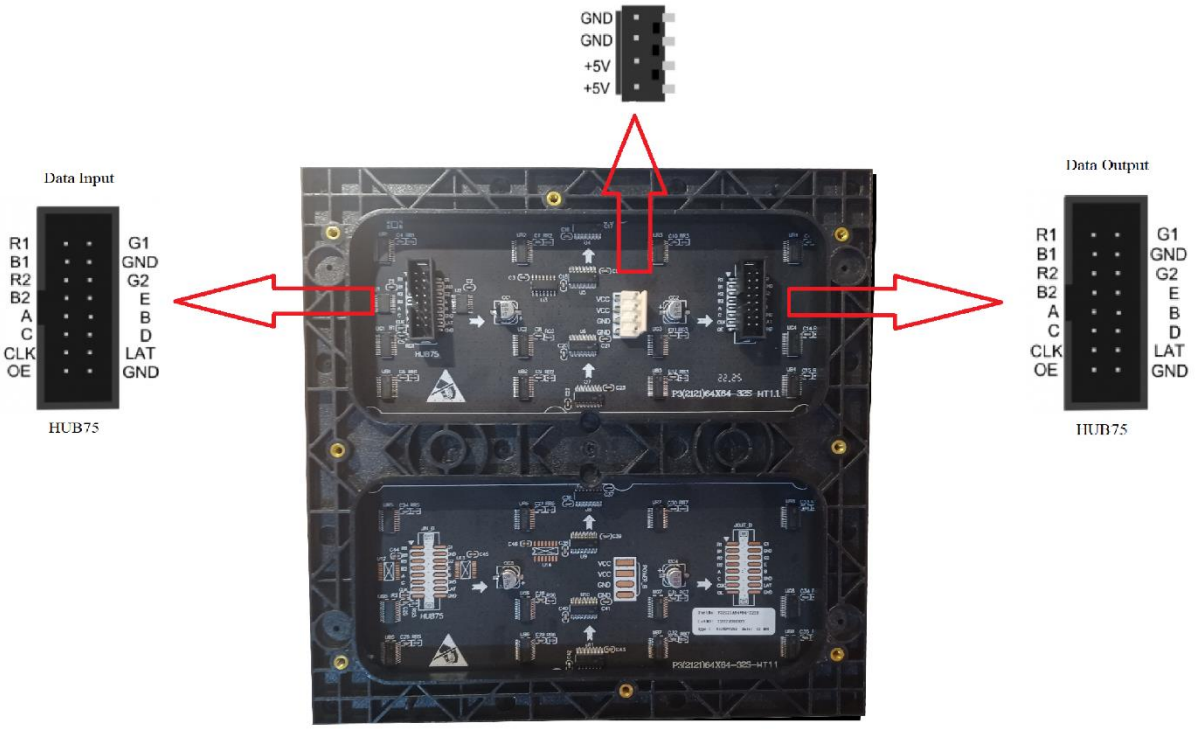
Arduino IDE (engl. *Integrated Development Enviroment*) je integrirano razvojno sučelje namijenjeno za pisanje koda i njegovo slanje na pločice. Za programiranje u Arduino razvojnom okruženju koristi se varijanta C++ programskog jezika proširena sa dodatnim metodama i funkcijama. Arduino razvojno sučelje samo po sebi nudi podršku samo za arduino pločice pa je potrebno instalirati dodatak za podršku ESP32 pločica. Za to je potrebno pomoću upravitelja pločica instalirati dodatak esp32 od strane *Espressif Systems*-a [12].

3.1.2 Matrični RGB LED pokaznik

Za vizualni prikaz igre koristi se matrični RGB LED pokaznik rezolucije 64x128 piksela koji je ostvaren povezivanjem dva manja pokaznika rezolucije 64x64 piksela. Svaki od matričnih pokaznika na sebi sadrži 4096 LED dioda. Pokaznik ima brzinu skeniranja 1/32 što znači da tokom svakog ciklusa kontrolira 128 LED dioda što na pokazniku predstavlja 2 reda, [13]. Uključenje jednog reda se vrši od vrha pokaznika dok se uključenje drugog reda vrši od sredine pokaznika kao što je prikazano prema slici 3.2. Tehničke karakteristike matričnih pokaznika navedene su u prilogu: **P.3.**

Slika 3.2. Prikaz načina skeniranja ledica na RGB LED matričnom pokazniku brzine skeniranja 1/32.

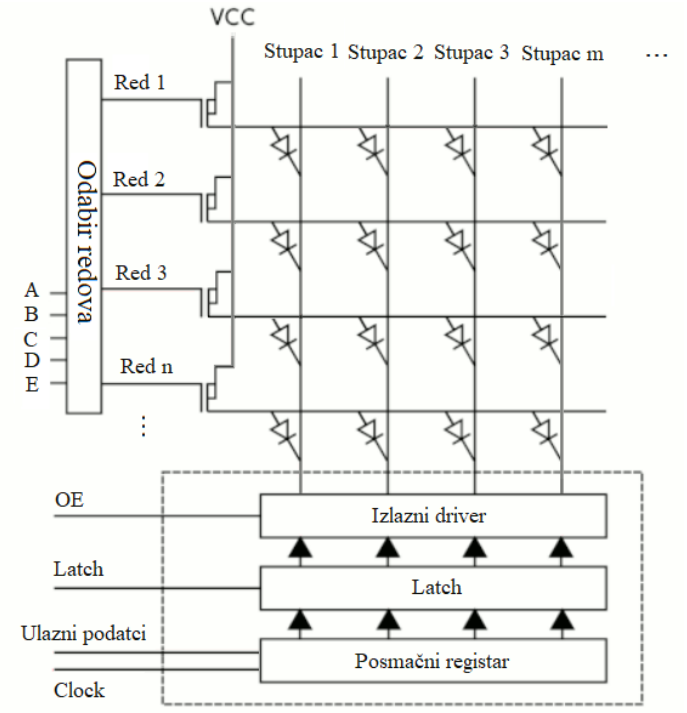
HUB75 jedan je od najčešće korištenih standarda za kontrolu matričnih pokaznika. Na pokazniku se nalaze dva HUB75 priključka kao što je prikazano prema slici 3.3. Jedan priključak je ulazni i pomoću njega se primaju signali za upravljanje matričnim pokaznikom dok je drugi izlazni i on se koristi za spajanje sa idućim pokaznikom. Priključak se sastoji od 16 pinova gdje se pinovi A, B, C, D i E koriste pri odabiru redova, a pinovi R1, B1, G1, R2, B2 i G2 za kontrolu boja. CLK pin se koristiti pri unosu podataka o bojama, a LAT i OE pin pri ispisu vrijednosti na LED diode. Na pozadini se još nalazi i priključak za napajanje sa 5V kao i priključci za uzemljenje.



Slika 3.3. Prikaz pozadinske strane RGB LED matričnog pokaznika i njegovih priključaka.

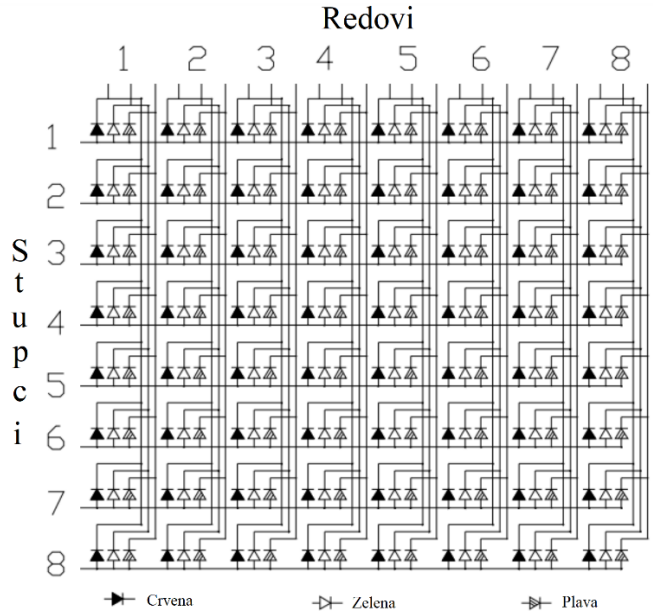
Na slici 3.4 prikazana je pojednostavljena struktura RGB LED matričnog pokaznika. Za početak može se vidjeti da su LED diode povezane u redove i stupce te se kontroliraju na taj način jer bi direktna kontrola svake od LED diode na matričnim pokaznicima velike rezolucije bila nerealna. Odabir redova je realiziran pomoću pinova A, B, C, D i E. Ovisno o kombinaciji vrijednosti dovedenih na te pinove ostvaruje se odabir različitih redova. Kako bi kontrolirali prikaz boja unutar ta dva reda tj. 128 RGB LED dioda gdje se svaka sastoji od tri boje, unutar pokaznika se nalaze 24 16-bitna posmačna registra što omogućuje unos 384 bit-a od čega je 8 registara odnosno 128 bit-a rezervirano za svaku boju. Pinovi R1 i R2 su povezani na posmačne registre određene za crvenu boju, pinovi G1 i G2 na registre za zelenu boju, a B1 i B2 na registre za plavu boju. Promjenom stanja na clock pinu u posmačne registre se upisuju ulazni podatci tj. bitovi koji predstavljaju boje. Nakon što su u posmačne registre upisani bitovi, promjenom stanja na latch pinu se trenutno stanje unutar posmačnih registara kopira u latch sklop, a zatim promjenom stanja na OE pinu ispisuje na LED diode. Latch sklop služi kao „memorija“ te se u njega sprema stanje koje se u određenom trenutku nalazi u posmačnim registrima jer na LED diode ne želimo ispisivati vrijednosti posmačnih registara prilikom unosa novih vrijednosti namjenjenih za idući red dioda već samo u trenutku kada su vrijednosti za određeni red dioda u potpunosti postavljene. Postupak odabira redova, upisa bitova u posmačne registre te uključivanja i isključivanja dioda se odvija

korak po korak no zbog jako velike brzine izvođenja ljudskom oku izgleda kao da se sve LED diode kontroliraju odjednom.

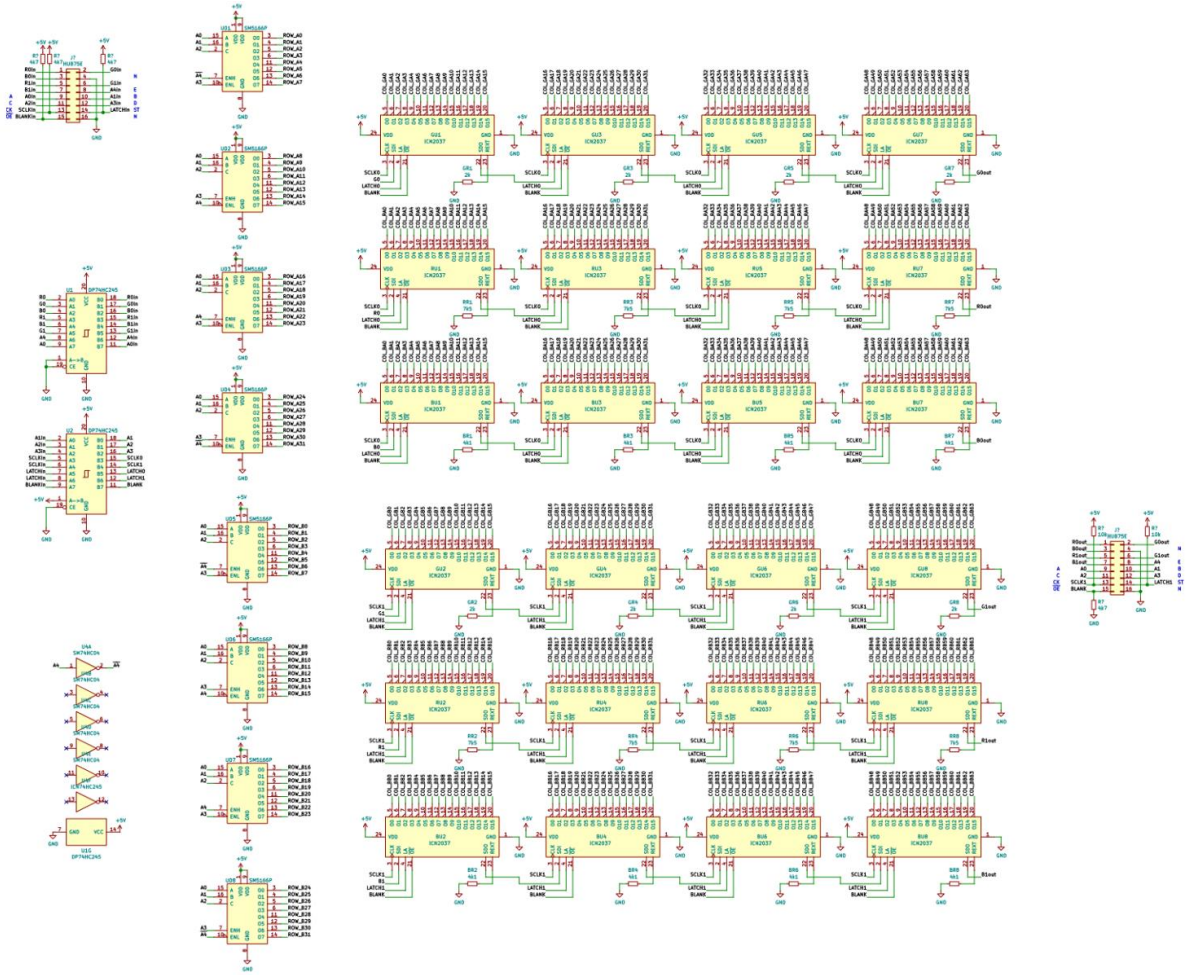


Slika 3.4. Prikaz strukture RGB LED matričnog pokaznika prema izvoru [13].

Slika 3.5 prikazuje detaljniji prikaz povezivanja RGB LED dioda unutar matričnog pokaznika rezolucije 8x8 no isti način povezivanja se primjenjuje i za pokaznike veće rezolucije.



Slika 3.5. Detaljan prikaz povezivanja LED dioda unutar RGB matričnog pokaznika prema [14].



Slika 3.6. Shema RGB LED matričnog pokaznika prema izvoru [15]

Slika 3.6 prikazuje detaljnu shemu RGB LED matričnog pokaznika. Iz nje se vidi kako se pokaznik sastoji od 24 16-bitna posmačna registra te 8 demultipleksera gdje svaki ima 8 izlaza što ukupno omogućuje odabir 64 reda. Na HUB75 priključak su također direktno spojeni čipovi koji služe kao međuspremnici.

Za uspješno upravljanje korištenim matričnim LED pokaznicima potrebno je instalirati biblioteku ESP32 HUB75 LED MATRIX PANEL DMA [16]. Biblioteka za određivanje razine osvjetljenosti pokaznika koristi BCM(engl. *Binary-code modulation*) modulaciju. Biblioteka u sebi također koristi elemente Adafruit GFX[17] biblioteke koja uglavnom sadrži funkcije za ispis piksela, linija, teksta i raznih oblika .

3.1.3 Kontrolne komponente

Upravljanje unutar igre ostvareno je upotrebom analogne upravljačke palice i gumb modula. Promjenom položaja palice ili pritiskom gumba na mikrokontroler se šalje upravljačka naredba koju mikrokontroler očitava i na temelju nje provodi željene operacije unutar igre.

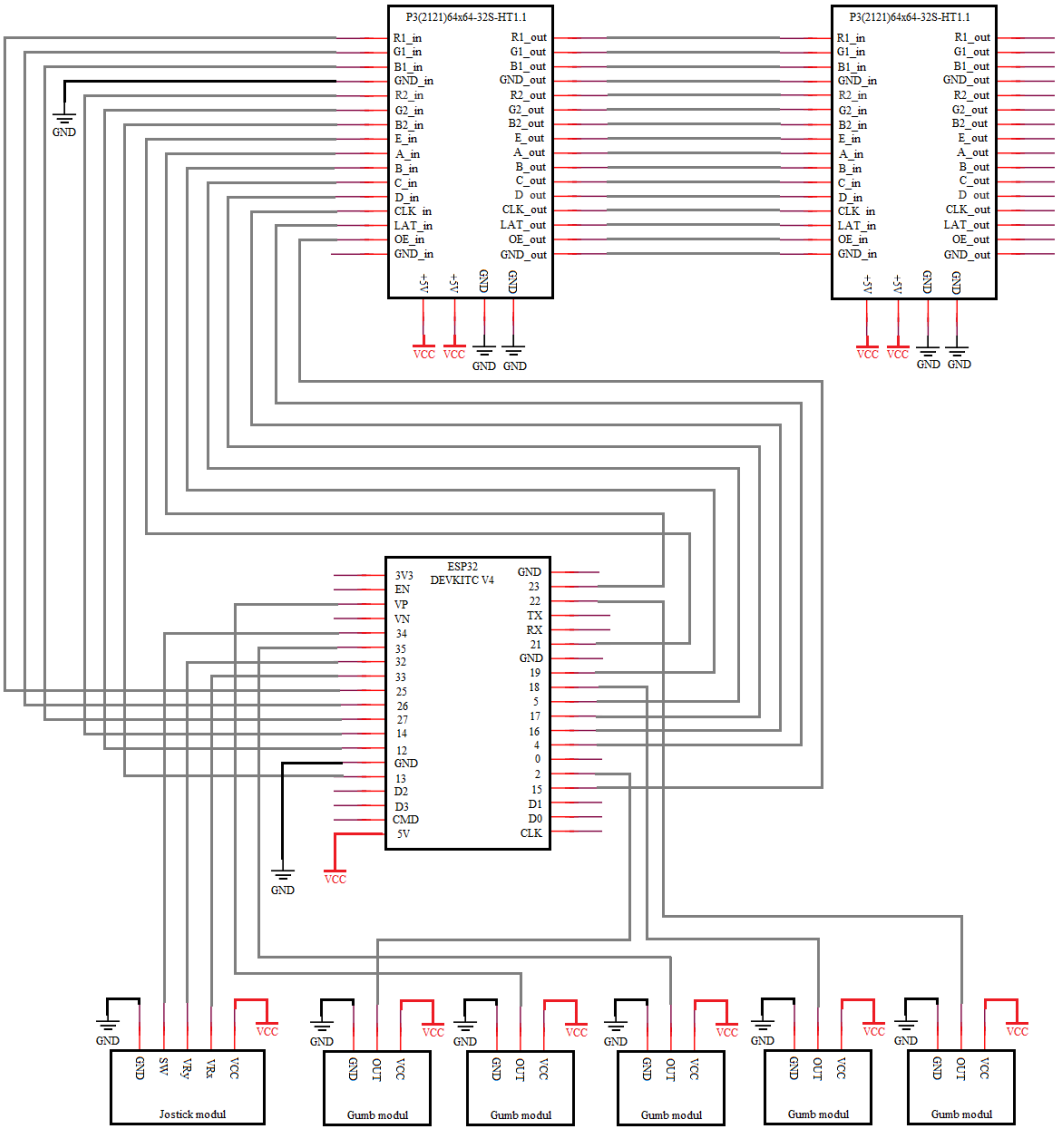
Slika 3.7 Analogna upravljačka palica i gumb modul.

3.2 Realizacija sklopovskog rješenja

Odabir sklopovlja ovisi o mnogim faktorima. Temeljem istraživanja i na temelju prijedloga mentora odabrane su komponente koje ispunjavaju zahtjeve određene sustavom. Matrični pokaznici uglavnom rade na isti način tako da su razlike između proizvođača minimalne, iz toga razloga pokaznici su odabrani na temelju dostupnosti i cijene. S obzirom da su odabrana dva matrična pokaznika rezolucije 64x64 piksela biti će potrebno kontrolirati ukupno 8192 LED dioda gdje se svaka sastoji od tri boje. Može se doći do zaključka da će za uspješnu realizaciju projekta biti potreban mikrokontroler velike brzine procesora. Također za unos upravljačkih naredbi odabrano je pet gumb modula i jedna upravljačka palica što zajedno sa matričnim pokaznicima čini veliki broj potrebnih pinova. Na temelju tih zahtjeva i istraživanja odabrana je pločica ESP32 DevkitC V4 sa ESP32-WROOM-32D mikrokontrolerom. Pločica ima 39 ulazno-izlaznih pinova, a ESP32-WROOM-32D sadrži dvojezgreni procesor sa taktnom frekvencijom do 240MHz.

ESP32 mikrokontroleri obično rade na naponu 3.3V, no pločica na sebi sadrži regulator napona što omogućava spajanje na izvor napajanja od 5V. Analizom ostalih komponenti utvrđeno je da i one rade na naponu od 5V te da svaki od matričnih pokaznika u trenutcima može ostvariti potrošnju struje do 4A, dok je potrošnja ostalih komponenti znatnije manja. Kako bi osigurali da

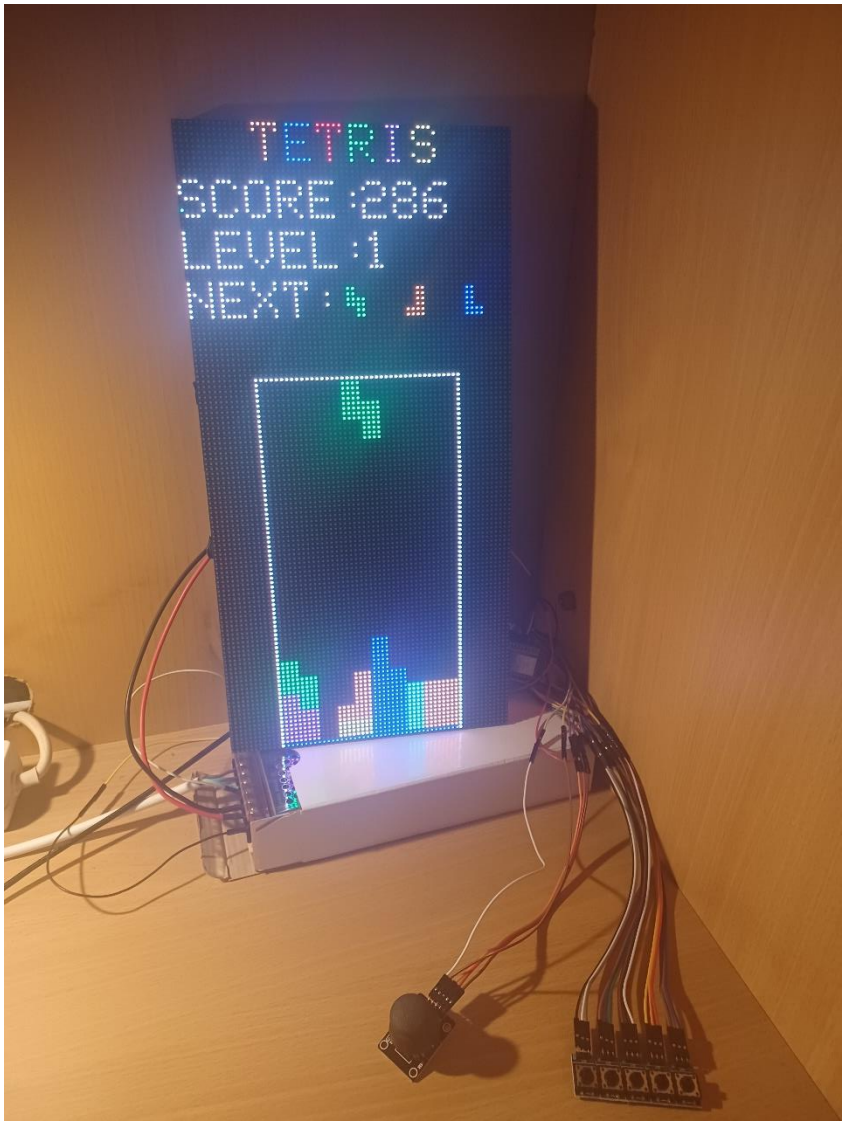
će sustav biti opskrbljen dovoljnom količinom snage odabran je izvor napajanja 5V i 20A. Pločica je na izvor napajanja povezana putem pina 5V. Pločica se može napajati i putem USB priključka ili dovođenjem napona od 3.3V na pin 3V3 no u određenom trenutku na pločicu smije biti spojen samo jedan način napajanja jer u suprotnom može doći do oštećenja sustava.



Slika 3.8 Električna shema sustava

Na slici 3.8 prikazana je električna shema sustava. Iz sheme se vidi da se OUT pinovi gumb modula povezuju na pinove 2, 18, 22, 35 i VP, a pinovi SW, VRx i VRy upravljačke palice na pinove 33, 32 i 34. Ti pinovi će se koristiti za očitanje vrijednosti upravljačkih komponenti tj. za upravljanje igrom. Iz sheme se također vidi da je izlazni priključak prvog matričnog pokaznika direktno povezan sa ulaznim priključkom drugog matričnog pokaznika. Povezivanjem pokaznika na taj način ostvaren je jedan pokaznik dvostruke širine. Pinovi 4, 5, 12, 13, 14, 15, 16, 17, 19, 21, 23, 25, 26 i 27 su povezani na ulazni priključak prvog matričnog pokaznika. Pomoću njih

mikrokontroler će na matrične pokaznike slati upravljačke signale pomoću kojih će se na pokaznicima ispisivati igra.Detaljan prikaz povezivanja pinova dostupan je i u tabličnom obliku u prilogu: **P.4**. Na slici 3.9 može se vidjeti izgled spojenog sustava.

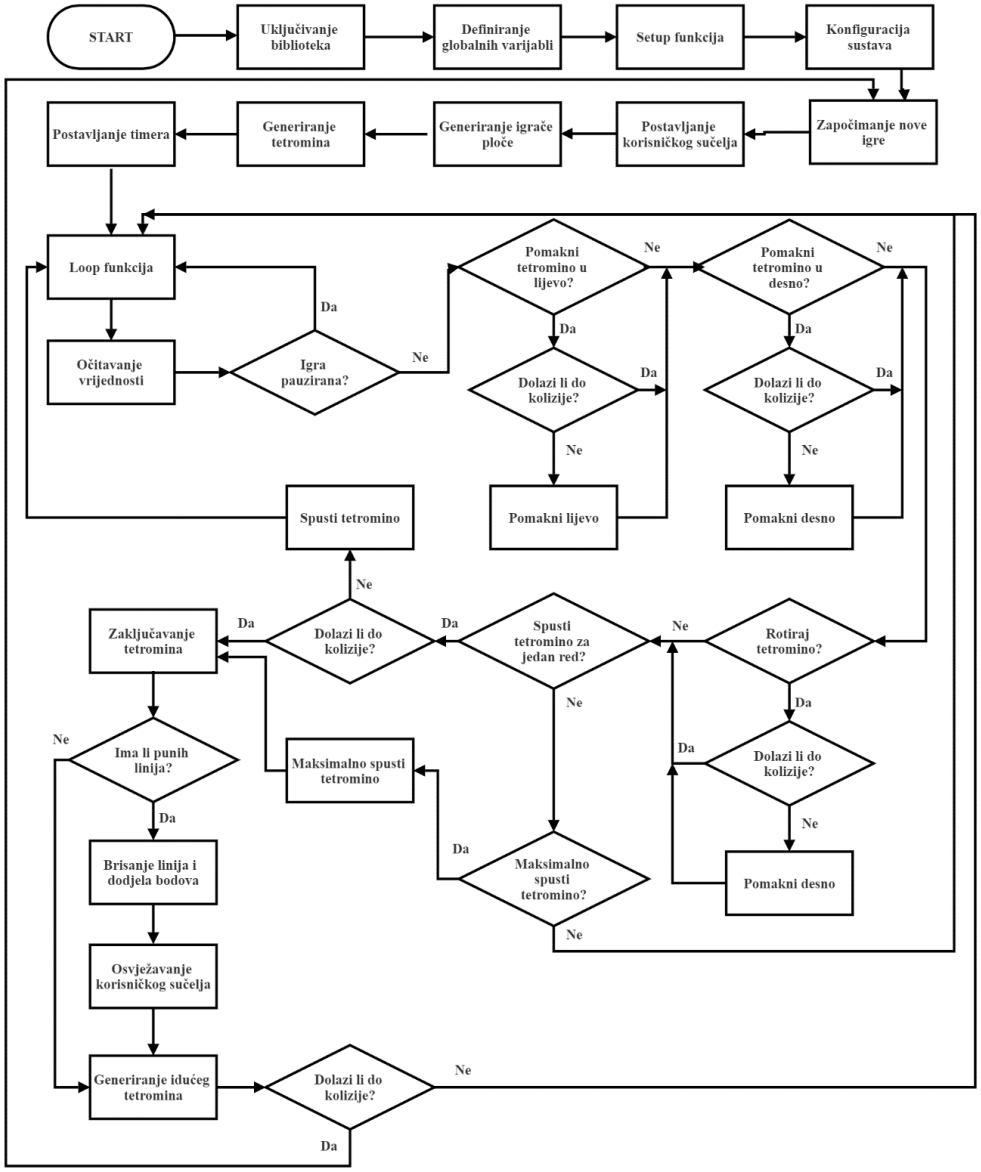


Slika 3.9 Izgled spojenog sustava

3.3 Realizacija algoritamskog rješenja

Nakon spajanja gumb modula, upravljačke palice i matričnih LED pokaznika sa mikrokontrolerom isprogramiran je rad sustava. Dobiveno algoritamsko rješenje radi kao i zamišljeno i omogućava ispravan rad sustava. Zadaća mikrokontrolera je procesiranje svih funkcija potrebnih za rad tetris igre kao i primanje upravljačkih signala te njihova inkorporacija u proces igre i naposljetku kontrola matričnih pokaznika kako bi se na njima ostvario prikaz igre. Programski kod u sebi

koristi biblioteke ESP32 HUB75 LED MATRIX PANEL DMA za jednostavniju kontrolu matričnih pokaznika i Adafruit_GFX koja nudi funkcije za kontrolu piksela kao i crtanje različitih geometrijskih oblika i ispisivanje teksta. Detaljan prikaz cijelog koda dostupan je u prilogu: **P.5**.



Slika 3.10 Dijagram toka algoritma mikrokontrolera

Na slici 3.10 je prikazan algoritam mikrokontrolera. Kao prvo potrebno je uključiti korištene biblioteke, izvesti pravilnu konfiguraciju sustava i definirati globalne varijable. Nakon što je sustav uspješno konfiguriran započinje setup funkcija. Unutar setup funkcije definiramo pinove gumb modula i upravljačke palice kao i njihove funkcije te pokrećemo crtanje na matrične pokaznike. Zatim se pomoću funkcije za pokretanje nove igre pokreće nova igra. Funkcija za pokretanje igre u sebi sadrži dodatne funkcije za generiranje tetromina, crtanje igrače ploče i komunikacijskog sučelja igre. Naposljetku se pokreće timer koji se koristi za određivanje vremenskih intervala u

kojima je potrebno aktivirati padanje tetromina. Nakon završetka setup funkcije pokreće se loop funkcija. Na početku svakog izvođenja loop funkcije vrši se očitanje vrijednosti sa gumb modula i upravljačke palice nakon čega se vrši provjera pauziranosti igre. Ukoliko igra nije pauzirana na temelju očitanih vrijednosti sa upravljačkih komponenti određuju se funkcije za izvršenje. Bitno je napomenuti da zbog brzine izvođenja loop funkcije nije dovoljno samo očitati vrijednost sa gumb modula jer za vrijeme jednog pritiska gumba loop funkcija se izvrši više puta što će rezultirati većim brojem izvođenja funkcije nego što je potrebno. Kako bi se rješio taj problem uvedene su dodatne varijable za praćenje prošlih i sadašnjih stanja gumbova kako bi se osiguralo da se za jedan pritisak gumba funkcija izvede samo jednom. Ako je utvrđeno da je pritisnut gumb koji pokreće funkciju za pomak u lijevo tada je potrebno izvesti provjeru kolizije odnosno provjeriti da li je pozicija lijevo od trenutne pozicije tetromina zauzeta. Ukoliko je pozicija zauzeta pomak se neće izvršiti, a ako pozicija nije zauzeta izvršiti će se pomak tetromina u lijevo. U svakom slučaju kada se dogodila promjena pozicije ili orijentacije tetromina isti je potrebno pomoću funkcije za brisanje tetromina obrisati te zatim pomoću funkcije za crtanje nacrtati na novoj poziciji. Funkcije za pomak u desno i rotaciju rade na isti način samo što se umjesto pomaka u lijevo izvršava pomak u desno ili rotacija. Nakon provjere prethodno navedenih funkcija vrši se provjera za spuštanje tetromina. Tetromino se mora spustiti u slučaju da je na odgovarajućem gumbu očitano da je pritisnut ili ako je na temelju timera utvrđeno da je vrijeme da se spusti. Timer započinje spuštanje tetromina u vremenskom intervalu od jedne sekunde. Ako je utvrđeno da tetromino treba spustiti, prvo se vrši provjera kolizije odnoso je li njegova buduća pozicija zauzeta. Ako je buduća pozicija zauzeta to znači da je tetromino dosegao dno igrače ploče ili već prethodno zaključani tetromino nakon čega se on zaključava te se provodi provjera punih linija. U slučaju da pri spuštanju tetromina ne dolazi do kolizije tetromino se spušta za jedan red i ponovno se pokreće loop funkcija. Ako na ploči nema punih linija generira se idući tetromino, ako ih ima one se brišu te se dodjeljuje odgovarajuća količina bodova. Nakon svake promjene bodova ili razine potrebno je osvježiti korisničko sučelje kako bi se u svakom trenutku prikazalo trenutno stanje bodova. Postoji i funkcija za maksimalno spuštanje tetromina koja se također pokreće pritiskom na odgovarajući gumb. Nakon pokretanja funkcije tetromino se spušta koliko god je moguće nakon čega se zaključava te se vrši proces provjere linija, bodovanja, osvježavanja korisničkog sučelja, te generiranje idućeg tetromina. Ako pri stvaranju idućeg tetromina dolazi do kolizije to znači da je igrača ploča popunjena do vrha te više nema mjesta za igru čime se označava kraj igre i ona se resetira. Detaljan prikaz svih spomenutih funkcija je dostupan u prilogu: **P.5**.

4.TESTIRANJE I REZULTATI

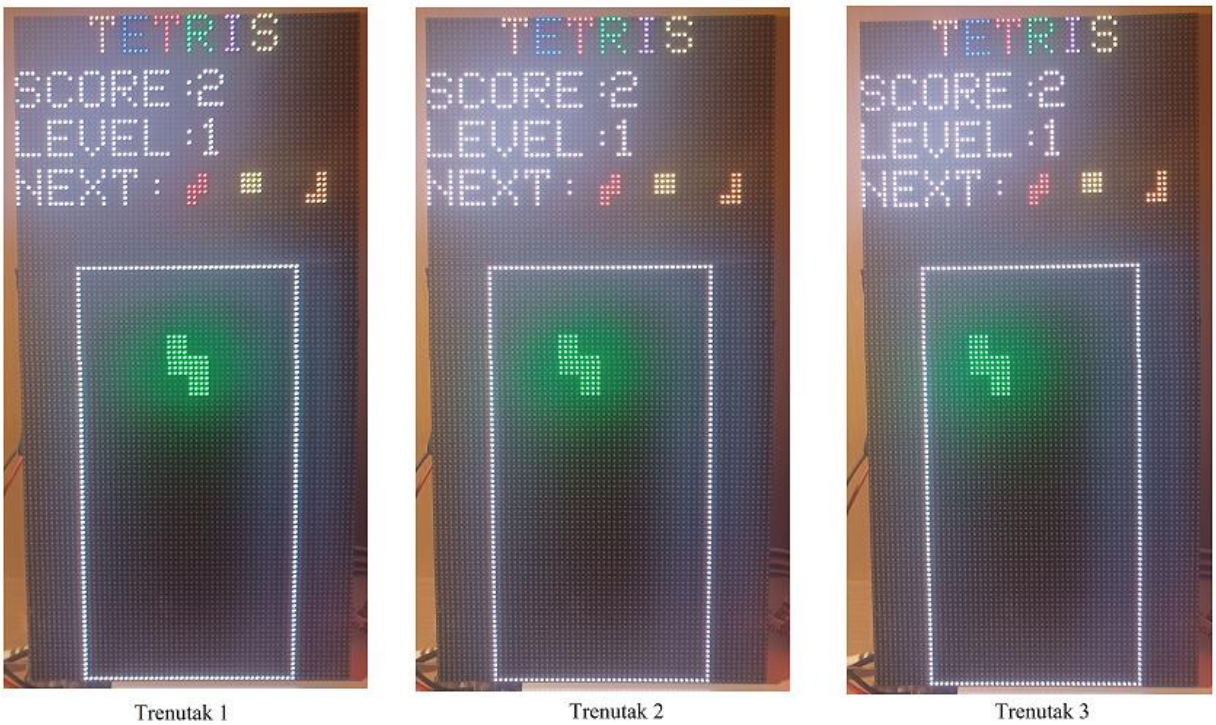
4.1 Metodologija testiranja

Prilikom testiranja utvrditi će se ispravnost dizajniranog algoritma i korištenih komponenata. Redom će se testirati svaka glavna funkcija koja za posljedicu ima promjenu ispisa za pokazniku te njezin rezultat prikazati na slikama.

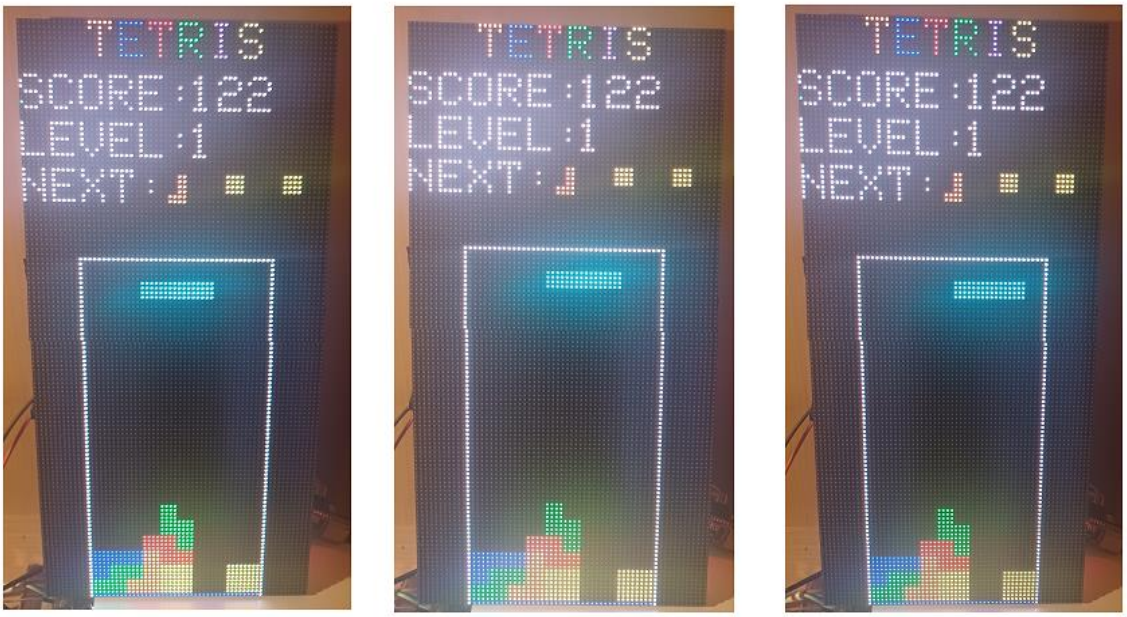
Igra mora biti responzivna, odnosno nakon aktiviranja upravljačke komponente odgovarajuća funkcija i njen prikaz na pokazniku mora se izvesti u kratkom vremenskom intervalu kako bi se osiguralo najbolje moguće iskustvo za korisnika.. Iz toga razloga biti će testirana brzina izvođenja isprogramiranih funkcija uz pomoć serijskog monitora Arduino razvojnog okruženja i funkcije micros() za praćenje broja mikrosekundi koje su prošle od pokretanja mikrokontrolera.

4.2 Rezultati testiranja

Testiranjem je utvrđeno kako sve komponente sustava rade ispravno te isprogramirane funkcije odrađuju svoje zadatke kao što je zamišljeno. Rezultati testiranja funkcija prikazani su na slikama 4.1 do 4.7.



Slika 4.1 Testiranje pomaka tetromina u lijevo

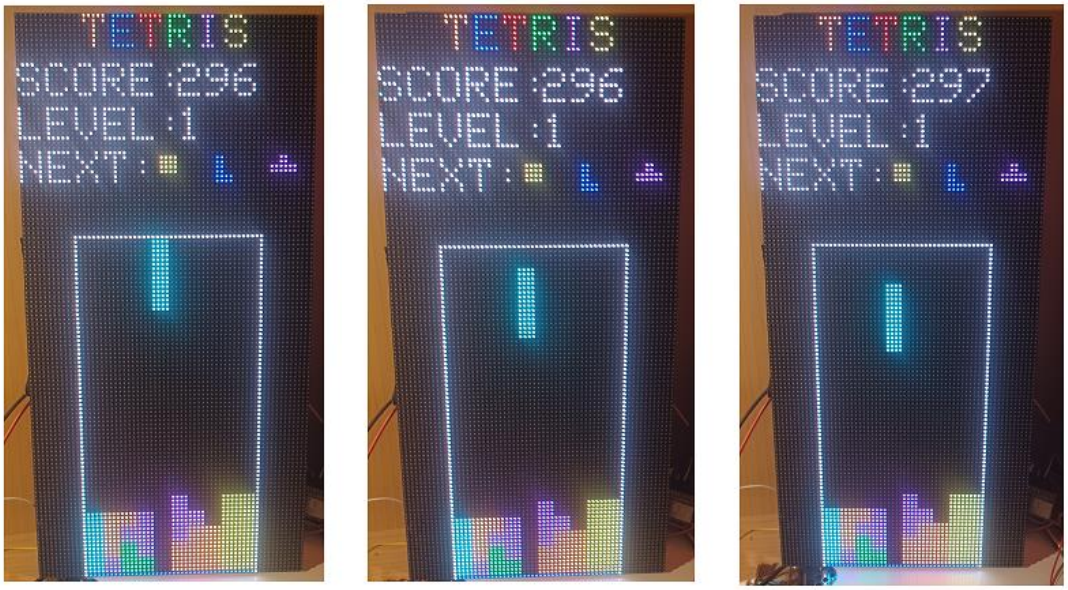


Slika 4.2 Testiranje pomaka tetromina u desno

Na slikama 4.1 i 4.2 prikazan je rezultat testiranja pomicanja tetromina u lijevo i u desno.

Slika 4.3 Testiranje rotacije tetromina

Slika 4.3 prikazuje rezultate testiranja rotacije tetromina. Tetromino je prikazan u tri različita trenutka. U prvom trenutku tetromino se nalazi u svome normalnom položaju gdje nije rotiran dok je u trenutku 2 rotiran za 90 stupnjeva, a u trenutku 3 za 180 stupnjeva.



Trenutak 1 Trenutak 2 Trenutak 3

Slika 4.4 Testiranje spuštanja tetromina po jedan red

Na slici 4.4 prikazano je spuštanje tetromina po jedan red. Spuštanje tetromina počinje primanjem ulaznog signala sa odgovarajuće upravljačke komponente ili na temelju postavljenog vremenskog intervala. U slučaju kada je spuštanje pokrenuto pomoću upravljačke komponente korisniku je potrebno dodijeliti jedan bod, a kada se promjeni broj bodova isti i ažurirati na pokazniku. Uspješno izvođenje toga postupka je vidljivo u prelasku iz trenutka 2 u trenutak 3.

Trenutak 1 Trenutak 2

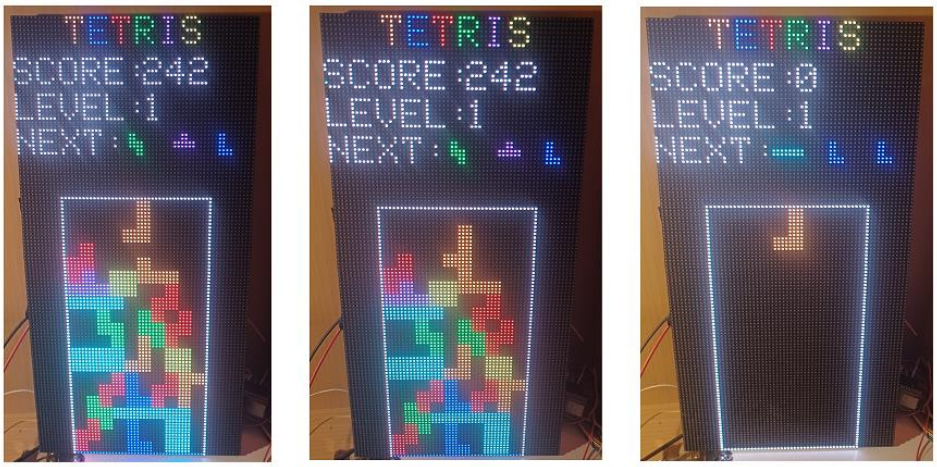
Slika 4.5 Testiranje trenutnog maksimalnog spuštanja tetromina

Na slici 4.5 prikazan je rezultat testiranja maksimalnog trenutnog spuštanja tetromina. U trenutku 1 tetromino se nalazi na vrhu igrače ploče dok se u trenutku 2 nalazi zaključan na dnu. Također se vidi da je uspješno generiran idući tetromino i dodjeljen bod za svaki spušteni red spuštanjem koje je pokrenuto neovisno o vremenskom intervalu te je ispravno ispisan trenutni broj bodova.

Slika 4.6 Testiranje provjere punih linija, brisanja, dodjele bodova i ispisivanja ispravnog stanja bodova na pokazniku

Na slici 4.6 prikazan je padajući tetromino u trenutku 1 i trenutku 2. U trenutku 3 tetromino doseže dno igrače ploče gdje se zaključava, nakon čega se pokreće provjera linija, brisanje linija, dodjela bodova i prikaz rezultata. Iz trenutaka 2 vidi se da će nakon zaključavanja donja tri reda biti potpuno popunjena i biti će ih potrebno obrisati. Iz trenutka 3 vidi se da su donje tri linije obrisane te ostali redovi iznad njih pomaknuti dolje. Prema rezultatu na vrhu pokaznika može se vidjeti da je on uspješno ažuriran te da je dodjeljen odgovarajući broj bodova.

Slika 4.7 prikazuje slučaj kada je igrača ploča ispunjena do vrha te više nije moguće stvoriti idući tetromino bez pojave kolizije što rezultira ponovnim pokretanjem igre. U prijelazu iz trenutka 2 u trenutak 3 vidljivo je da je prilikom generiranja idućeg tetromina došlo do kolizije i da je igra kao i sve ostale varijable za praćenje bodova, razine i tetromina uspješno resetirana i iznova ispisana na pokaznik.



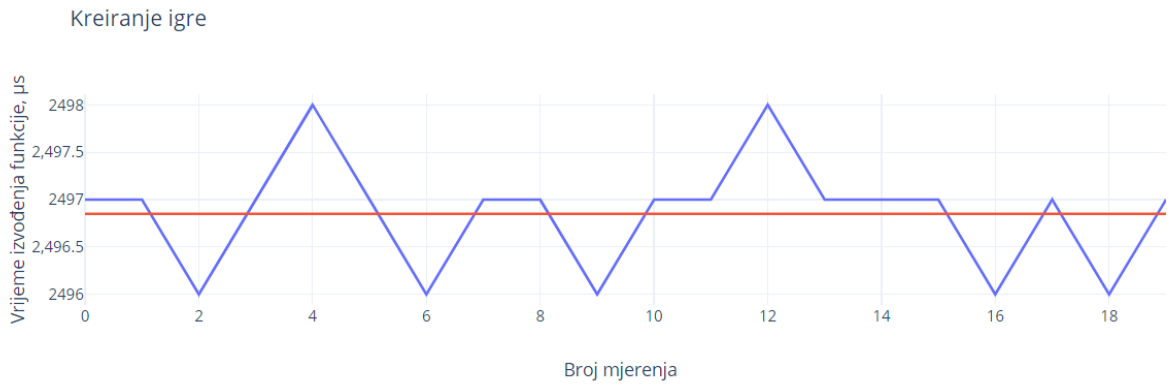
Trenutak 1 Trenutak 2 Trenutak 3

Slika 4.7 Testiranje detekcije kraja igre i početak nove

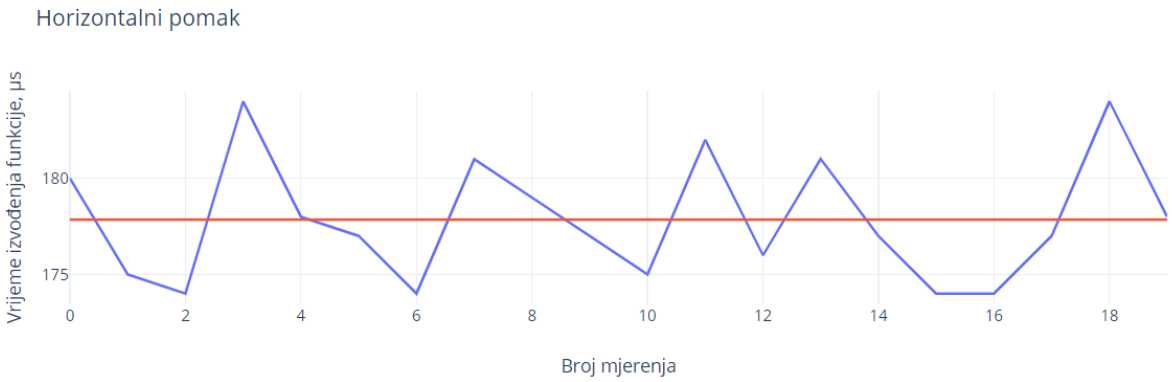
Tablica 4.1 Rezultati mjerenja vremena potrebnog za izvođenje funkcija

Broj testa	Kreiranje nove igre [µS]	Horizontalni pomak [µS]	Rotacija [µS]	Spuštanje [µS]	Trenutno spuštanje [µS]	Provjera linija i bodovanje [µS]
1.	2497	180	179	173	9496	22
2.	2497	175	171	172	9494	23
3.	2496	174	172	175	9501	22
4.	2497	184	172	172	9487	22
5.	2498	178	181	172	9489	23
6.	2497	177	173	173	9484	22
7.	2496	174	176	173	9502	22
8.	2497	181	174	175	9497	23
9.	2497	179	178	172	9504	22
10.	2496	177	172	171	9489	22
11.	2497	175	171	174	9496	23
12.	2497	182	171	175	9488	23
13.	2498	176	174	173	9501	23
14.	2497	181	179	174	9502	22
15.	2497	177	178	177	9485	22
16.	2497	174	172	171	9496	23
17.	2496	174	174	172	9503	23
18.	2497	177	180	172	9484	23
19.	2496	184	174	173	9491	22
20.	2497	178	177	172	9493	22
Prosječna vrijednost [µS]	2496.85	177.85	174.9	173.05	9494.1	22.45

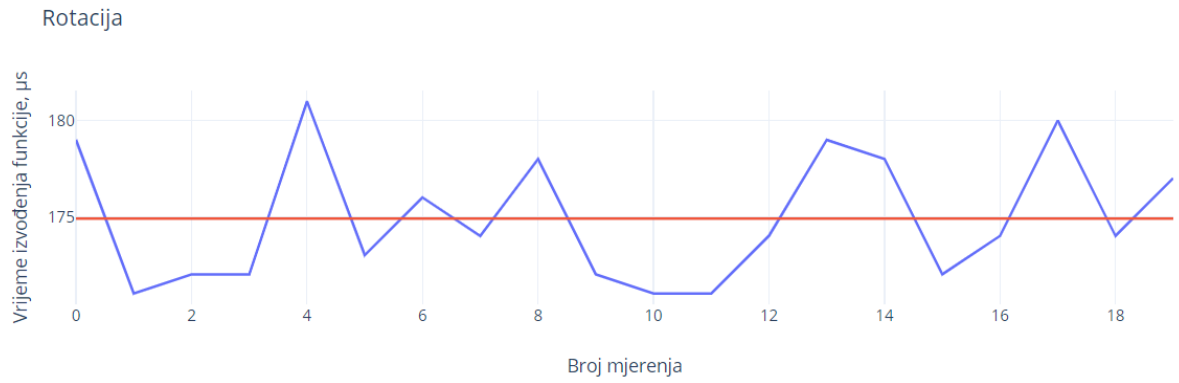
Iz tablice 4.1 možemo zaključiti da će igra imati odličnu responzivnost s obzirom da su rezultati izvođenja funkcija reda mikrosekundi i milisekundi. Funkcija sa najvećim potrebnim vremenom izvođenja je trenutni pad koja izravno ovisi o visini na kojoj se tetromino nalazi. Što je tetromino bliži dnu igrače ploče to će se funkcija izvesti brže. Mjerenje te funkcije je izvedeno pri maksimalnoj visini odnosno pri spuštanju od vrha ploče do dna igrače ploče. Rezultati mjerenja funkcije za provjeru i bodovanje su mjerenji pri pojavi jednog punog reda. Ako postoji više punih redova trajanje funkcije će se povećati.



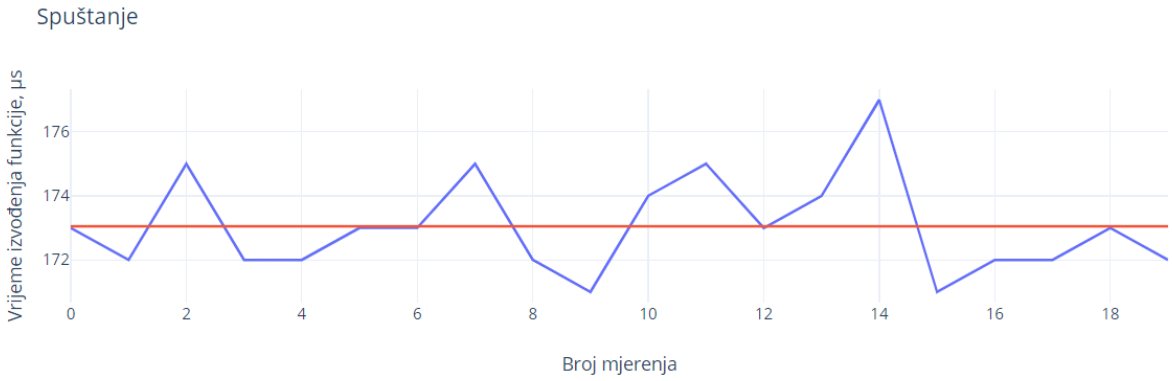
Slika 4.8 Graf vremena izvršavanja funkcije za kreiranje igre



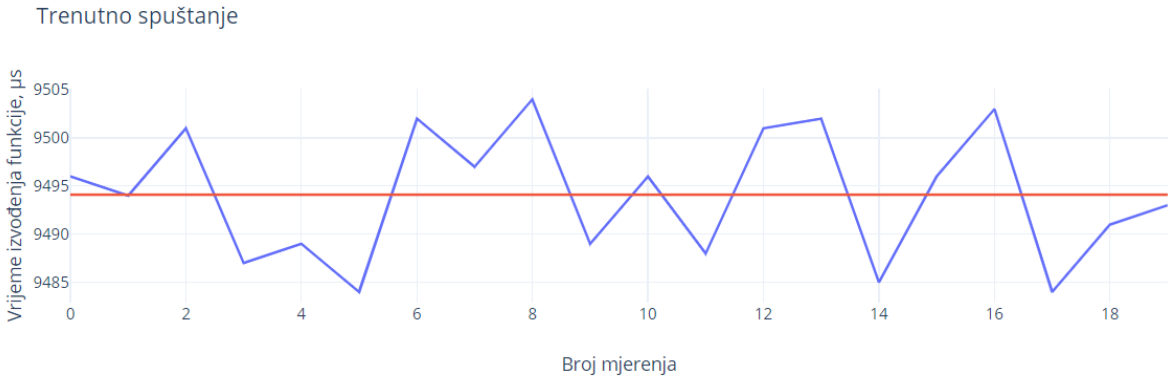
Slika 4.9 Graf vremena izvršavanja funkcije za horizontalni pomak



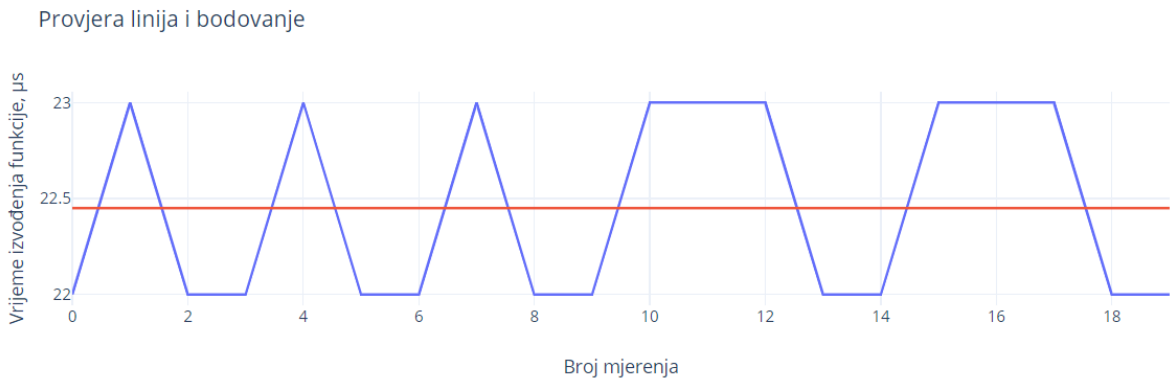
Slika 4.10 Graf vremena izvršavanja funkcije za rotiranje



Slika 4.11 Graf vremena izvršavanja funkcije za spuštanje



Slika 4.12 Graf vremena izvršavanja funkcije za trenutno spuštanje



Slika 4.13 Graf vremena izvršavanja funkcije za provjeru linija i bodovanje

5. ZAKLJUČAK

U završnom radu prikazano je kako pomoću ESP32-WROOM-32D mikrokontrolera, matričnih LED pokaznika, gumb modula i upravljačke palice izraditi tetris igru. Odabrani su matrični pokaznici rezolucije 64x64 kako bi se njihovim povezivanjem dobio pokaznik velike rezolucije kojim se može jasno prikazati igru. U radu je korišten ESP32-WROOM-32D mikroupravljač koji zbog svojih odličnih karakteristika može ispuniti zahtjeve za procesorskom snagom i koji zbog velikog broja pinova omogućuje povezivanje cijelog sustava. Proučeni su i korišteni postupci i metode poput DMA(Direct-memory access) izravnog pristupa memoriji i BCM(Binary-code modulation) modulacije koji će znatno smanjiti opterećenje na mikrokontroleru. Za programiranje mikrokontrolera korišteno je Arduino integrirano razvojno sučelje te su implementirane dodatne biblioteke poput ESP32 HUB75 LED MATRIX PANEL DMA koja značajno olakšava komunikaciju sa matričnim pokaznicima. Naposljetku je provedeno testiranje kako bi provjerili ispravnost dizajniranog rješenja.

LITERATURA

[1] *Data visualization in games,* https://gameanalytics.com/blog/data-visualization-games/ , pristup: 11.07.2023

[2] *LCD Liquid Crystal Display,* https://www.techtarget.com/whatis/definition/LCD-liquid-crystal-display, pristup: 11.07.2023

[3] *An introduction to OLED displays*, https://www.oled-info.com/oled-introduction, pristup: 11.07.2023

[4] *How RGB LEDs work and how to control color*, https://www.circuitbread.com/tutorials/how-rgb-leds-work-and-how-to-control-color, pristup: 04.07.2023

[5] *LED's explained*, https://theengineeringmindset.com/leds-explained/, pristup: 04.07.2023

[6] *What is the visible light spectrum?*, https://www.thoughtco.com/the-visible-light-spectrum-2699036, pristup: 04.07.2023

[7] *How to dim an LED*, https://www.circuitbread.com/tutorials/how-to-dim-an-led, pristup: 11.07.2023

[8]*Binary code modulation*, https://www.programming-electronics-diy.xyz/2021/01/binary-code-modulation-bcm-aka-bit.html, pristup: 04.07.2023

[9] *Designing game controls*, https://www.gamedeveloper.com/disciplines/designing-game-controls, pristup: 11.07.2023

[10] *ESP32WROOM32D & ESP32WROOM32U datasheet*, https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf, pristup: 04.07.2023

[11] *ESP32-DevKitC V4 Getting Started Guide*, https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html#get-started-esp32-devkitc-board-front, pristup: 04.07.2023

[12] *Arduino-ESP32*, https://github.com/espressif/arduino-esp32, pristup: 04.07.2023

[13] *Everything You Didn't Want to Know About RGB Matrix Panels*, https://www.sparkfun.com/news/2650, pristup: 04.07.2023

[14] *Driving a 8x8 RGB LED matrix with Arduino*, https://rjk.codes/post/driving-a-8x8-rgb-led-matrix-with-arduino/, pristup: 11.07.2023

[15] *LED panel schematic*, https://github.com/esden/led-panel-docs/blob/master/P2-1515-64x64-32S-V1.0/led-panel-P2-1515-64x64-32S-V1.0-sch.pdf, pristup: 11.07.2023

[16] *ESP32-HUB75-MatrixPanel-DMA*, https://github.com/mrfaptastic/ESP32-HUB75-MatrixPanel-DMA, pristup: 04.07.2023

[17] *Adafruit-GFX-Library* , https://github.com/adafruit/Adafruit-GFX-Library, pristup: 04.07.2023

SAŽETAK

Cilj ovoga rada bila je izrada Tetris igre, te projektiranje sustava koji ju može podržati. Za mikrokontroler je odabran ESP32-WROOM-32D zbog svojih odličnih karakteristika te velikog broja pinova. U radu su implementirani postupci i metode kako bi se smanjio teret na mikrokontroleru te su korištene dodatne biblioteke unutar razvojnog okruženja kako bi se olakšao postupak pisanja koda. Jasan prikaz igre je omogućen upotrebom dva matrična LED pokaznika rezolucije 64x64 piksela. Korisnik upravlja igrom pomoću upravljačkog uređaja sa pet gumb modula i upravljačkom palicom. Na kraju je sustav testiran kako bi se utvrdila njegova ispravnost.

ABSTRACT

The goal of this project was to create a Tetris game, and to design and explain a system that can support it. The ESP32-WROOM-32D was chosen for the microcontroller due to its excellent characteristics and a large number of pins. The project implemented procedures and methods to reduce the load on the microcontroller and additional libraries were used within the development enviroment to simplify the process of writing code. A clear display of the game is made possible with the use of two matrix LED displays with a resolution of 64x64 pixels. The user controls the game using a controling device with five button modules and a joystick module. Finally, the system was tested to determine its correctness.

ŽIVOTOPIS

Robert Pavić rođen je 22.04.2000. u Slavonskom Brodu. Nakon završetka osnovne škole upisuje Tehničku školu u Slavonskom Brodu gdje po završetku stječe zanimanje elektrotehničar. Svoje obrazovanje nastavlja na preddiplomskom sveučilišnom studiju Elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

PRILOZI

1.Tehničke karakteristike ESP32-WROOM-32D mikrokontrolera:

Microprocessor	Dvojezgreni Tensilica Xtensa LX6
Maksimalna radna frekvencija	240MHz
Radni napon	3.3V
Analogni ulazni pinovi	12-bit, 18 kanal
DAC Pinovi	8-bit, 2 kanal
Digitalni I/O Pinovi	39 (34 su normalni GPIO pinovi)
DC struja na I/O Pinovima	40 mA
DC struja na 3.3V Pinu	50 mA
SRAM	520 KB
Komunikacija	SPI(4), I2C(2), I2S(2), CAN, UART(3)
Wi-Fi	802.11 b/g/n
Bluetooth	V4.2 – Podržava BLE i klasični Bluetooth

2. ESP32-WROOM-32D nazivi pinova i njihove funkcionalnosti

Naziv	Tip	Funkcija
3V3	P	Napajanje 3.3V
EN	I	CHIP_PU, Reset
VP	I	GPIO36, ADC1_CH0, S_VP
VN	I	GPIO39, ADC1_CH3, S_VN
IO34	I	GPIO34, ADC1_CH6, VDET_1
IO35	I	GPIO35, ADC1_CH7, VDET_2
IO32	I/O	GPIO32, ADC1_CH4, TOUCH_CH9, XTAL_32K_P
IO33	I/O	GPIO33, ADC1_CH5, TOUCH_CH8, XTAL_32K_N
IO25	I/O	GPIO25, ADC1_CH8, DAC_1
IO26	I/O	GPIO26, ADC2_CH9, DAC_2
IO27	I/O	GPIO27, ADC2_CH7, TOUCH_CH7
IO14	I/O	GPIO14, ADC2_CH6, TOUCH_CH6, MTMS
IO12	I/O	GPIO12, ADC2_CH5, TOUCH_CH5, MTDI
GND	G	Ground
IO13	I/O	GPIO13, ADC2_CH4, TOUCH_CH4, MTCK
D2	I/O	GPIO9, D2
D3	I/O	GPIO10, D3
CMD	I/O	GPIO11, CMD

5V	P	Napajanje 5V
GND	G	Ground
IO23	I/O	GPIO23
IO22	I/O	GPIO22
TX	I/O	GPIO1, U0TXD
RX	I/O	GPIO3, U0RXD
IO21	I/O	GPIO21
GND	G	Ground
IO19	I/O	GPIO19
IO18	I/O	GPIO18
IO5	I/O	GPIO5
IO17	I/O	GPIO17
IO16	I/O	GPIO16
IO4	I/O	GPIO4, ADC2_CH0, TOUCH_CH0
IO0	I/O	GPIO0, ADC2_CH1, TOUCH_CH1, Boot
IO2	I/O	GPIO2, ADC2_CH2, TOUCH_CH2
IO15	I/O	GPIO15, ADC2_CH3, TOUCH_CH3, MTDO
D1	I/O	GPIO8, D1
D0	I/O	GPIO7, D0
CLK	I/O	GPIO6, CLK

P: Napajanje; I: Ulaz; O: Izlaz.

Pinovi D0, D1, D2, D3, CMD i CLK interno se koriste za komunikaciju između ESP32 i SPI flash memorije. Grupirani su s obje strane blizu USB priključka. Izbjegavajte korištenje ovih iglica jer to može poremetiti pristup SPI flash memoriji / SPI RAM-u.

Pinovi GPIO16 i GPIO17 dostupni su za korištenje samo na pločama s modulima ESP32-WROOM i ESP32-SOLO-1. Ploče s ESP32-WROVER modulima imaju pinove rezervirane za internu upotrebu.

3. Tehničke karakteristike P3 SMD2121 192x192 1/32S RGB LED matričnog pokaznika

Čip	Epstar
Veličina čipa	Crvena 9 mil, Zelena 12. Plava 12mil
Tip svjetlosti	SMD2121
Veličina piksela	3mm
Veličina modula	192mmX192mm
Rezolucija modula	64 X 64 (dots)
Gustoća piksela	111111dots/m2
Konfiguracija piksela	1R1G1B
Valna duljina crvena	625±2nm
Valna duljina zelena	525±2nm
Valna duljina plava	470±2nm
Boje	16777216

Optimalna udaljenost gledanja	≥2m
Kut gledanja	120°/90°
Maksimalna potrošnja snage	20W
Osvjetljenost	1500cd/m2
Sivi tonovi	14 bita po boji
Temperatura boje	6500k
Metoda kontrole	Sinkronizacija ili asinkronizacija
Driving method	1/32 skeniranje
Frekvencija okvira	60Hz
Frekvencija osvježavanja	≥600Hz
Radni napon	DC5V
IP ocjena	IP54
MTBF	Preko 8,000 sati
Životni vijek	80,000 sati

4. Detaljan opis veza između pinova komponenata izražen u tabličnom obliku

Gumb modul br.1	**Gumb modul br.2**	**Gumb modul br.3**	**Gumb modul br.4**	**Gumb modul br.5**	**ESP32 Pločica**
VCC	VCC	VCC	VCC	VCC	5V
GND	GND	GND	GND	GND	GND
OUT					2
	OUT				VP
		OUT			35
			OUT		18
				OUT	22

Tablica 1. Veze između pinova gumb modula i ESP32 pločice

Joystick modul	**ESP32 Pločica**
VCC	VCC
GND	GND
SW	34
VRx	32
VRy	33

Tablica 2. Veze između pinova joystick modula i ESP32 pločice

RGB pokaznik br.1	**ESP32 Pločica**
Data Input	
R1	25
B1	27
R2	14
B2	13
A	23
C	5
CLK	16
OE	15
G1	26
GND	
G2	12
E	21
B	19
D	17
LAT	4
GND	GND

Tablica 3. Veze između Data input priključka RGB pokaznika br.1 i ESP32 pločice

RGB pokaznik br.1	**RGB pokaznik br.2**
Data Output	**Data Input**
R1	R1
B1	B1
R2	R2
B2	B2
A	A
C	C
CLK	CLK

OE	OE
G1	G1
GND	GND
G2	G2
E	E
B	B
D	D
LAT	LAT
GND	GND

Tablica 4. Veze između Data output priključka RGB pokaznika br.1 i Data input priključka RGB pokaznika br.2

RGB pokaznik br.1	RGB pokaznik br.2	Izvor napajanja
VCC	VCC	+V
VCC	VCC	+V
GND	GND	-V
GND	GND	-V

Tablica 5. Veze između priključaka napajanja RGB pokaznika i izvora napajanja

5. Programski kod mikrokontrolera

```
#define R1 25
#define G1 26
#define B1 27
#define R2 14
#define G2 12
#define B2 13
#define A 23
#define B 19
#define C 5
#define D 17
#define E 21
#define CLK 16
#define LAT 4
#define OE 15

#define BTN_LEFT 2
```

```
#define BTN_SOFTDROP 36
#define BTN_RIGHT 35
#define BTN_HARDDROP 18
#define BTN_ROTATE 22
#define BTN_SW 34
#define JS_X_AXIS 33
#define JS_Y_AXIS 32

#define BLOCK_SIZE 4

#include <ESP32-HUB75-MatrixPanel-I2S-DMA.h>

#define PANEL_RES_X 64      // Širina pojedinog pokaznika u pikselima
#define PANEL_RES_Y 64      // Visina pojedinog pokaznika u pikselima
#define PANEL_CHAIN 2    // Ukupan broj povezanih pokaznika

//MatrixPanel_I2S_DMA dma_display;
MatrixPanel_I2S_DMA *dma_display = nullptr;

  // Konfiguracija modula
  HUB75_I2S_CFG::i2s_pins _pins={R1, G1, B1, R2, G2, B2, A, B, C, D, E, LAT, OE,
CLK};
  HUB75_I2S_CFG mxconfig(
    PANEL_RES_X,   // širina modula
    PANEL_RES_Y,   // visina modula
    PANEL_CHAIN,   // broj povezanih pokaznika
    _pins // pin map
  );

uint16_t myWHITE = dma_display->color565(255, 255, 255);
uint16_t myBlue = dma_display->color565(0, 0, 255);
uint16_t myCyan = dma_display->color565(0, 255, 255);
uint16_t myYellow = dma_display->color565(255, 255, 0);
uint16_t myRed = dma_display->color565(255, 0, 0);
uint16_t myPurple = dma_display->color565(160, 32, 240);
uint16_t myOrange = dma_display->color565(255, 165, 0);
uint16_t myGreen = dma_display->color565(0, 255, 0);
uint16_t RGB[]={myOrange,myBlue,myRed,myGreen,myPurple,myYellow,myCyan};

hw_timer_t * timer = NULL;
const int width =10;
const int height =20;
int piece_origin_x=(width/2-4/2);
int piece_origin_y=0;
int pieceIndex=0;
int* pieceIndexptr=&pieceIndex;
int draw_origin_x=12;
int draw_origin_y=47;
int* origin_y_ptr=&draw_origin_y;
```

```
int* origin_x_ptr=&draw_origin_x;
int currentRotation=0;
int* currentRotation_ptr=&currentRotation;
int score = 0;
int* score_ptr=&score;
int level = 1;
int* level_ptr=&level;
int lines = 0;
int* lines_ptr=&lines;
bool pauseGame = false;

int buttonState_LEFT = 0;
int buttonState_SOFTDROP = 0;
int buttonState_RIGHT = 0;
int buttonState_HARDDROP = 0;
int buttonState_ROTATE = 0;
int buttonState_SW = 0;

int lastButtonState_LEFT = 0;
int lastButtonState_SOFTDROP = 0;
int lastButtonState_RIGHT = 0;
int lastButtonState_HARDDROP = 0;
int lastButtonState_ROTATE = 0;
int lastButtonState_SW = 0;

int Pieces[4]={0};
int Board[width][height+1]={0};
int ActivePiece[4][4]={0};
String str;

  int piece_L[4][4] = { // definiranje oblika tetromina
    {1, width+1, width*2+1, width*2+2},
    {width+2, width*2, width*2+1, width*2+2},
    {1, 2, width+2, width*2+2},
    {width, width+1, width+2, width*2}
  };

  int piece_J[4][4] = {
    {2, width+2, width*2+1, width*2+2},
    {width, width*2, width*2+1, width*2+2},
    {1, width+1, width*2+1, 2},
    {0, 1, 2, width+2}
  };

  int piece_Z[4][4] = {
    {0,width,width+1,width*2+1},
    {width+1, width+2,width*2,width*2+1},
    {0,width,width+1,width*2+1},
    {width+1, width+2,width*2,width*2+1}
```

```
  };

  int piece_S[4][4] = {
    {2,width+1,width+2,width*2+1},
    {width, width+1,width*2+1,width*2+2},
    {2,width+1,width+2,width*2+1},
    {width, width+1,width*2+1,width*2+2}
  };

  int piece_T[4][4] = {
    {1,width,width+1,width+2},
    {1,width+1,width+2,width*2+1},
    {width,width+1,width+2,width*2+1},
    {1,width,width+1,width*2+1}
  };

  int piece_O[4][4] = {
    {1,2,width+1,width+2},
    {1,2,width+1,width+2},
    {1,2,width+1,width+2},
    {1,2,width+1,width+2},
  };

  int piece_I[4][4] = {
    {width,width+1,width+2,width+3},
    {1,width+1,width*2+1,width*3+1},
    {width,width+1,width+2,width+3},
    {1,width+1,width*2+1,width*3+1}
  };

void drawBoard(int Board[width][height+1]){ //funkcija za crtanje igrače ploče na
pokaznik
  for (int i=0; i<width; i++) {
    for (int j=0; j<height+1; j++) {
        if(Board[i][j]<7){
                  dma_display-
>fillRect(*origin_y_ptr+j*BLOCK_SIZE,*origin_x_ptr+i*BLOCK_SIZE,BLOCK_SIZE,BLOCK_
SIZE,RGB[Board[i][j]]);
        }
    }
  }
};

void refreshBoard(int Board[width][height+1]){ //funkcija za za osvježavanje
igrače ploče na pokazniku
  for (int i=0; i<width; i++) {
    for (int j=0; j<height+1; j++) {
        if(Board[i][j]==7){
```

```
                    dma_display-
>fillRect(*origin_y_ptr+j*BLOCK_SIZE,*origin_x_ptr+i*BLOCK_SIZE,BLOCK_SIZE,BLOCK_
SIZE,0);
        }
    }
  }
};

void drawPiece(int piece[][4]){ //funkcija za crtanje tetromina na pokaznik
    for (int i=0; i<4; i++) {
      dma_display-
>fillRect(*origin_y_ptr+piece[*currentRotation_ptr][i]/width*BLOCK_SIZE,*origin_x
_ptr+(piece[*currentRotation_ptr][i]%width)*BLOCK_SIZE,BLOCK_SIZE,BLOCK_SIZE,RGB[
pieceIndex]);
  }
};

void erasePiece(int piece[][4]){ //funkcija za brisanje tetromina sa pokaznika
    for (int i=0; i<4; i++) {
      dma_display-
>fillRect(*origin_y_ptr+piece[*currentRotation_ptr][i]/width*BLOCK_SIZE,*origin_x
_ptr+(piece[*currentRotation_ptr][i]%width)*BLOCK_SIZE,BLOCK_SIZE,BLOCK_SIZE,0);
  }
};

void generatePiece(int Board[width][height+1],int piece[][4],int
currentRotation,int pieces[4]){ //funkcija za generiranje aktivnog tetromina
  *currentRotation_ptr=0;
  if(pieces[0]<=1){
    *pieceIndexptr=0;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piece[i][j]=piece_L[i][j]+width*piece_origin_y+piece_origin_x;
      }
    }
  }
  if(pieces[0]>1 && pieces[0]<=2){
    *pieceIndexptr=1;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piece[i][j]=piece_J[i][j]+width*piece_origin_y+piece_origin_x;
      }
    }
  }
  if(pieces[0]>2 && pieces[0]<=3){
    *pieceIndexptr=2;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piece[i][j]=piece_Z[i][j]+width*piece_origin_y+piece_origin_x;
```

```
      }
    }
  }
    if(pieces[0]>3 && pieces[0]<=4){
    *pieceIndexptr=3;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piece[i][j]=piece_S[i][j]+width*piece_origin_y+piece_origin_x;
      }
    }
  }
    if(pieces[0]>4 && pieces[0]<=5){
    *pieceIndexptr=4;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piece[i][j]=piece_T[i][j]+width*piece_origin_y+piece_origin_x;
      }
    }
  }
    if(pieces[0]>5 && pieces[0]<=6){
    *pieceIndexptr=5;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piece[i][j]=piece_O[i][j]+width*piece_origin_y+piece_origin_x;
      }
    }
  }
    if(pieces[0]>6 && pieces[0]<=7){
    *pieceIndexptr=6;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piece[i][j]=piece_I[i][j]+width*piece_origin_y+piece_origin_x;
      }
    }
  }
  for(int i=1;i<4;i++){
    pieces[i-1]=pieces[i];
  }
  pieces[3]=random(0,8);

  for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
      if(Board[(piece[currentRotation][i])%width][(piece[currentRotation][i])/wid
th]<7){
        RestartGame(Board,piece,pieces);
      }
    }
    }
};
```

```
void lockPiece(int Board[width][height+1],int piece[][4],int colorIndex,int
currentRotation,int pieces[4]){ //funkcija za zaključavanje tetromina
  for(int i=0;i<4;i++){
    Board[(piece[currentRotation][i])%width][(piece[currentRotation][i])/width]=c
olorIndex;
  }
  checkLines(Board);
  refreshBoard(Board);
  drawBoard(Board);
  generatePiece(Board,piece,currentRotation,pieces);
  refreshScoreBoard(pieces);
  drawPiece(piece);
};

void softDrop(int Board[width][height+1],int piece[][4],int currentRotation, int
direction,int colorIndex,int pieces[4]){ //funkcija za padanje za jedan red
  if(!checkVerticalCollision(Board,piece,currentRotation,direction)){
    erasePiece(piece);
    if(direction==1){
     for(int i=0;i<4;i++){
       for(int j=0;j<4;j++){
         piece[i][j]=piece[i][j]+width;
       }
     }
    }
    drawPiece(piece);
  }else{
    lockPiece(Board,piece,colorIndex,currentRotation,pieces);
  }
    refreshScoreBoard(pieces);
};

void hardDrop(int Board[width][height+1],int piece[][4],int currentRotation, int
direction,int colorIndex,int pieces[4]){ //funkcija za trenutno maksimalno
padanje
    bool collision=false;
    while(collision == false){
      if(!checkVerticalCollision(Board,piece,currentRotation,direction)){
    erasePiece(piece);
    if(direction==1){
     for(int i=0;i<4;i++){
       for(int j=0;j<4;j++){
         piece[i][j]=piece[i][j]+width;
       }
     }
    }
    drawPiece(piece);
    *score_ptr=*score_ptr+1;
```

```
  }else{
    collision = true;
    lockPiece(Board,piece,colorIndex,currentRotation,pieces);
  }
  }
    refreshScoreBoard(pieces);
};

void moveHorizontaly(int Board[width][height+1],int piece[][4],int
currentRotation, int direction){ //funkcija za horizontalni pomak
  if(!checkHorizontalCollision(Board,piece,currentRotation,direction)){
    erasePiece(piece);
    if(direction==1){
     for(int i=0;i<4;i++){
       for(int j=0;j<4;j++){
         piece[i][j]=piece[i][j]+1;
       }
     }
    }else if (direction==-1){
     for(int i=0;i<4;i++){
       for(int j=0;j<4;j++){
         piece[i][j]=piece[i][j]-1;
       }
     }
    }
    drawPiece(piece);
  }
};

bool checkHorizontalCollision(int Board[width][height+1],int piece[][4],int
currentRotation,int direction){ //funkcija za provjeru horizontalne kolizije
    for(int i =0;i<4;i++){
  int x1=(piece[currentRotation][i])%(width);
  int x2=(piece[currentRotation][i]+direction)%(width);
  int y1=(piece[currentRotation][i])/(width);
  int y2=(piece[currentRotation][i]+direction)/(width);
  if(y2!=y1){
    return true;
  }
  if(Board[x2][y2]<7){
    return true;
  }
  }
  return false;
};

bool checkVerticalCollision(int Board[width][height+1],int piece[][4],int
currentRotation,int direction){ //funkcija za provjeru vertikalne kolizije
  for(int i=0;i<4;i++){
```

```
    if(Board[(piece[currentRotation][i])%width][(piece[currentRotation][i]+width)
/width]<7){
      return true;
    }
  }
  return false;
};

void rotatePiece(int Board[width][height+1],int piece[][4]){ // funkcija za
rotaciju
  bool collision=false;
  int temp= *currentRotation_ptr<3?*currentRotation_ptr+1:0;
  for(int i=0;i<4;i++){
    if((Board[piece[temp][i]%(width)][piece[temp][i]/(width)]<7)){
        collision = true;
    }
  }
  if(!collision){
    erasePiece(piece);
    *currentRotation_ptr<3?*currentRotation_ptr=*currentRotation_ptr+1:*currentRo
tation_ptr=0;
    drawPiece(piece);
  }
};

void generatePieces(int pieces[4]){ //generiranje liste tetromina
  for(int i=0;i<4;i++){
     pieces[i]=random(0,8);
     Serial.print(pieces[i]);
  }
};

void refreshScoreBoard(int pieces[4]){ //funkcija za osvježavanje prikaza
bodova,levela i budućih tetromina
  dma_display->fillRect(10,0,16,31,0);
  dma_display->fillRect(27,0,8,37,0);
    dma_display->setRotation(3);
  int text_size = 1;
  dma_display->setTextSize(text_size);
  dma_display->setTextWrap(false);

  dma_display->setCursor(34, 10);
  uint8_t u = 0;
  String myString = String(score);
  for (u=0; u<myString.length(); u++) {
    dma_display->setTextColor(myWHITE);
    dma_display->print(myString[u]);
  }
    dma_display->setCursor(34, 19);
```

```
uint8_t y = 0;
String myString2 = String(level);
for (y=0; y<myString2.length(); y++) {
  dma_display->setTextColor(myWHITE);
  dma_display->print(myString2[y]);
}
dma_display->setRotation(0);

for(int j=0; j<3;j++){
int piecetemp[4][4]={0};
int color=0;
if(pieces[j]<=1){
  for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
    piecetemp[i][j]=piece_L[i][j];
    }
  }
}
if(pieces[j]>1 && pieces[j]<=2){
  color=1;
  for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
    piecetemp[i][j]=piece_J[i][j];
    }
  }
}
if(pieces[j]>2 && pieces[j]<=3){
  color=2;
  for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
    piecetemp[i][j]=piece_Z[i][j];
    }
  }
}
  if(pieces[j]>3 && pieces[j]<=4){
    color=3;
  for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
    piecetemp[i][j]=piece_S[i][j];
    }
  }
}
  if(pieces[j]>4 && pieces[0]<=5){
    color=4;
  for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
    piecetemp[i][j]=piece_T[i][j];
    }
  }
```

```
  }
    if(pieces[j]>5 && pieces[j]<=6){
      color=5;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piecetemp[i][j]=piece_O[i][j];
      }
    }
  }
    if(pieces[j]>6 && pieces[j]<=7){
      color=6;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
      piecetemp[i][j]=piece_I[i][j];
      }
    }
  }
  int yn= 29;
  int xn= 4;
    for (int i=0; i<4; i++) {
      dma_display-
>fillRect(yn+(piecetemp[0][i]/width)*2,xn+j*4*3+(piecetemp[0][i]%width)*2,2,2,RGB
[color]);
    }
  }

};

void loadGUI(int pieces[4]){ // funkcija za učitavanje korisničkog sučelja
  dma_display->drawRect(46, 11, BLOCK_SIZE*20+2, BLOCK_SIZE*10+2, myWHITE);
  dma_display->setRotation(3);

  int text_size=1;
  dma_display->setTextSize(text_size);
  dma_display->setTextWrap(false);

  dma_display->setCursor(14, 0);
  uint8_t w = 0;
  const char *str1 = "TETRIS";
  for (w=0; w<strlen(str1); w++) {
    dma_display->setTextColor(RGB[w]);
    dma_display->print(str1[w]);
  }
    dma_display->setCursor(0, 10);
  uint8_t u = 0;
  const char *str2 = "SCORE:";
  for (u=0; u<strlen(str2); u++) {
    dma_display->setTextColor(myWHITE);
    dma_display->print(str2[u]);
```

```
  }
      dma_display->setCursor(0, 19);
  uint8_t q = 0;
  const char *str3 = "LEVEL:";
  for (q=0; q<strlen(str3); q++) {
    dma_display->setTextColor(myWHITE);
    dma_display->print(str3[q]);
  }
       dma_display->setCursor(0, 28);
  uint8_t a = 0;
  const char *str4 = "NEXT:";
  for (a=0; a<strlen(str4); a++) {
    dma_display->setTextColor(myWHITE);
    dma_display->print(str4[a]);
  }
  dma_display->setRotation(0);
  refreshScoreBoard(pieces);
}

void RestartGame(int Board[width][height+1],int piece[][4],int pieces[4]){ //
funkcija za započimanje nove igre
  dma_display->clearScreen();
  for(int i=0;i<width;i++){
    for(int j=0;j<height;j++){
      Board[i][j]=7;
    }
  }
  for(int i=0;i<width;i++){
    Board[i][height]=1;
  }
  generatePieces(pieces);
  generatePiece(Board,piece,*currentRotation_ptr,pieces);
  drawBoard(Board);
  drawPiece(piece);
  loadGUI(pieces);
};

void checkLines(int Board[width][height+1]){ //provjera linija i dodjela bodova
  int linesCounter=0;
    for (int j=0; j<height; j++) {
      int counter=0;
      for (int i=0; i<width; i++){
        if(Board[i][j]<7){
          counter++;
        }
        if(counter==width){
          *lines_ptr=*lines_ptr+1;
          *level_ptr=*lines_ptr/10+1;
          linesCounter++;
```

```
          for(int k=j; k>0; k--){
            for(int i=0;i<width;i++){
              Board[i][k]=Board[i][k-1];
            }
          }
          if(j==0){
            for(int i=0;i<width;i++){
              Board[i][j]=7;
            }
          }
        }
      }
    }
    if(linesCounter==1){
      *score_ptr=*score_ptr+(*level_ptr*100);
      }
      else if(linesCounter==2){
      *score_ptr=*score_ptr+(*level_ptr*300);
      }
      else if(linesCounter==3){
      *score_ptr=*score_ptr+(*level_ptr*500);
      }
      else if(linesCounter==4){
      *score_ptr=*score_ptr+(*level_ptr*800);
      }
};

void IRAM_ATTR timer_isr() { //kada timer dosegne potrebnu vrijednost aktivira se
funkcija za padanje tetromina
    softDrop(Board,ActivePiece,currentRotation,1,pieceIndex,Pieces);
}

void setup() {
  Serial.begin(230400);
  dma_display = new MatrixPanel_I2S_DMA(mxconfig);
  pinMode(BTN_LEFT, INPUT_PULLUP);
  pinMode(BTN_SOFTDROP, INPUT_PULLUP);
  pinMode(BTN_RIGHT, INPUT_PULLUP);
  pinMode(BTN_HARDDROP, INPUT_PULLUP);
  pinMode(BTN_ROTATE, INPUT_PULLUP);
  pinMode(BTN_SW, INPUT_PULLUP);
  dma_display->begin();
  dma_display->setBrightness8(80); //postavljanje razine osvjetljenosti
  dma_display->clearScreen();
  RestartGame(Board,ActivePiece,Pieces);//započimanje nove igre
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &timer_isr, true);
  timerAlarmWrite(timer, 1000000, true);
  timerAlarmEnable(timer);
```

```
}

void loop() {
  buttonState_LEFT = digitalRead(BTN_LEFT);
  buttonState_ROTATE = digitalRead(BTN_ROTATE);
  buttonState_RIGHT = digitalRead(BTN_RIGHT);
  buttonState_SOFTDROP = digitalRead(BTN_SOFTDROP);
  buttonState_HARDDROP = digitalRead(BTN_HARDDROP);
  buttonState_SW = digitalRead(BTN_SW);

  if(!pauseGame){
    if(buttonState_LEFT == 1 && lastButtonState_LEFT == 0 ||
analogRead(JS_X_AXIS)<1047){
      lastButtonState_LEFT = 1;
      moveHorizontaly(Board,ActivePiece,currentRotation,-1); //pomakni u desno
    } else if(buttonState_ROTATE == 1 && lastButtonState_ROTATE == 0){
      lastButtonState_ROTATE = 1;
      rotatePiece(Board,ActivePiece);
    } else if(buttonState_RIGHT == 1 && lastButtonState_RIGHT == 0 ||
analogRead(JS_X_AXIS)>3047){
      lastButtonState_RIGHT = 1;
      moveHorizontaly(Board,ActivePiece,currentRotation,1); //pomakni u lijevo
    } else if(buttonState_SOFTDROP == 1 && lastButtonState_SOFTDROP == 0){
      softDrop(Board,ActivePiece,currentRotation,1,pieceIndex,Pieces);
//spuštanje za jedan red
      lastButtonState_SOFTDROP = 1;
      *score_ptr=*score_ptr+1; //dodaj 1 bod
    } else if(buttonState_HARDDROP == 1 && lastButtonState_HARDDROP == 0){
      lastButtonState_HARDDROP = 1;
      hardDrop(Board,ActivePiece,currentRotation,1,pieceIndex,Pieces);
//maksimalno spuštanje
    }
    }
    if(buttonState_SW == 1 && lastButtonState_SW == 0){
      pauseGame=!pauseGame;
      lastButtonState_SW = 1;
    }

    if(buttonState_LEFT == 0){
      lastButtonState_ROTATE = 0;
    }
    if(buttonState_ROTATE == 0){
      lastButtonState_ROTATE = 0;
    }
    if(buttonState_RIGHT == 0){
      lastButtonState_RIGHT = 0;
    }
    if(buttonState_SOFTDROP == 0){
      lastButtonState_SOFTDROP = 0;
```

```
    }
    if(buttonState_HARDDROP == 0){
      lastButtonState_HARDDROP = 0;
    }
    if(buttonState_SW == 0){
      lastButtonState_SW = 0;
    }
    if(pauseGame == 1 ){
      timerWrite(timer,0);
    }
}
```