SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

SVEUČILIŠNI PREDDIPLOMSKI STUDIJ ELEKTROTEHNIKE

Sanjin Lamza

MODELIRANJE MIKROPROCESORSKIH SUSTAVA I PROGRAMSKE VEZE SA SUSTAVOM UDALJENOG LABORATORIJA U RAZVOJNOM PAKETU PROTEUS

ZAVRŠNI RAD

Rijeka, 2010.

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

SVEUČILIŠNI PREDDIPLOMSKI STUDIJ ELEKTROTEHNIKE

Sanjin Lamza

MODELIRANJE MIKROPROCESORSKIH SUSTAVA I PROGRAMSKE VEZE SA SUSTAVOM UDALJENOG LABORATORIJA U RAZVOJNOM PAKETU PROTEUS

ZAVRŠNI RAD

Student: Sanjin Lamza

JMBAG: 0069040864

Mentor: doc. dr. sc. Viktor Sučić

Kolegij: Digitalna logika

Smjer: Automatika

Rijeka, 2010

(ubaci original zadatka završnog rada)

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni preddiplomski studij elektrotehnike

I Z J A V A

U skladu s člankom 11. Pravilnika o završnom radu i završnom ispitu na preddiplomskim sveučilišnim i stručnim studijima Tehničkog fakulteta u Rijeci, izjavljujem da sam samostalno izradio završni rad prema zadatku br. 602-04/10-03/06 od 22.02.2010.

Rijeka, rujan 2010.

_____

Sanjin Lamza

Ovom prilikom htio bih se zahvaliti mojem mentoru doc. dr. sc. Viktoru Sučiću na ukazanom znanju iz teorije predmeta Digitalna logika i iznimnoj pristupačnosti tijekom izrade mojeg završnog rada. Također, želio bih se zahvaliti asistentu mr. sc. Sandiju Ljubiću na njegovom angažmanu tijekom izrade mojeg završnog rada, ali i na ukazanom znanju pri vrlo jednostavnom i motivirajućem izlaganju auditornih vježbi iz predmeta Digitalna logika.

SADRŽAJ

1. Uvod

Osnovna tema koja će biti razrađena tijekom slijedećih nekoliko poglavlja jest simuliranje različitog sklopovlja digitalne elektronike u programskom paketu Proteus. Način rada ovog programskog paketa detaljno će biti prikazan kroz simulaciju osnovnih digitalnih sklopova u njegovoj razvojnoj okolini, kao što su osnovna logička vrata, multipleksor i demultipleksor, različiti bistabili, te sekvencijski sklopovi projektirani da izvršavaju funkciju sinkronih i asinkronih brojila. Ovi će osnovni sklopovi poslužiti kao uvod u složenije sustave digitalne elektronike, a to su mikrokontrolerski sustavi. Razvoj mikrokontrolerskih sustava vrlo je složena grana digitalne elektronike, te sa jednakom ozbiljnošću kao što se mikrokontrolerskim sustavima prilazi u projektiranju njihovih aplikacija treba pristupiti i edukaciji o radu i implementaciji mikrokontrolera u pojedine aplikacije. Razvoju mikrokontrolerskih sustava može se pristupiti sa programskog i sklopovskog stajališta. Sa programskog stajališta pristupa se razvoju mikrokontrolerskih aplikacija, namijenjenih samom mikrokontroleru. Primjeri ne tako složenih mikrokontrolerskih aplikacija opisani su u četvrtom poglavlju i njih je potrebno negdje testirati. Upravo u svrhu simulacije mikrokontrolerskih aplikacija, u ovom je radu izrađen, odnosno simuliran u Proteusu, razvojni sustav za mikrokontrolerske aplikacije. Ovaj je razvojni sustav zapravo skup unaprijed spojenih perifernih komponenti s kojima korisnik može vršiti interakciju preko mikrokontrolera i opisan je u petom poglavlju, u kojem su stavljeni i prikazi simulacije mikrokontrolerskih aplikacija navedenih u četvrtom poglavlju.

U završnom dijelu ovog rada opisana je poveznica programskog paketa Proteus sa sustavom udaljenog laboratorija. Korisniku je pomoću te poveznice, odnosno aplikacije, omogućeno, uz prethodno spajanje određenih perifernih komponenti na mikrokontroler unutar Proteusa, da taj isti spoj testira u razvojnoj okolini udaljenog laboratorija.

2. Rad u programskom paketu Proteus

2.1 Proteus VSM

Proteus VSM (Virtual System Modelling), što u prijevodu znači modeliranje virtualnih sustava, programski je paket koji omogućava interaktivnu simulaciju električnih krugova u razvojnom okruženju. Modeliranje virtualnih sustava vrši se u kombinaciji simulatora analognih električnih krugova (*engl. Simulation Program with Integrated Circuit Emphasis - SPICE*), mikrokontrolerskih modela i modela interaktivnih komponenti, što omogućava i uvelike olakšava simulaciju kompletnih mikrokontrolerskih sustava. Najvažnija činjenica koja proizlazi iz Proteus VSM-a je mogućnost razvijanja i testiranja dizajna nekog sustava prije nego što je uopće napravljen fizički prototip.

Interakcija sa dizajnom ostvarena je putem različitih indikatora kao što su LED diode i LCD ekrani, te različitih upravljačkih komponenti kao što su sklopke i tipkala. Sama simulacija vrši se u realnom vremenu ili s vrlo malim odstupanjem.

Proteus VSM pruža i opsežan sadržaj za otklanjanje pogrešaka uključujući točke prekida, izvršavanje programa korak po korak, prikaz varijabli koda samog sklopa, ali i koda čiji je izvor neki programski jezik više razine.

Kao pomoć sadržaju za otklanjanje pogreški u programskom paketu Proteus nalaze se dijagnostičke poruke. One omogućavaju sagledavanje uloge određene komponente u bilo kojem trenutku simulacije i pružaju detaljne tekstualne poruke o svoj aktivnosti i međusobnoj interakciji sustava. Time je omogućeno lociranje i otklanjanje pogreški kako u programskom, tako i u hardverskom dijelu, puno brže nego kod rada na fizičkom prototipu [1].

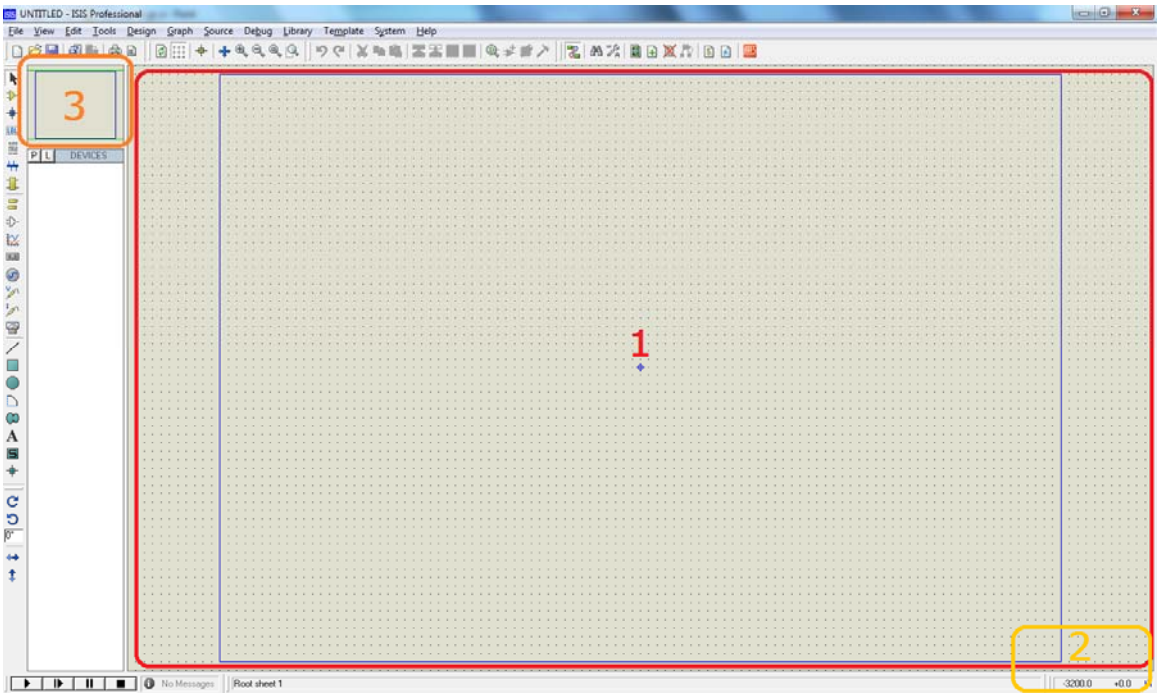
2.2 Grafičko sučelje

2.2.1 Izgled grafičkog sučelja

Najveći dio grafičkog sučelja Proteus VSM-a (Slika 2.1), naziva se *Prozor za uređivanje* i djeluje kao prozor postavljen iznad crteža. Unutar ovog prozora postavljaju se i žicama spajaju komponente. Mreža točkica u *Prozoru za uređivanje* koristi se kako bi olakšala poravnavanje komponenti i njihovo međusobno povezivanje. Izgled ove mreže može se mijenjati ili se ona može kompletno ukloniti naredbom *Grid* i njenim opcijama u *View* meniju.

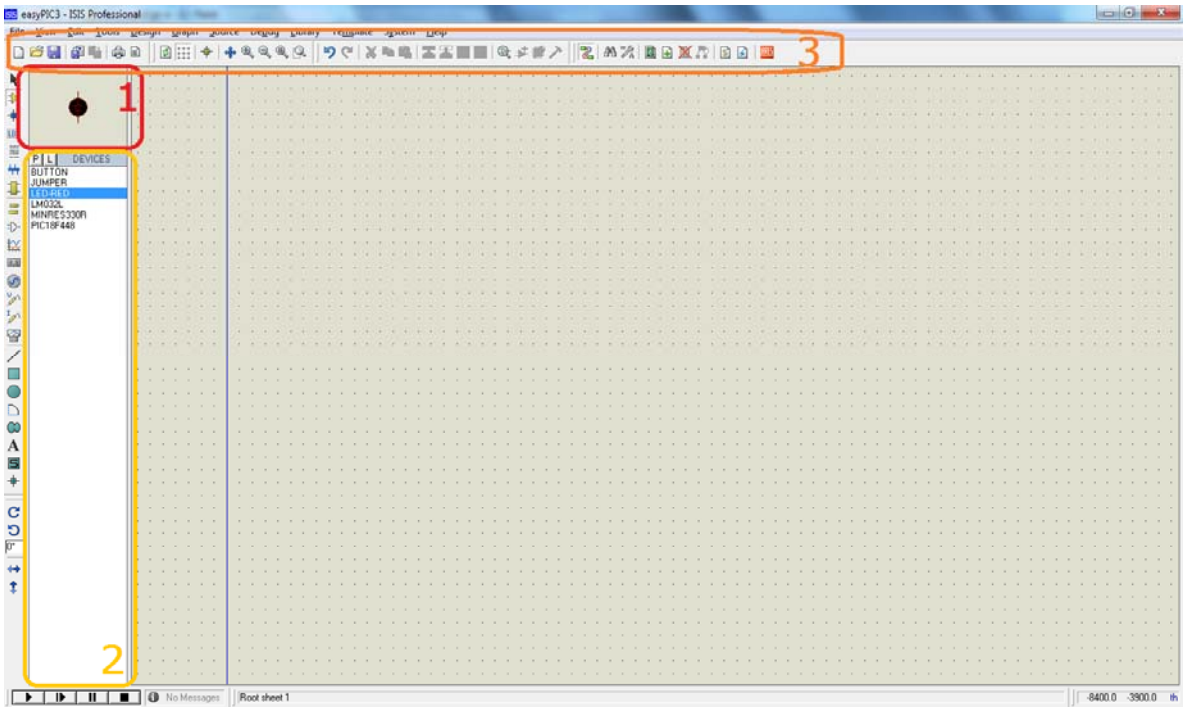
Manji dio u donjem desnom dijelu grafičkog sučelja (Slika 2.1), predstavlja prikaz koordinata na kojem su prikazane trenutne koordinate kursora kada se on nalazi iznad crteža.

U gornjem lijevom kutu grafičkog sučelja (Slika 2.1), nalazi se *Prozor za pregled*. Ukoliko je kursor trenutno fokusiran na crtežu i nije označena nijedna komponenta, *Prozor za pregled*, kao što mu samo ime kaže, služi za pregled cijelog crteža, a plavi rub na *Prozoru za uređivanje* označava dio crteža koji je obuhvaćen *Prozorom za pregled.*



Slika 2.1 Grafičko sučelje: 1) Prozor za uređivanje 2) Prikaz trenutnih koordinata kursora 3) Prozor za pregled (prikaz vidljivog područja u Prozoru za uređivanje)

Međutim, ukoliko je u *Biraču komponenti* označen neki objekt, *Prozor za pregled* koristi se za prikaz izgleda te komponente(Slika 2.2). Već spomenuti *Birač komponenti* nalazi se ispod *Prozora za pregled* (Slika 2.2), koristi se za odabir komponenti, simbola i ostalih objekata iz knjižnice Proteusa.



Slika 2.2 Grafičko sučelje: 1) Prozor za pregled (prikaz izgleda komponenti) 2) Birač komponenti 3) Alatna traka

Alatna traka Proteus VSM-a (Slika 2.2), sadrži ikone različitih funkcija od kojih su one važnije prikazane u nastavku.

 - otvaranje novog dokumenta

 - otvaranje postojećeg, već spremljenog dizajna

 - spremanje promjena u dizajnu

 - uključivanje/isključivanje mreže točkica

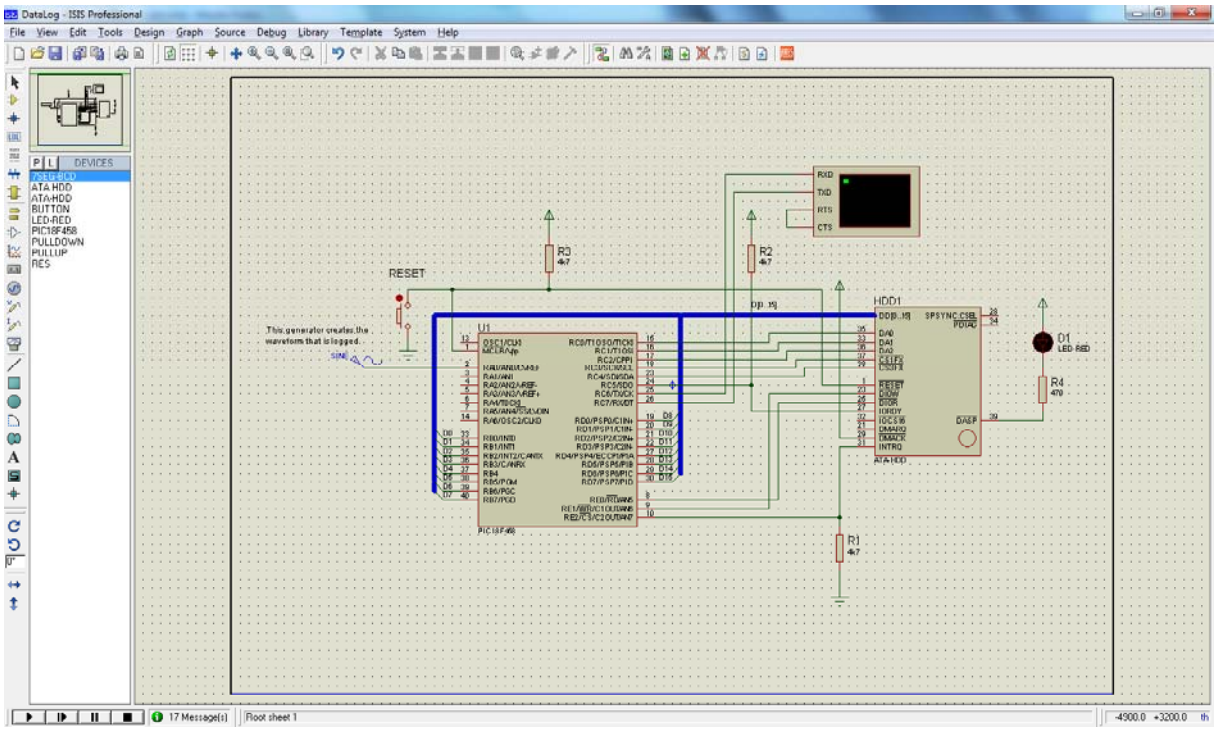
 - odabir komponenti iz odgovarajućih knjižnica

 - alat za dodavanje svojstava komponentama u dizajnu

2.2.2 ISIS

Inteligentni sustav unosa shematskih prikaza, poznatiji kao ISIS (Intelligent Schematic Input System), koristi se za crtanje i uređivanje shema i predstavlja samu jezgru Proteus programskog paketa. Ovaj sustav kombinira dizajnersku okolinu sa mogućnostima definiranja različitih aspekata izgleda shema.

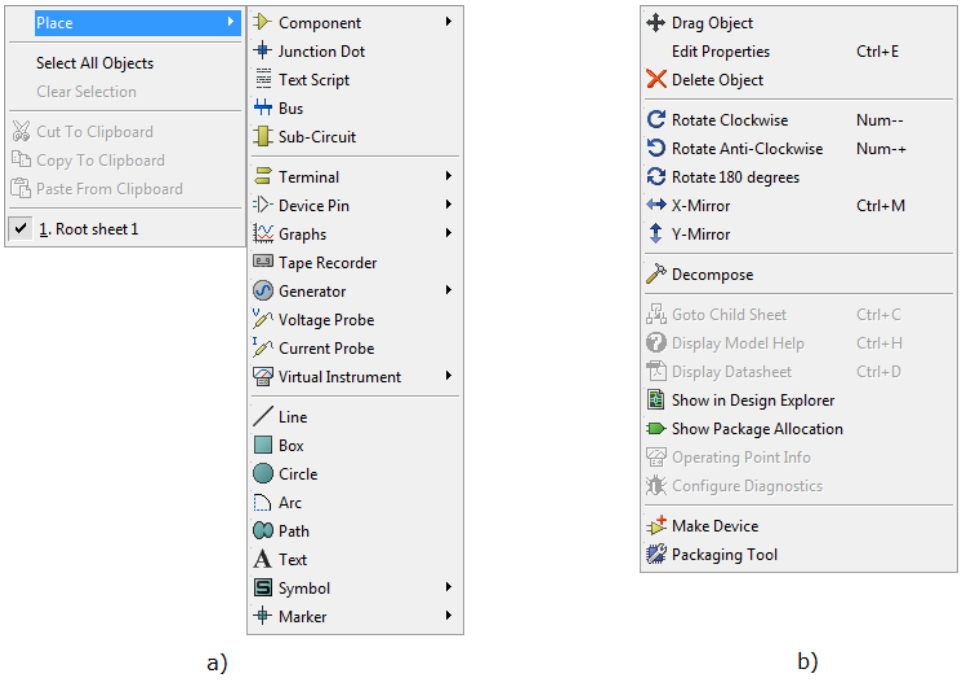
ISIS osigurava potpunu kontrolu nad mijenjanjem izgleda shema u pogledu širine linija, stilova ispunjavanja, boja i fontova i još puno različitih opcija. Izgled shema definiran je u smislu stilskog predloška, što je posebno pogodno ukoliko korisnik želi primijeniti općeniti izgled za sve svoje dizajne. Nadalje, sama shema omogućava prilagođavanje izgleda komponenti iz knjižnice Proteus programskog paketa po vlastitim potrebama ili ukusu.



Slika 2.3 Radna okolina Proteus VSM-a [S1]

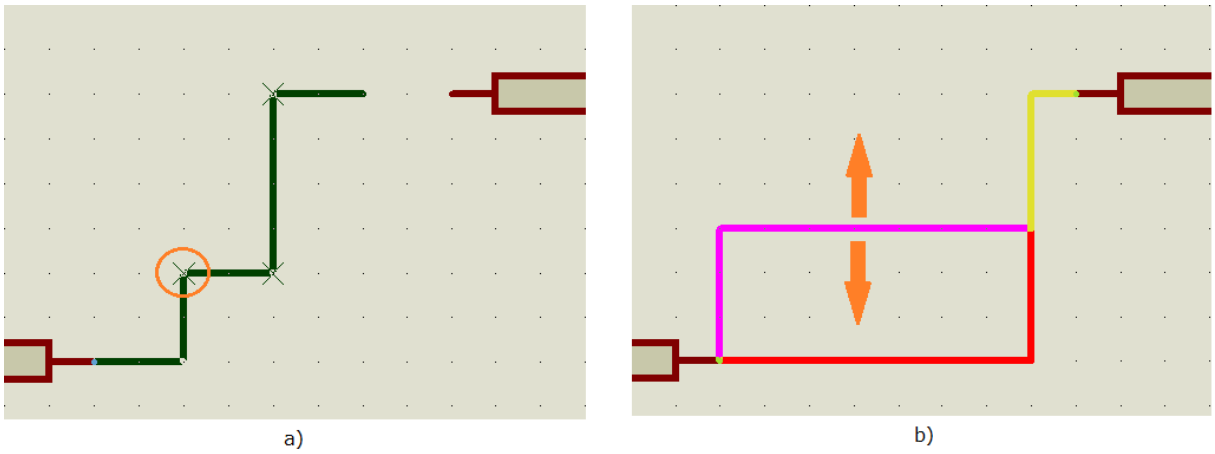
Pri izradi ISIS-a puno pažnje posvećeno je, što je moguće većem, pojednostavljenju najučestalijih radnji. Kao prvi primjer nameće se činjenica da ISIS nema posebnog moda za povezivanje komponenti žicama. Umjesto toga, korisnik može jednostavno i u bilo kojem trenutku kliknuti na izvod neke od komponenti na shemi ili na već postavljenu žicu i tako povezivati željene komponente ili žice. Naredbe za postavljanje komponenti na shemu, uređivanje,

pomicanje i brisanje mogu se izvršiti jednostavno otvaranjem posebnog menija kojem se pristupa direktno desnim klikom miša, bez potrebe traženja menija i ikona za pristupanje tim naredbama, što ISIS čini još bržim i jednostavnijim.



Slika 2.4 Izbornik nakon pritisnute desne tipke miša: a) Iznad radne površine b) Iznad komponente

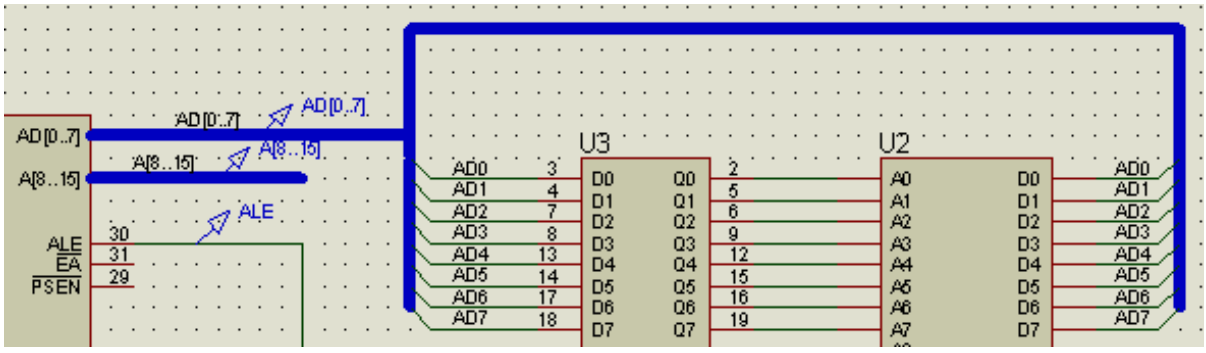
Postavljanje žica može se izvršiti jednostavno pritiskom lijeve tipke miša na dva izvoda koja se želi povezati, dok ostali posao obavlja automatski usmjerivač žica (*engl. Wire Auto-Router*). Ukoliko postoji potreba za postavljanjem žice na točno određeno mjesto, pritiskom lijeve tipke miša po površini sheme žica se može usmjeravati do željenog izvoda ili pomicati naknadno nakon spajanja sa krajnjim izvodom. Automatski usmjerivač žica koristan je i kod pomicanja povezanih komponenti, gdje on automatski popravlja položaj žice obuhvaćene premještanjem komponenti.



Slika 2.5 a) Promjena smjera žice klikom na radnu površinu b) Pomicanje žice nakon što su komponente povezane

Osim što podržava normalne dizajne preko više radnih površina (ekvivalentne sustavu koji je proširen na više komada papira), ISIS ima potpunu podršku za hijerarhijsko uređenje unutar dizajna. Drugim riječima, određena komponenta može biti definirana kao modul, koji je zatim predstavljen daljnjom shemom, odnosno podsustavom. Hijerarhija može biti raširena preko nekoliko razina, a hijerarhijski moduli mogu biti nacrtani kao standardne komponente ili kao posebni pod-sustavni blokovi na koje se naknadno mogu dodavati izvodi koji povezuju podsustave.

Stvar koja ISIS čini idealnim za mikrokontrolerske dizajne je podrška za sabirničke vodove, terminale, priključke modula i izvode različitih komponenti. Kod kompliciranijih mikrokontrolerskih sustava neki od komponenti mogu imati preko 400 fizičkih izvoda. Eventualno mukotrpno uređivanje sabirnica izbjegnuto je zasebnim crtanjem svake adrese i podatkovne sabirnice kao jedan izvod. Izvodi sabirnica omogućuju i sabirničko povezivanje hijerarhijskih modula.



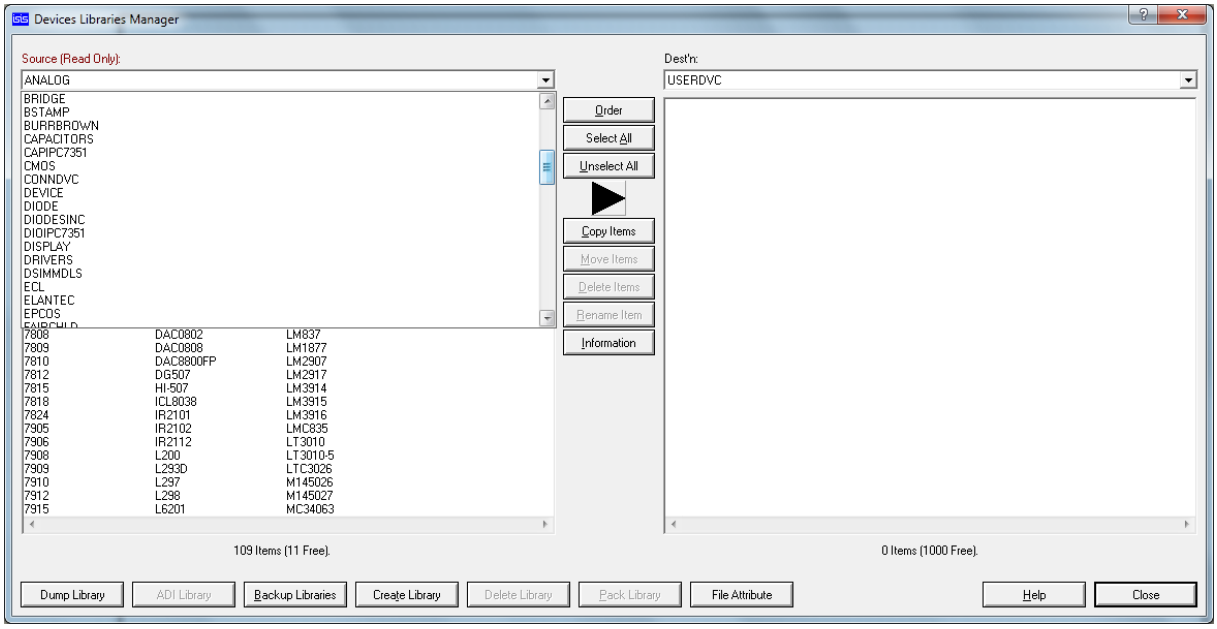
Slika 2.6 Primjer povezivanja komponenti pomoću sabirnice

U Programskom paketu Proteus, točnije unutar ISIS-a, nalaze se i njegove knjižnice simbola i komponenata. ISIS sadrži dvije knjižnice simbola i preko 25 knjižnica komponenti od kojih su u nastavku nabrojane one koje su najčešće korištene:

- knjižnica za komponente tranzistorsko-tranzistorske logike (TTL)
- knjižnica za CMOS komponente
- knjižnica za komponente ECL logičke grupe
- knjižnica mikrokontrolera
- knjižnica memorijskih komponenti
- knjižnica analognih integriranih krugova

Od ostalih knjižnica treba spomenuti knjižnice sa stotinama različitih vrsta dioda, bipolarnih i FET tranzistora i ostalih poluvodičkih komponenti, ali i one knjižnice koje sadrže komponente specifičnih proizvođača.

Komponente unutar knjižnica mogu se u bilo kojem trenutku uređivati po vlastitim potrebama pomoću različitih alata za uređivanje. Nove komponente mogu se izrađivati pomoću linija, okvira, krugova, lukova, teksta i specijalnih izvoda. Neelektrični simboli mogu se napraviti za uporabu u različitim mehaničkim i blok dijagramima [2].

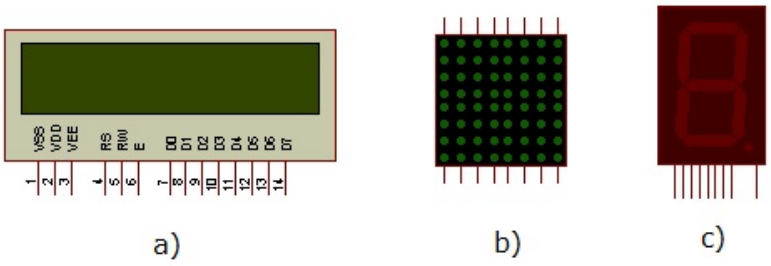


Slika 2.7 Menadžer knjižnice komponenti

2.3 Simulacija

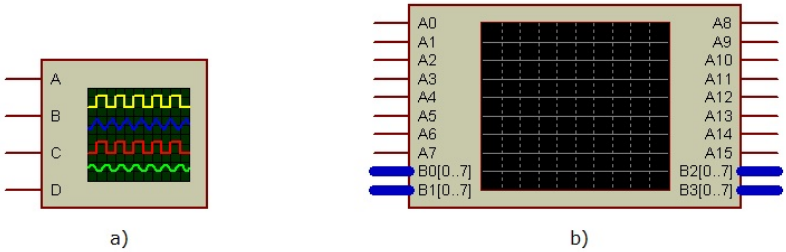
Simulacija u Proteus VSM paketu vrši se u realnom vremenu ili sa vrlo malim odstupanjem pomoću SPICE-a. SPICE (Simulation Program with Integrated Circuit Emphasis) je simulator električnih krugova koji se koristi u dizajniranju integriranih krugova i tiskanih pločica kako bi provjeravao integritet dizajna električnih krugova i predvidio ponašanje električnog kruga.

Proteus VSM podržava interaktivnu simulaciju i onu temeljenu na grafovima, a pored osnovnih komponenti nudi i različite grafičke predmete koji se mogu postaviti na shemu da bi se omogućila simulacija različitih vremenskih, frekvencijskih i ostalih varijabli. Pri simulaciji se može koristiti veliki broj interaktivnih perifernih komponenti poput LED i LCD ekrana, 7-segmentnih ekrana, univerzalnih tipkovnica i cijele knjižnice sklopki, generatora zvuka, LED dioda i ostalih interaktivnih predmeta.



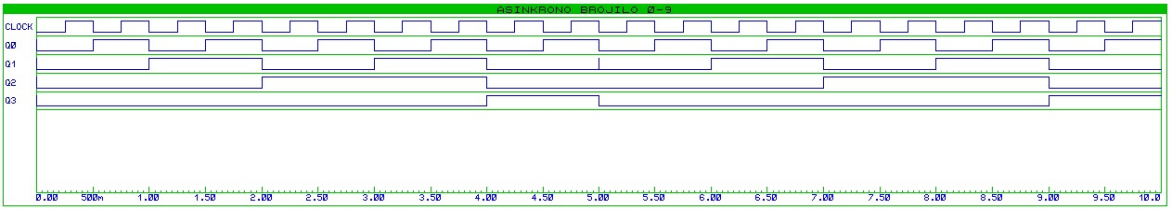
Slika 2.8 a) 16x2 LCD ekran b) 8x8 LED ekran c) 7-segmentni ekran

Kako bi uspješno mjerili, odnosno pratili određene signale ili pak stanja određenih varijabli, pri simulaciji je moguće koristiti virtualne osciloskope, logičke analizatore, VSM brojač vremena za mjerenje vremenskih intervala i one jednostavnije virtualne mjerne instrumente poput ampermetra i voltmetra. Moguće je generirati digitalne i analogne signale pomoću virtualnih funkcijskih generatora ili generatora korisnički određenog uzorka signala.



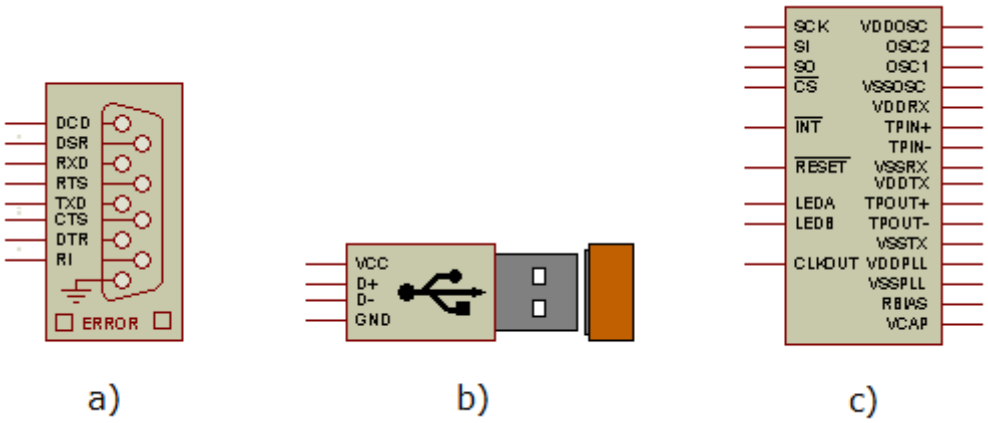
Slika 2.9 a) 4-kanalni osciloskop b) 24-kanalni logički analizator

Grafovi se mogu stavljati na shemu poput bilo kojeg drugog predmeta, kako bi se na njima iscrtao željeni signal. Tijekom pokretanja svake simulacije grafovi će iscrtati jedan kompletan ciklus promatranog signala. Ukoliko korisnik iz nekog razloga ne želi pokrenuti simulaciju cijelog sklopa, desnim klikom miša na graf može pokrenuti samostalnu simulaciju tog grafa bez simulacije ostatka sheme. Na grafovima se mogu prikazati analize prijelaznih pojava, frekvencije, buke, smetnji, AC i DC uzorci i Fourierova transformacija. Audio graf omogućuje reprodukciju simuliranih valnih oblika.



Slika 2.10 Digitalni signal prikazan na grafu

Sve navedene komponente rade u virtualnom okruženju Proteus VSM-a, njih je moguće povezati sa realnim komponentama, čime je omogućeno simuliranje električnih krugova čiji su dijelovi realne periferne komponente. Time su uvelike proširene mogućnosti Proteus programskog paketa jer to znači da je moguće mjeriti realne varijable pomoću različitih senzora i ostvariti komunikaciju sa realnim uređajima. Veza virtualnog i realnog okruženja ostvaruje se pomoću virtualnih komponenti koje zasebno predstavljaju realne priključke za RS232 serijsku komunikaciju, priključak za USB komunikaciju i priključak za ethernet komunikaciju.



Slika 2.11 a) Komponenta za RS232 serijsku komunikaciju b) Komponenta za USB komunikaciju c) Komponenta za ethernet komunikaciju

3. Simulacija osnovnih digitalnih (kombinacijskih i sekvencijskih) modula

3.1 Logička vrata

Logička vrata izvode logičke operacije za jedan ili više logičkih ulaznih signala i zatim daju jedan izlazni signal. Logika ovih vrata izvodi se u binarnom sustavu i najčešće ih se nalazi u digitalnoj logici. Logička vrata obično se izvode elektronički pomoću dioda i tranzistora iako postoje i druge izvedbe. U elektroničkoj logici logički nivo predstavljen je padom napona. Viši napon označava binarnu vrijednost 1, dok onaj niži označava binarnu vrijednost 0, a točna vrijednost tih napona ovisi o izvedbi elektroničke logike koja se koristi.

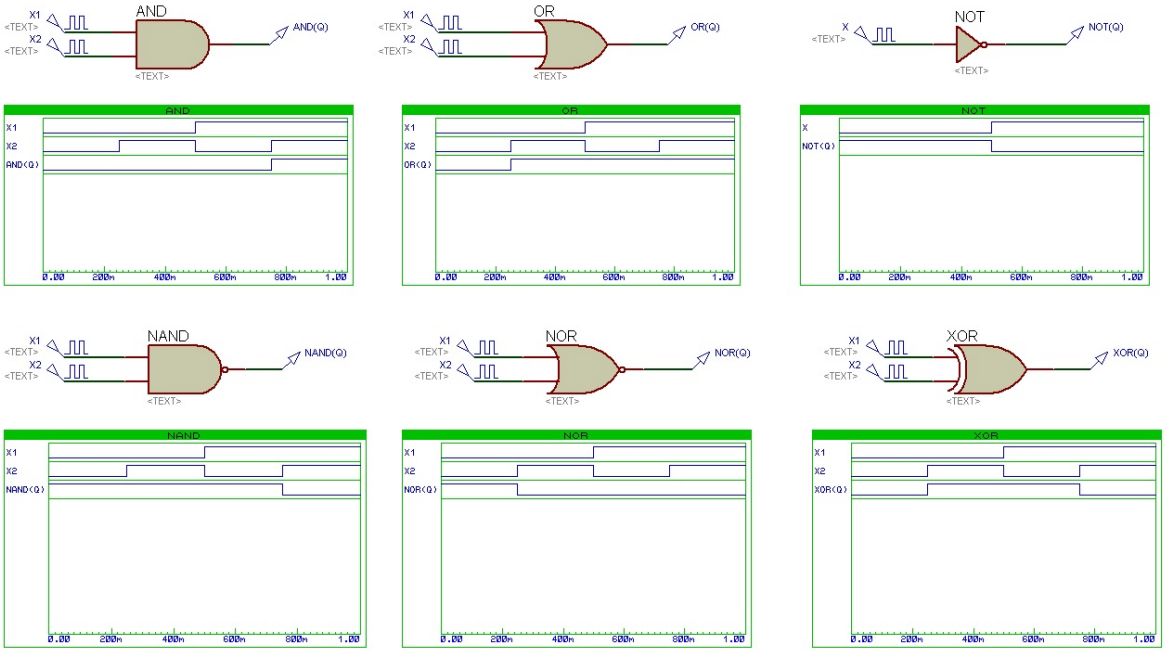
Pod osnovna logička vrata svrstavamo I (*engl. AND*), ILI (*engl. OR*), NE (*engl. NOT*) ,NI (*engl. NAND*), NILI (*engl. NOR*) i XILI (*engl. XOR*) logička vrata. Svaka od ovih logičkih vrata imaju dva ulaza i jedan izlaz, osim NE logičkih vrata koja imaju jedan ulaz i jedan izlaz.

Osnovna logička vrata simulirana su u Proteusu (Slika 3.1). Na ulaze svake od ovih logičkih vrata, pomoću funkcijskih generatora, dovedeni su digitalni signali frekvencije 1Hz. Izlazni signal koji je rezultat logičke operacije pojedinih logičkih vrata, mjeri se pomoću sonde za mjerenje napona. Za svaka osnovna logička vrata zasebno je iscrtan graf mogućih stanja (Slika 3.1), kako bi se prikazala logička operacija koju ista vrše. Slovom Q označeni su izlazni signali. Ulazni signal u slučaju NE logičkih označen je slovom X jer ona imaju samo jedan ulaz, dok se kod svih ostalih logičkih vrata nalaze dva ulaza pa su ona označena sa X1 i X2.

Iz grafa I logičkih vrata vidi se da će ona na izlazu imati logičko stanje 1 samo u slučaju da su oba ulaza u logičkom stanju 1, dok u ostalim slučajevima imaju logičko stanje 0. Kod grafa ILI logičkih vrata se pak vidi da će ona dati na izlazu logičku 0 samo u slučaju da su oba ulaza u stanju logičke 0, dok će u ostalim slučajevima imati logičko stanje 1. NE logička vrata imaju samo jedan ulaz, a na njihovom izlazu biti će suprotno logičko stanje u odnosu na ulaz jer ona djeluju kao inverter. Graf NI logičkih vrata pokazuje da će ona imati na izlazu logičku 0 samo u trenutku kada su oba ulaza jednaka logičkoj 1, u svim ostalim slučajevima izlaz je jednak logičkoj 1. NILI logička vrata na izlazu će dati logičku 1 samo kada su oba ulaza jednaka nuli, u ostalim slučajevima izlaz je jednak

logičkoj 0. EKSILI logička vrata, kako pokazuje njihov graf, na izlazu daju logičku 0 ukoliko oba ulaza istog logičkog stanja, a logičku 1 daju ukoliko su ulazi različitog logičkog stanja.

Na ovim grafovima vidi se kompletan ciklus promjene stanja pojedinih logičkih vrata, ali je on zamrznut u vremenu. Promjene stanja u realnom vremenu tijekom trajanja simulacije korisnik može vidjeti u obliku točki koje se tijekom simulacije mijenjaju iz plave u crvenu boju i obrnuto, ovisno o trenutnom stanju(Slika 3.1). Crvena boja u ovom slučaju predstavlja logičku jedinicu, dok plava boja predstavlja logičku nulu.



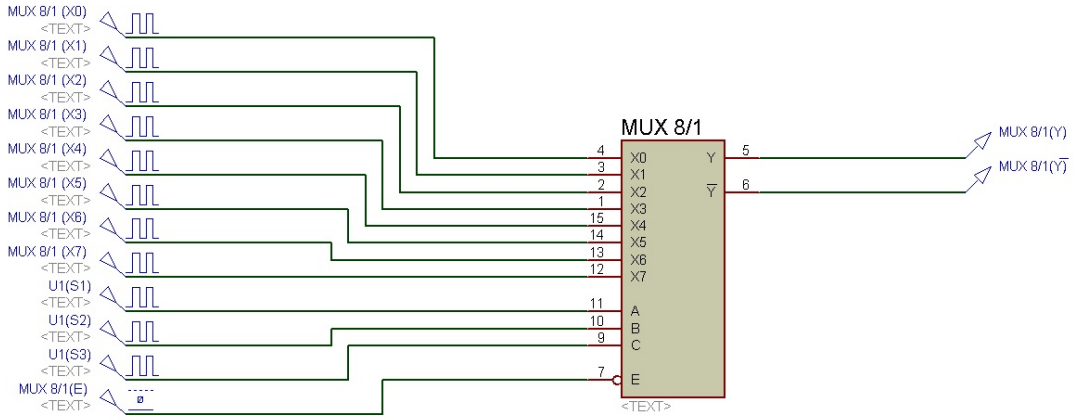
Slika 3.1 Osnovna logička vrata

3.2 Multipleksor 8/1

Mutipleksor ili skraćeno Mux (*engl. Multiplexer*), logički je sklop koji izvodi multipleksiranje signala. On odabire jedan ili više analognih ili digitalnih signala na ulazu i upućuje ga na izlaz. Multipleksor sa 2^n ulaza ima n selektivnih ulaza, kojima se odabire koji će od ulaznih signala biti proslijeđen na izlaz.

Proteus VSM sadrži više izvedbi multipleksora dostupnih za simulaciju. Za ovaj primjer odabran je multipleksor 8/1 (Slika 3.2). Brojka 8 u njegovom imenu označava 8 podatkovnih ulaza, koji su na komponenti označeni sa X0-X7. Ovaj multipleksor ima 3 selektivna ulaza, jer multipleksor koji ima 8 ulaza, što je jednako 2^3, prema već navedenoj relaciji mora imati 3 selektivna ulaza, u ovom primjeru označena sa A, B i C.

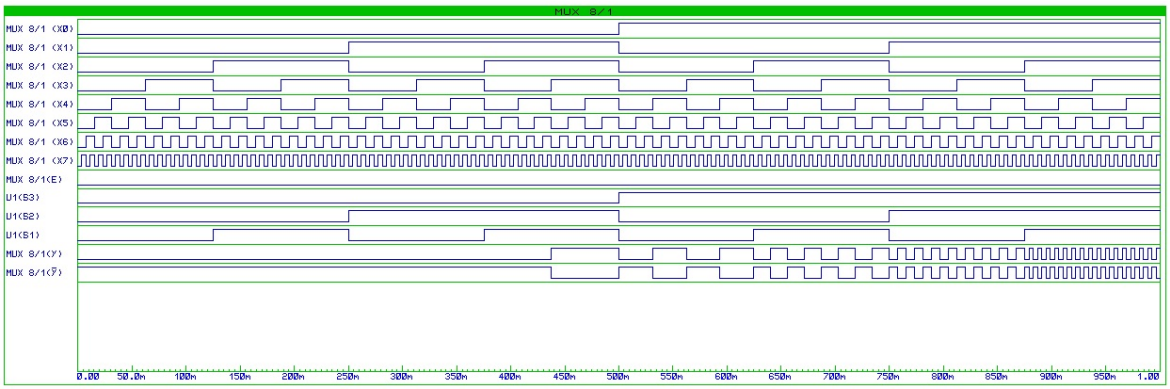
Ova komponenta sadrži i E ulaz na koji se dovodi *Enable* bit koji određuje u kojem će trenutku multipleksor biti aktivan, kružić na komponenti označava da je ovaj multipleksor aktivan kada je *Enable* bit jednak 0. Na ulaz E tako je funkcijskim generatorom doveden digitalni signal koji je cijelo vrijeme jednak 0 kako bi multipleksor bio aktivan. Na desnoj strani ove komponente prikazana su dva izlaza. Slovom Y označen je izlazni signal, a sa $\overline{Y}$ označen je njegov komplementirani ekvivalent.



Slika 3.2 Multipleksor simuliran u Proteus VSM-u

Na podatkovne ulaze multipleksora postavljeni su funkcijski generatori digitalnih signala. Na grafu (Slika 3.3) su vidljivi ulazni podatkovni signali čije su frekvencije jednake broju 2 potenciranom brojem uz slovo X u njihovom imenu. Dakle, frekvencija prvog ulaznog podatkovnog signala na komponenti, gledavši od gore prema dolje, biti će 2^0, odnosno 1 Hz, dok će frekvencija onog zadnjeg iznositi 2^7, odnosno 128 Hz.

Signali na grafu označeni sa S1, S2 i S3 predstavljaju selektivne ulazne signale A, B i C. Sva tri signala zajedno čine binarne kombinacije kojima se određuje koji će se signal ulazni podatkovni signal proslijediti na izlaz. Njihove kombinacije mijenjaju se od 000 do 111, što znači da se u dekadskom ekvivalentu mijenjaju od 0 do 7, pa se sukladno tome na izlaz prosljeđuju ulazni podatkovni signali od X1 do X7. Promatra li se na grafu izlazni signal Y, u odnosu na promjenu stanja selektivnih ulaza S1,S2 i S3, vidi se da se frekvencija izlaznog signala Y mijenja sa promjenama selektivnih signala, te se upravo po toj frekvenciji može lako zaključiti koji je ulazni podatkovni signal u određenom trenutku proslijeđen na izlaz.

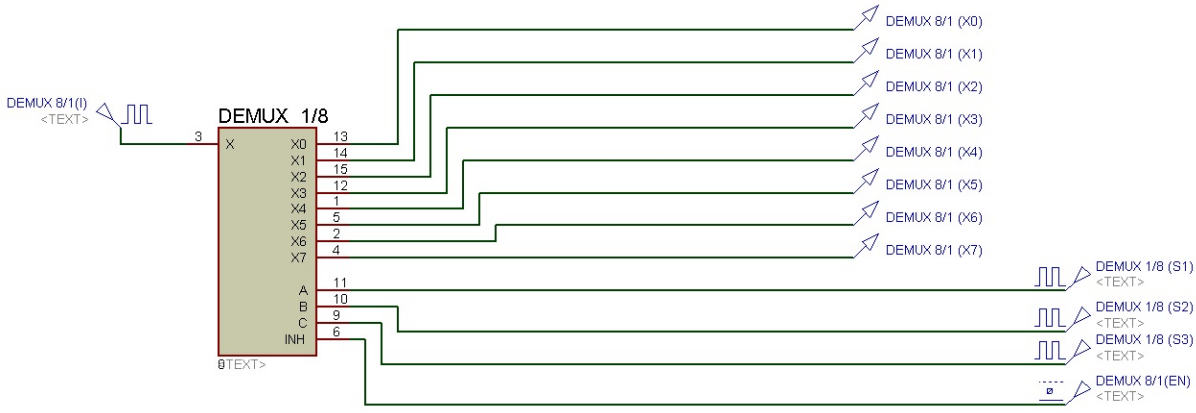


Slika 3.3 Prikaz izlaznih signala i podatkovnih i selektivnih ulaznih signala multipleksora

3.3 Demultipleksor 1/8

Demultipleksor ili Demux (engl. *Demultiplexer*) logički je sklop koji ima obrnutu funkciju u odnosu na multipleksor. On ima jedan podatkovni ulaz, a u odnosu na n selektivnih ulaza ima 2^n izlaza. Trenutno stanje njegovih selektivnih ulaza označava na koji će izlaz u tom trenutku biti proslijeđen ulazni podatkovni signal.

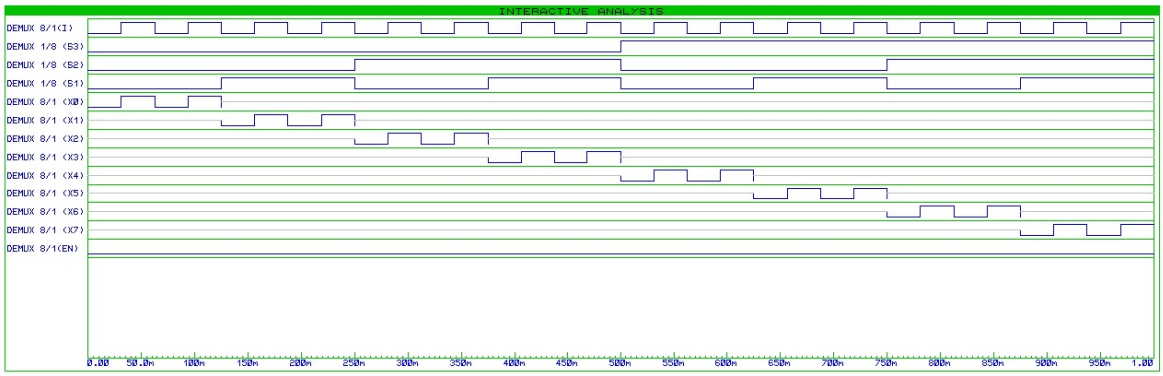
Proteus VSM sadrži više izvedbi demultipleksora, a radi sličnosti sa primjerom multipleksora odabran je demultipleksor 1/8 (Slika 3.4). Njegov podatkovni ulaz na komponenti je označen samo slovom X, selektivni ulazi označeni su sa A, B i C i u ovom slučaju nalaze se sa desne strane komponente. Izlazi su označeni oznakama X0-X7. Signal koji određuje je li demultipleksor aktivan ili neaktivan, *Enable*, označen je na komponenti sa INH. Kao i prikazani multipleksor, ovaj demultipleksor ima aktivno stanje kada je na *Enable* signal jednak 0, pa je na ovaj ulaz funkcijskim generatorom i dovedena signal koji cijelo vrijeme ima vrijednost 0.



Slika 3.4 Demultipleksor simuliran u Proteus VSM-u

Na ulaz X doveden je, pomoću funkcijskog generatora, digitalni signal proizvoljne frekvencije 16 Hz(Slika 3.5). Podatkovni signali također su proizvoljne frekvencije 4 Hz, a njihova međusobna kombinacija u nekom trenutku određuje na koji će od izlaza biti proslijeđen ulazni podatkovni signal u tom nekom trenutku. Kombinacije se mijenjaju od 000 do 111, što u dekadskom brojevnom sustavu označava promjenu od 0 do 7. Sukladno tome, mijenja i se izlaz na koji je proslijeđen ulazni podatkovni signal od X0-X7. Dakle u početnom trenutku na grafu (Slika 3.5), kada je stanje selektivnih ulaza 000, ulazni signal proslijeđen je na izlaz X0, dok zadnja kombinacija 111 označava da se podatkovni ulaz prosljeđuje na izlaz X7. Iz grafa je vidljivo da se svakom slijedećom promjenom kombinacije selektivnih ulaza, ulazni signal pomiče na slijedeći po redu izlaz.

U trenucima kada na određeni izlaz nije doveden ulazni podatkovni signal , njegova vrijednost je plutajuća (*engl. Floating*), što je na grafu označeno sivom crtom.



Slika 3.5 Prikaz izlaznih signala i podatkovnih i selektivnih ulaznih signala demultipleksora

3.4 Bistabili

U digitalnoj logici, naziv bistabil (*engl. Flip-flop*), označava sekvencijalne sklopove digitalne elektronike koji imaju dva stabilna stanja, pa ih je zato moguće koristiti za memoriranje 1 bita. Bistabil može imati jedan ili više ulaza, a oni se označavaju ovisno o vrsti bistabila. Izlazi se označavaju sa Q (logičko stanje bistabila) i $\overline{Q}$ (logički komplement izlaza Q) i to vrijedi za sve vrste bistabila. Promjena iz jednog stabilnog stanja u drugo naziva se okidanje bistabila.

Bistabile možemo podijeliti u dvije skupine i to na asinkrone i sinkrone bistabile. Asinkroni bistabili imaju svojstvo reakcije na promjenu ulaznog impulsa, što daje pogrešan rezultat kada se impulsi ne dovode istovremeno na ulaz.

Sinkroni bistabil je verzija asinkronog sa nadodanim CLK signalom (*engl. Clock*) na koji se dovode sinkronizacijski impulsi konstantne frekvencije tako da bistabil mijenja stanja na promjenu okidnog impulsa.

Bistabili se mogu dalje podijeliti na tipove koji se primjenjuju i u asinkronim i u sinkronim sustavima. Od tih tipova najčešći su SR („Set-Reset“) bistabil, D („Data“) bistabil, T („Toogle“) bistabil i JK bistabil. Svaki od njih može se izvesti sa većinom ostalih bistabila i dodavanjem logičkih vrata. Ponašanje određenog tipa bistabila opisano je karakterističnom jednadžbom koja se sastoji od „slijedećeg“ stanja izlaza Q_{n+1} (stanje nakon slijedeće promjene *Clock* signala) i sadašnjeg stanja Q_n.

3.4.1 SR bistabil

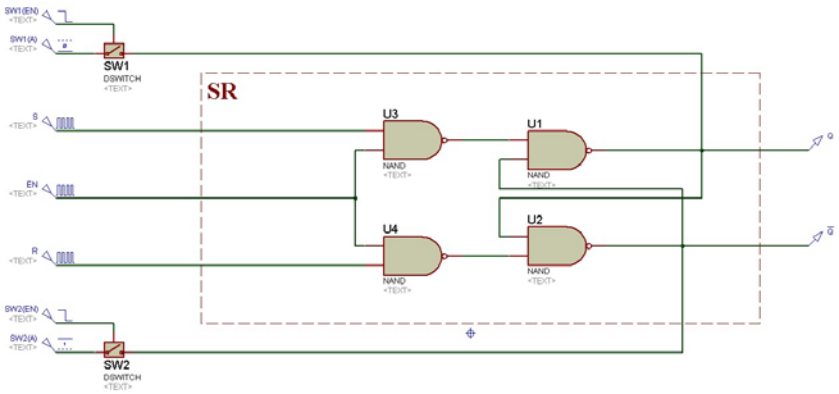
SR bistabil ima dva ulaza, S i R po kojima je i dobio ime. Ulaz S naziva se Set (*engl. Set,* postaviti) ulaz, dok se ulaz R naziva Reset (*engl. reset*, poništiti). Uobičajeno, u modu za čuvanje stanja, S i R imaju stanje logičke 0, a izlazi Q i $\overline{Q}$ ostaju nepromijenjeni. U slučaju da S ima stanje logičko stanje 1 dok R ima stanje 0, izlaz Q bit će prisiljen imat logičko stanje 1. To stanje ostat će nepromijenjeno čak i ako se ulaz S vrati na stanje logičke 0. Slično tome, ukoliko R ulaz ima stanje 1, dok ulaz S ima stanje 0, izlaz Q biti će prisiljen biti u stanju 0 i ostat će u tom stanju i ako se R vrati na stanje 0. U trenutku kada S i R imaju stanje 1 dolazi do zabranjenog stanja i u tom trenutku ne može se predvidjeti stanje Q izlaza. Ovaj bistabil može biti izveden od jednog para ukršteno povezanih NI ili NILI logičkih vrata.

SR bistabil može se opisati jednadžbom stanja (3.1), gdje Q_n predstavlja sadašnje stanje bistabila. Q_{n+1} postaje Q_n nakon slijedećeg brida *Clock* signala. Uvođenjem dodatne jednadžbe SR=0 uklanja se mogućnost pojave zabranjenog stanja.

$$Q_{n+1} = S + \overline{R}Q_n \qquad (\mathbf{3.1})$$

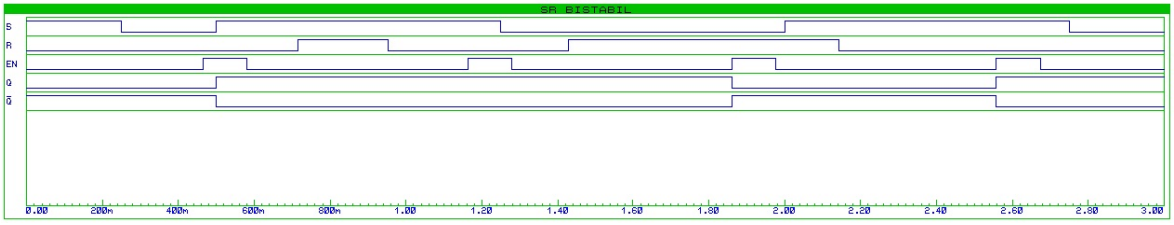
SR bistabil, prikazan i simuliran u Proteus VSM-u (Slika 3.6), izveden je od četiri primjerka NI logičkih vrata. Na njegov S i R ulaz dovedeni su signali proizvoljne frekvencije. Signal EN isto je proizvoljne frekvencije i ukoliko je njegovo

stanje 1 ovaj bistabil je u aktivnom području, on sinkronizira rad ovog sklopa. Dvije idealne digitalne sklopke SW1 i SW2 stavljene su da bi se moglo postaviti stanje Q i $\overline{Q}$ u početnom trenutku simulacije. Početna stanja postignuta su tako da je na kontrolni signal sklopke doveden koji ju čini propusnom samo u početnom trenutku, te se samo tad propusti signal početnog stanja koji u slučaju Q iznosi 0, dok je kod $\overline{Q}$ on jednak 1.



Slika 3.6 SR bistabil

S, R, EN, Q i $\overline{Q}$ signali ovog bistabila prikazani su na grafu (Slika 3.7). Vidljivo je početno stanje izlaza Q koje iznosi 0. Signali S i R mijenjaju se tokom cijele simulacije, no njihovo stanje bitno je samo ukoliko je EN signal (*Clock* signal) jednak 1 jer je samo tada bistabil u aktivnom području rada. U trenutku kada je EN signal jednak 1, signal S ima vrijednost 1, a signal R ima vrijednost 0. Iz tog razloga stanje izlaznog signala Q mijenja se iz stanja 0 u stanje 1. Slijedeći put kada je EN signal jednak 1, signal S se mijenja iz stanja 1 u stanje 0, dok je stanje R ulaza ostalo nepromijenjeno i tada se vidi da je izlazni signal Q ostao nepromijenjen. R signal, kod slijedećeg stanja 1 EN signala, ima vrijednost 1, a signal S ima stanje 0 pa je ulazni signal prisiljen imati vrijednost 0, što se iz grafa i vidi.



Slika 3.7 Graf SR bistabila

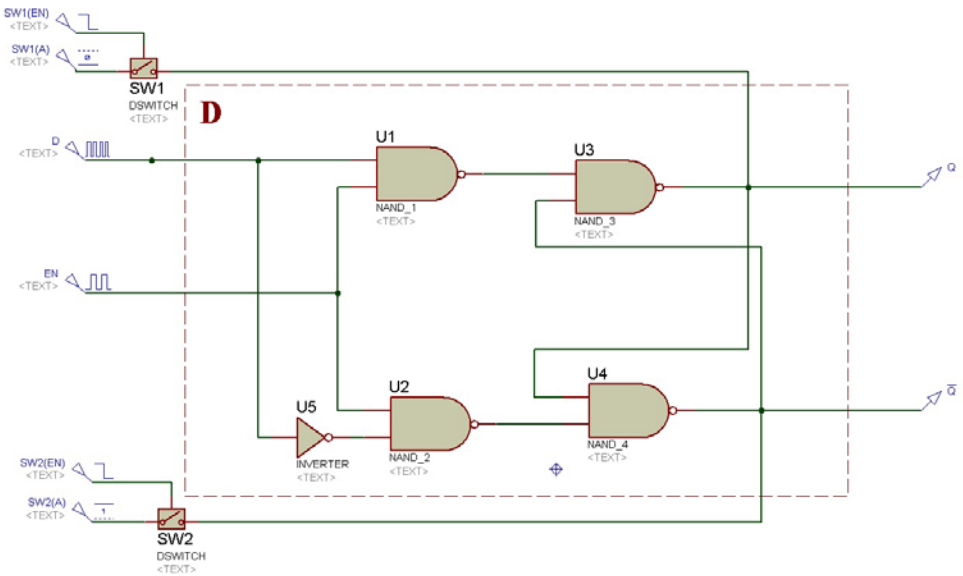
3.4.2 D bistabil

D bistabil je modifikacija SR bistabila koja je dobivena tako da se ulazna varijabla D spoji direktno na S ulaz, a na ulaz R se pomoću NE vrata dovede komplementirana vrijednost ulaza S, što znači da ovaj bistabil ima samo jedan podatkovni ulaz. Ovaj bistabil jednostavno na izlazu daje podatak koji mu je dan na ulazu, što opisuje njegova karakteristična jednadžba (3.2). Iz tog razloga može ga se gledat kao sklop za memoriranje jednog bita ili kao sklop za kašnjenje ukoliko se priključi i *Clock* ulazni signal. Upravo zbog svojih svojstava (*engl. Data,* podatak i *engl. Delay,* kašnjenje) ovaj je bistabil i dobio ime D bistabil.

$$Q_{n+1} = D \tag{3.2}$$

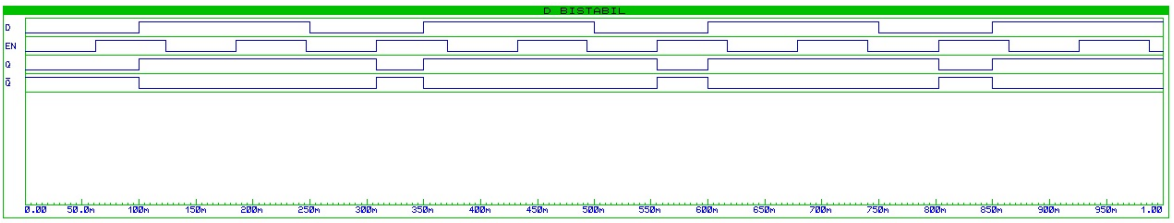
Proteus VSM nudi nam gotove komponente D bistabila, ali pri ovoj simulaciji D bistabila korištena je modificirana verzija SR bistabila(Slika 3.8), kako bi se prikazala cjelokupna izvedba ovog bistabila. S ulaz sada je nazvan D ulaz i na njega se dovodi podatkovni signal proizvoljno određenog uzorka.

EN ulazni signal vrši istu funkciju kao i kod SR bistabila, sinkronizira rad ovog sklopa, što znači da će ovaj D bistabil biti aktivan samo kada je signal EN u stanju logičke 1 i jedino tada stanje izlaznog signala Q biti će jednako stanju ulaznog signala D. Idealne sklopke SW1 i SW2 i u ovom primjeru postavljaju početno stanje izlaz Q i njegovog komplementiranog izlaza $\overline{Q}$.



Slika 3.8 D bistabil

Na grafu (Slika 3.9) su prikazani ulazni signal D, ulazni signal EN i izlaz Q, te njegova komplementirana komponenta $\overline{Q}$. Signali D, EN i Q imaju početno stanje 0. U vremenu dok je EN signal jednak 1, D bistabil je u aktivnom stanju, pa će tako u svim vremenskim intervalima kada je EN u logičkom stanju 1 izlaz Q pratiti vrijednost ulaza D. Pri završetku svakog od tih intervala izlaz Q zadržati će trenutnu vrijednost sve do slijedećeg intervala i promjene vrijednosti podatkovnog ulaza D.



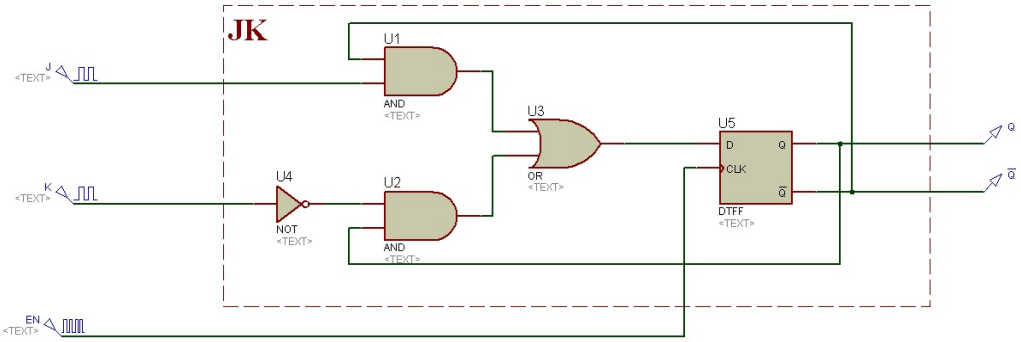
Slika 3.9 Graf D bistabila

3.4.3 JK bistabil

JK bistabil razlikuje se od SR bistabila samo po činjenici da u njegovom slučaju nema zabranjenih stanja. Kada su J i K logički istog stanja na izlazu Q daju stanja koja ovise o prethodnim stanjima bistabila, kompletno ponašanje ove vrste bistabila opisuje njegova karakteristična jednadžba (3.3). Kod JK bistabila uveden je i *Clock* signal koji služi za sinkronizaciju. Naziv ove vrste bistabila nema nikakve veze sa njegovim načinom rada. Ime su dobili po inicijalima svog autora Jacka Kilbyja.

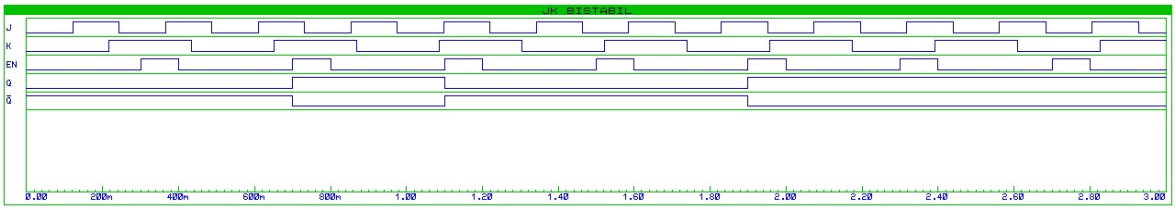
$$Q_{n+1} = J\overline{Q_n} + \overline{K}Q_n \tag{3.3}$$

Pri simulaciji JK bistabila u Proteus VSM-u korištena je prikazana shema (Slika 3.10). Kombinacija komponente D bistabila i osnovnih logičkih vrata izvedena je da tvori već spomenutu karakterističnu jednadžbu JK bistabila (3.3). Na J i K ulaze dovedeni su signali različite proizvoljne frekvencije, a signal EN je doveden na *Clock* ulaz kako bi određivao aktivno područje, odnosno sinkronizirao rad D bistabila unutar sklopa, a time i cijelog sklopa. Signal EN također je proizvoljne frekvencije.



Slika 3.10 JK bistabil

Na grafu (Slika 3.11) vidi se karakteristično ponašanje signala ovog bistabila. Svi ulazi i Q izlaz imaju početno logičko stanje 0. Stanje izlaza mijenja se pri uzlaznoj promjeni signala EN, što u ovom slučaju određuje komponenta D bistabila koja je izvedena da reagira na uzlazni brid *Clock* signala. Kada *Clock* signal EN promjeni stanje u logički jedan, dakle, pri prvom uzlaznom bridu tog signala, signal Q ostaje nepromijenjen jer je u tom vremenu signal K jednak 1,a signal J jednak 0 pa je izlaz Q prisiljen imat vrijednost 0. Pri slijedećem uzlaznom bridu *Clock* signala, izlazni signal Q promijenit će vrijednost jer su u tom trenutku ulazi J i K oba jednaki logičkom stanju 1. Pri trećem uzlaznom bridu stanje izlaza Q se opet mijenja jer su u tom trenutku ulazi J i K opet oba u logičkom stanju 1. U trenutku četvrtog uzlaznog brida signali J i K su u stanju logičke 0 pa ne dolazi do promjene izlaznog signala. U trenutku petog uzlaznog brida Clock signala vidi se četvrta, ujedno i posljednja moguća kombinacija ulaza J i K, gdje ulaz J ima stanje 1,a ulaz K stanje 0, što znači da je izlaz Q prisiljen biti u stanju logičke 1.



Slika 3.11 Graf JK bistabila

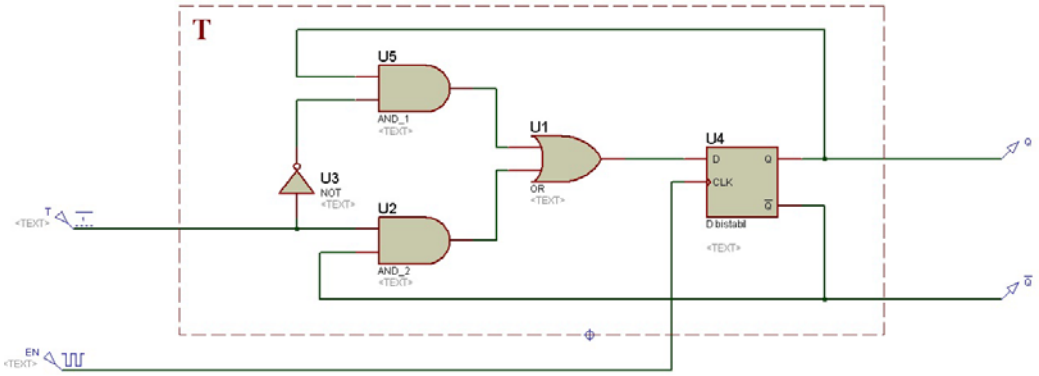
3.4.4 T bistabil

T bistabil je modificirana verzija JK bistabila. Ulaz T spojen je na K ulaz JK bistabila koji je preko NE logičkih vrata spojen na ulaz J. Isto kao i svi ostali bistabili do sad i T bistabil ima *Clock* ulaz za sinkronizaciju njegovog rada. Njegov

rad opisan je njegovom karakterističnom jednadžbom (3.4). Ukoliko je na ulaz T dovedena logička 1, dobiva se sklop koji mijenja logičko stanje na brid EN (*Clock*) signala čime se na izlazu Q dobiva signal dvostruko manje frekvencije. T bistabil dobio je ime upravo po svojstvu promjene stanja (*engl. Toogle*, prebacivati).

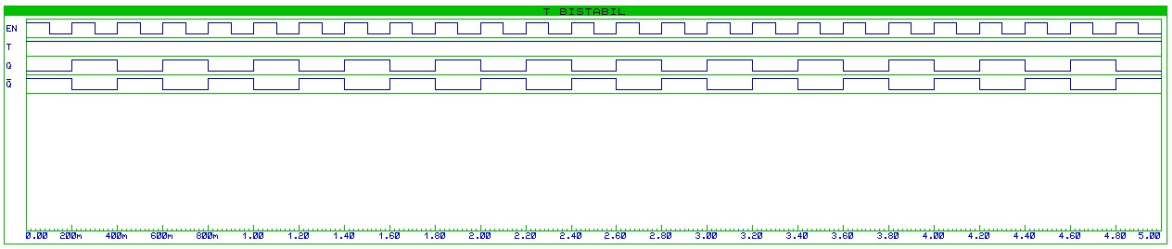
$$Q_{n+1} = T\overline{Q_n} + \overline{T}Q_n \tag{3.4}$$

Shema T bistabila, korištena u simulaciji (Slika 3.12), sastavljena je od komponente D bistabila i osnovnih logičkih vrata. Ove komponente zajedno tvore karakterističnu jednadžbu T bistabila. Na ulaz T doveden je signal koji je tijekom cijelog vremena u stanju logičke 1, na *Clock* ulaz D bistabila doveden je signal EN.



Slika 3.12 T bistabil

Graf T bistabila (Slika 3.13) pokazuje nam njegove karakteristične ulazne i izlazne signale. Tijekom cijele simulacije ulaz T je u stanju logičke 1. Kao rezultat toga na izlazu bistabila dobivamo signal Q dvostruko manje frekvencije u odnosu na *Clock* signal EN, što znači da je T bistabil moguće upotrijebiti kao dijelilo frekvencije.



Slika 3.13 Graf T bistabila

3.5 Brojila

Brojila su sekvencijski sklopovi koji se koriste u digitalnoj elektronici. Njihova uloga je pohranjivanje broja promjena koje su se dogodile unutar nekog događaja ili procesa, često u vezi sa *Clock* signalom. U praksi su brojila podijeljena na ona koja povećavaju neku vrijednost i ona koja neku vrijednost smanjuju. Ona mogu biti izvedena prilično jednostavno koristeći bistabile. Broj bistabila unutar brojila označava njegov maksimalni brojevni ciklus. Dakle, brojilo koje se sastoji od n bistabila imati će maksimalni brojevni ciklus od broja 0 do broja 2^n-1. Brojevni ciklus brojila može se ograničiti po potrebi dodavanjem dodatnih logičkih vrata kako bi se preko njih vratili željeni izlazni signali bistabila na Reset ulaze bistabila i time ih poništili, nakon čega bi cijeli ciklus brojanja započeo ispočetka. Po načinu spajanja *Clock* signala na ulaze bistabila razlikuju se sinkroni bistabili koji međusobno imaju zajednički *Clock* signal i oni asinkroni koji maju međusobno različite *Clock* signale.

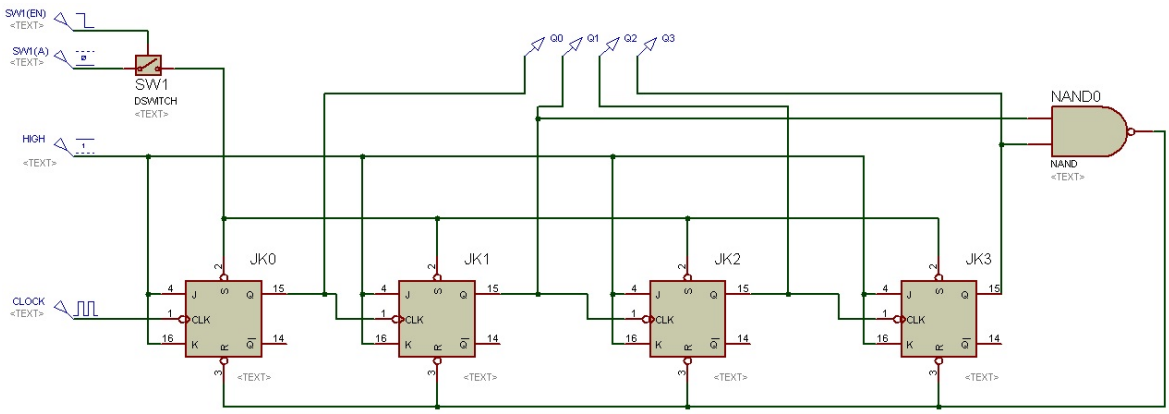
3.5.1 Asinkrono brojilo 0-9

Asinkrona brojila mogu biti izvedena od različitog broja bistabila, na koje su dovedeni međusobno različiti *Clock* signali. Njihov maksimalni brojevni ciklus ovisi, kako je već rečeno, o broju bistabila unutar njih, a taj ciklus se ograničava dovođenjem određene kombinacije izlaznih signala na *Reset* ulaze preko logičkih vrata kako bi se poništilo stanje bistabila, čime bi brojanje krenulo ispočetka.

Asinkrono brojilo simulirano u Proteus VSM-u (Slika 3.14) izvedeno je pomoću četiri JK bistabila, koji su pokrenuti dovođenjem logičke 0 u početnom trenutku simulacije pomoću idealne digitalne sklopke SW1. Signali Q0, Q1, Q2 i Q3 predstavljaju izlazne signale svakog od bistabila. Zajedno, njihove kombinacije predstavljaju binarni zapis brojevnog stanja bistabila. Signal Q0 predstavlja bit najmanje težine i njegova frekvencija je najveća. Da bi brojanje bilo jedinično po dekadskom brojevnom sustavu svaki slijedeći izlazni signal, počevši od Q0, mora imati dvostruko manju frekvenciju. Kako bi JK bistabili funkcionirali kao dijelila frekvencije na njihove J i K ulaze dovedena je logička 1 tijekom cijele simulacije, što znači da će oni na izlazu davati izlazni signal dvostruko manje frekvencije u odnosu na *Clock* signal doveden na njihov ulaz. Spajanjem izlaznog signala Q0, bistabila JK0, na *Clock* signal bistabila JK1 dobiven je izlazni signal Q1 koji je

dvostruko manje frekvencije u odnosu na Q0. Daljnjim spajanjem Q1 na *Clock* ulaz JK2 i Q2 na *Clock* ulaz JK3 frekvencija je svaki puta dvostruko smanjivana sve do signala Q3 koji predstavlja bit najveće težine, što znači da mora imati najmanju frekvenciju.

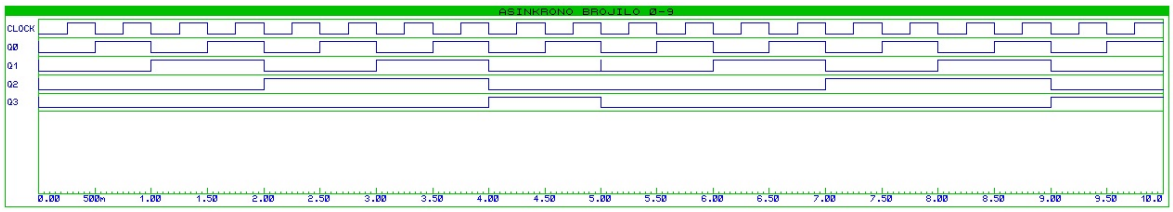
U teoriji, brojilo sastavljeno od četiri bistabila može brojati od 0 do 15. U ovom slučaju to nije tako. Ciklus brojanja ovog bistabila ograničen je naknadno dodanim NI logičkim vratima na čije su ulaze spojeni izlazi Q1 i Q3. Oba ova ulaza biti će u stanju logičke 1 u trenutku kada je stanje brojila 1010 u binarnom zapisu (dekadska vrijednost 10). To znači da će u tom trenutku na izlazu NI vrata biti logička 0, što će poništiti bistabile jer njihovi *Reset* ulazi reagiraju na logičku 0. Ciklus brojanja ovog asinkronog brojila tako je ograničen na brojanje od 0 do 9.



Slika 3.14 Asinkrono brojilo 0-9

Na grafu (Slika 3.15) je vidljiva promjena izlaznih signala Q0, Q1, Q2 i Q3 i ulaznog *Clock* signala za bistabil JK0, odnosno osnovnog *Clock* signala ovog sklopa. Kombinacija signala Q3, Q2, Q1 i Q0 tijekom jedne periode osnovnog *Clock* signala predstavlja 4-bitni zapis u binarnom brojevnom sustavu, gdje je Q3 bit najveće težine, a Q0 bit najmanje težine. Kombinacija signala u početnom trenutku glasi 0000, što u dekadskom brojevnom sustavu predstavlja broj 0. Tijekom slijedeće periode osnovnog *Clock* signala kombinacija glasi 0001, što predstavlja broj 1 u dekadskom brojevnom sustavu. Tijekom svake slijedeće periode osnovnog *Clock* signala, gledajući u dekadskom brojevnom sustavu, izlazno stanje bistabila će se jedinično povećavati sve do broja 9. Približno u istom trenutku kada se stanje brojila promijeni u broj 10, odnosno 1010, zapisano u binarnom brojevnom sustavu, brojilo se poništi zbog signala Q3 i Q1 čija stanja

logičke 1 preko NI vrata poništе stanja bistabila. Nakon poništavanja bistabila stanje brojila, odnosno kombinacija izlaznih signala Q3, Q2, Q1 i Q0 biti će opet 0000 i tako će cijeli ciklus brojanja krenuti ispočetka.



Slika 3.15 Graf asinkronog brojila 0-9

3.5.2 Sinkrono brojilo 0-9

Sinkrona brojila, kao i ona asinkrona, mogu biti izvedena od različitog broja bistabila, ali je na *Clock* ulaze svih bistabila unutar brojila doveden zajednički *Clock* signal. Zajednički Clock signal znači i sinkroniziran rad, pa ih stoga i nazivamo sinkrona brojila.

Njihov maksimalni brojevni ciklus ograničen je sa brojem bistabila unutar brojila. Projektiranje sinkronih brojila ograničenog brojevnog ciklusa prilično je zahtjevnije nego projektiranje asinkronih brojila i zahtijeva detaljnije projektiranje cjelokupnog sekvencijskog sklopa za određeni brojevni ciklus.

Projektiranje svakog sekvencijskog sklopa za brojanje nekog brojevnog ciklusa započinje crtanjem tablice stanja tog sklopa. Brojilo u ovom primjeru ima ciklus brojanja od 0 do 9, a

Tabela 1 Stanja sinkronog brojila 0-9predstavlja njegovu tablicu stanja. U prvom stupcu ispisane su izlazne dekadske vrijednosti sinkronog brojila, a u zadnji redak ovog stupca stavljen je broj 0 koji označava da će brojilo nakon broja 9 promijeniti stanje u 0 i time ponovo započet ciklus brojanja. Stupci Q3, Q2, Q1, i Q0 predstavljaju izlazne signale svakog od bistabila čija kombinacija predstavlja binarni zapis dekadske vrijednosti u prvom stupcu. Ostali stupci predstavljaju J i K ulaze svih bistabila, gdje su J0 i K0 ulazi bistabila JK0, J1 i K1 ulazi bistabila JK1 itd. U ove stupce upisuju se logičke vrijednosti koje je potrebno dovesti na bistabile kako bi se kombinacije izlaznih stanja brojila mijenjale u skladu sa stupcima Q3, Q2, Q1, i Q0.

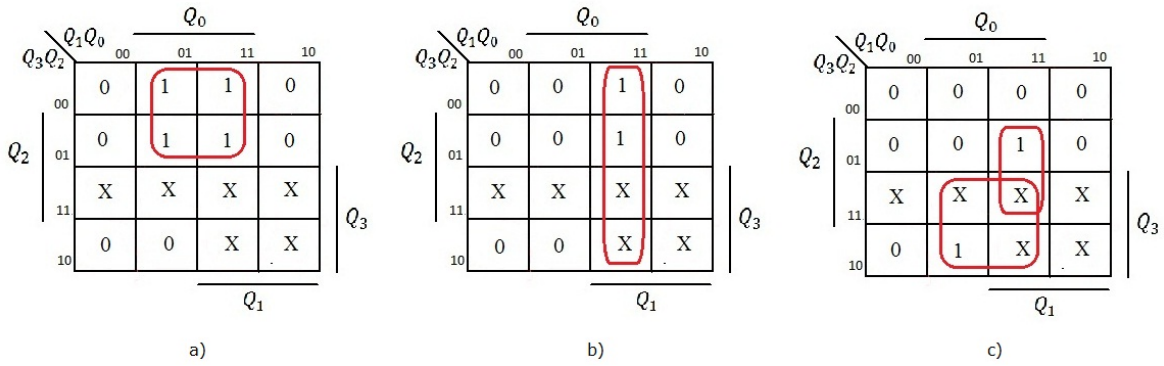
Dekadska vrijednost	Q3	Q2	Q1	Q0	J0	K0	J1	K1	J2	K2	J3	K3
0	0	0	0	0	1	1	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1	0	0	0	0
2	0	0	1	0	1	1	0	0	0	0	0	0
3	0	0	1	1	1	1	1	1	1	1	0	0
4	0	1	0	0	1	1	0	0	0	0	0	0
5	0	1	0	1	1	1	1	1	0	0	0	0
6	0	1	1	0	1	1	0	0	0	0	0	0
7	0	1	1	1	1	1	1	1	1	1	1	1
8	1	0	0	0	1	1	0	0	0	0	0	0
9	1	0	0	1	1	1	0	0	0	0	1	1
0	0	0	0	0								

Tabela 1 Stanja sinkronog brojila 0-9

U početnom trenutku kombinacija izlaznih signala je 0000, a pri slijedećoj periodi *Clock* signala ona se mora promijeniti u 0001. Stanja signala Q3, Q2 i Q1 moraju ostati u stanju 0, što znači da na ulazne signale J1-J3 i K1-K3 moraju biti dovedene logičke 0, pa se u prve redove tih stupaca upisuju 0. Logičko stanje signala Q0 mijenja se iz 0 u 1, što znači da na ulaze J0 i K0 mora biti dovedena logička 1 pa se broj 1 upisuje u prvi redak tih stupaca. Ovaj postupak se ponavlja za svaku slijedeću promjenu sve do one zadnje.

Na temelju vrijednosti unutar stupaca J0-J3 i K0-K3 iz

Tabela 1 Stanja sinkronog brojila 0-9popunjavaju se Karnaughove tablice (Slika 3.16) svakog stupca zasebno kako bi se izvela minimizirana funkcija ulaza koji predstavljen tim stupcem. Stupci J0 i K0 imaju u svakom trenutku vrijednost 1 i time su maksimalno minimizirane pa za njih nije potrebno crtati Karnaughovu tablicu. Iz stupaca ostalih ulaza vidljivo je da ulazi J1 i K1, J2 i K2 te J3 i K3 imaju iste vrijednosti u svakom retku pa za te parove vrijede iste Karnaughove tablice.



Slika 3.16 Karnaughove tablice signala: a) J1 i K1 b) J2 i K2 c) J3 i K3

Nakon minimiziranja Karnaughovim tabelama (Slika 3.16) dobivene su funkcije za svaki od J i K ulaza prikazane izrazima (3.5),(3.6) i (3.7). Ovi izrazi pokazuju koji se izlazi i u kojoj kombinaciji moraju dovesti na određeni ulaz. Kombinacije različitih izlaza Q realiziraju se pomoću osnovnih logičkih vrata.

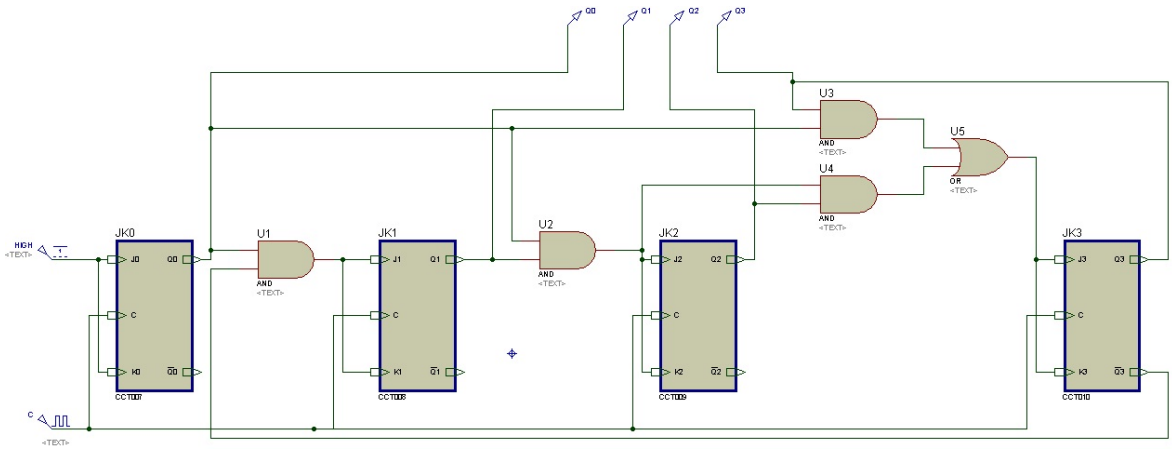
$$J1 = K1 = \overline{Q_3} Q_0 \tag{3.5}$$

$$J2 = K2 = Q_1 Q_0 \tag{3.6}$$

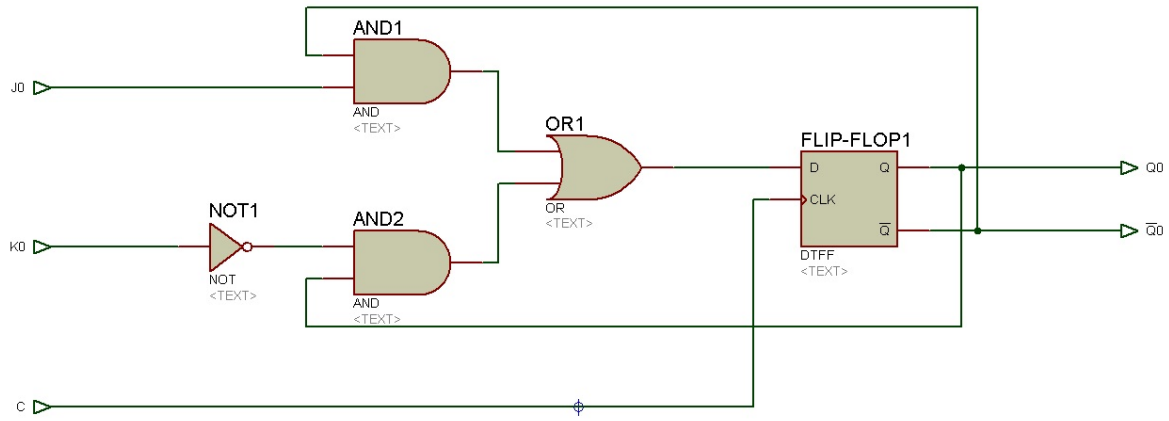
$$J3 = K3 = Q_3 Q_0 + Q_2 Q_1 Q_0 \tag{3.7}$$

Za simulaciju u Proteus VSM-u korišten je sklop sastavljen od kombinacije četiri JK bistabila i više osnovnih logičkih vrata koje su potrebne za dovođenje određenih, projektiranjem dobivenih kombinacija izlaza Q, na određene J i K ulaze bistabila (Slika 3.17). Kućišta koja predstavljaju JK bistabile u ovom sklopu (JK0-JK3) nisu standardne komponente iz knjižnice Proteus VSM-a, već su to kućišta unutar kojih se nalaze podsustavi (Slika 3.18) izrađeni od osnovnih logičkih vrata i D bistabila. Osnovna logička vrata i D bistabil unutar svakog podsustava sačinjavaju sklop prikazan u poglavlju 3.4.3 JK bistabil koji djeluje kao JK bistabil. Ulazi J0 i K0 postavljeni su u stanje logičke 1 tijekom cijele simulacije u skladu sa tablicom stanja (

Tabela 1). Na J i K ulaze ostalih bistabila, preko logičkih vrata, dovedeni su signali čija logička stanja u svakom trenutku odgovaraju funkcijama (3.5), (3.6) i (3.7).

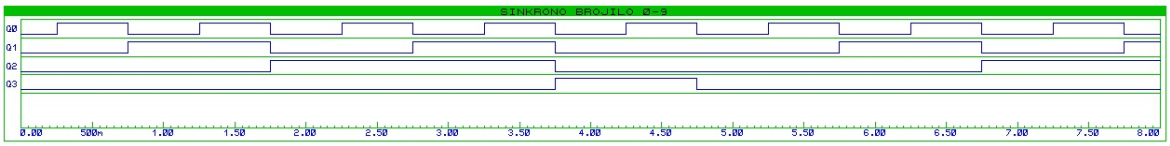


Slika 3.17 Sinkrono brojilo 0-9



Slika 3.18 Podsustav jednog od JK kućišta

Na grafu ovog sinkronog brojila (Slika 3.19) prikazani su njegovi izlazni signali Q3, Q2, Q1 i Q0. Kako je već poznato, njihova kombinacija čini brojevno stanje ovog brojila, gdje je signal Q3 bit najveće težine, a signal Q0 bit najmanje težine. Ako se binarna kombinacija ovih signala očitava tijekom svake njene promjene i zatim se ista pretvara u dekadski brojevni sustav, primjećuje se da se tijekom jedne promjene kombinacije izlaznih signala stanje brojila dekadski jedinično poveća. Slijedeća kombinacija izlaznih signala brojila, nakon kombinacije 1001 (dekadski 9), je kombinacija 0000. To znači da brojevni ciklus tada kreće ispočetka. Time se može zaključiti da je ovaj sekvencijski sklop pravilno projektiran kako bi vršio funkciju sinkronog brojanja od 0 do 9.

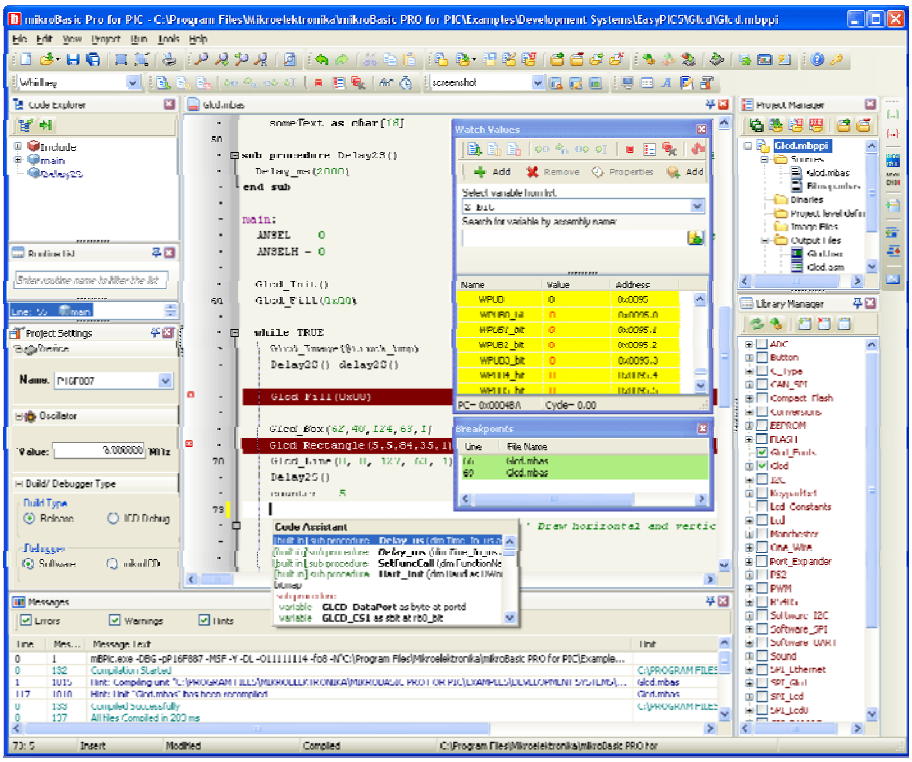


Slika 3.19 Graf sinkronog brojila 0-9

4. Razvoj mikrokontrolerskih aplikacija

4.1 Uvod u mikroPascal

Najvažniji dio svakog mikrokontrolerskog sustava svakako je mikrokontroler. Sklopovski gledano, mikrokontroler sastoji se od kućišta koje sadrži izvode za napajanje mikrokontrolera, izvode za spajanje kvarcnog oscilatora koji određuje njegovu radnu frekvenciju i određeni broj ulazno/izlaznih izvoda čija količina varira u odnosu na izvedbu. Unutar kućišta nalaze se razne komponente poput centralne procesorske jedinice, analogno/digitalnog pretvarača, RAM memorije, unutarnjeg oscilatora, sklopa za serijsku komunikaciju, sklopa za napajanje itd. Sama funkcija mikrokontrolera, međutim, nije sklopovski određena, već je mikrokontroler potrebno programirat kako bi mu se time zadala neka funkcija. Podršku za razvoj u programskom dijelu pružaju razvojne okoline za razvoj programa mikrokontrolera. Razvojne se okoline razlikuju s obzirom na vrstu odabranog mikrokontrolera i s obzirom na vrstu programskog jezika u kojem želimo programirati odabrani mikrokontroler.

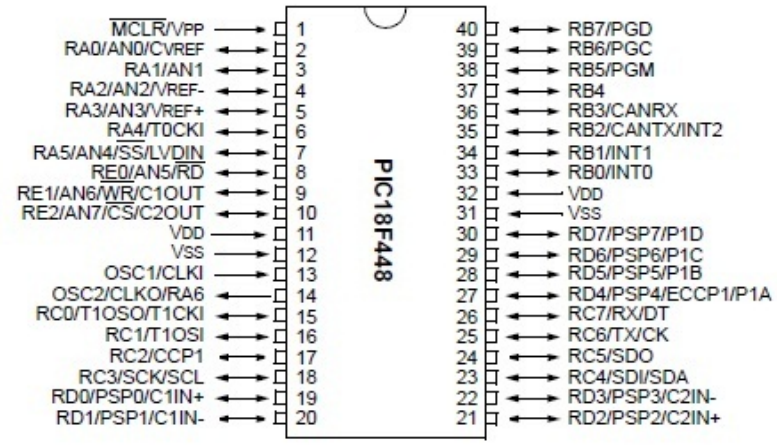


Slika 4.1 Grafičko sučelje mikroPascal-a [S2]

Jedna od takvih razvojnih okolina je mikroPascal (Slika 4.1) za PIC mikrokontrolere. To je PASCAL programski prevoditelj dizajniran za razvijanje aplikacija za PIC mikrokontrolere, te njihov prijenos na sam mikrokontroler. On ujedno sadrži i program za uočavanje pogreški kako bi se njihov broj sveo na minimum. Upravo u mikroPascalu biti će izrađeni primjeri u daljnjem tekstu, koji su korišteni pri simulaciji razvojnog sustava za PIC mikrokontrolere 5.3 Prikaz simulacije komponenti razvojnog sustava pomoću odgovarajućih mikrokontrolerskih aplikacija.

4.2 PIC18F448 mikrokontroler

Kako bi se u potpunosti razumjeli primjeri u mikroPascalu koji slijede potrebno je napomenuti i opisati osnovne karakteristike mikrokontrolera korištenog u tim primjerima. Radi se o PIC mikrokontroleru, serije PIC18, potpunog naziva PIC18F448. Ovaj mikrokontroler sadrži 40 izvoda (Slika 4.2) koji se mogu podijeliti u nekoliko osnovnih skupina. Prva skupina su tzv. osnovni izvodi u koje spadaju izvodi preko kojih se napaja mikrokontroler, izvod za uzemljenje, te $\overline{\text{MCLR}}$ izvod koji se koristi za poništavanje svih funkcija koje trenutno izvodi mikrokontroler. Kao drugu skupinu može se navesti skupina dodatnih izvoda u koju spadaju izvodi za spajanje vanjskog oscilatora, ukoliko korisnik ima potrebu raditi na frekvenciji koju ne podržava oscilator unutar samog mikrokontrolera. Ostale skupine čine ulazno/izlazni podatkovni izvodi podijeljeni u pet skupina RA, RB, RC, RD i RE. Njihova ulazno/izlazna funkcija određuje se pomoću TRIS registara svake skupine. TRIS registar svake skupine podatkovnih izvoda ima broj bitova jednak broju ulazno/izlaznih izvoda njemu odgovarajuće skupine. U slučaju PIC18F448 mikrokontrolera, postavljanjem npr. određenog TRISB bita u stanje 1, dodijelit će njemu odgovarajućem izvodu skupine RB ulogu ulaznog izvoda, dok će postavljanje određenog TRISB izvoda u stanje 0 dodijeliti njemu odgovarajućem izvodu skupine RB ulogu izlaznog izvoda. Stanje TRIS registara za podatkovne izvode koje će se koristi u određenom projektu tijekom programiranja u mikroPascalu potrebno je odrediti na početku svakog projekta.



Slika 4.2 Dijagram izvoda PIC18F448 mikrokontrolera [S3]

4.3 „Trčeće“ diode

U ovom projektu kao periferne jedinice koriste se LED diode, točnije njih 8, spojene na RB izvode mikrokontrolera, od RB0 do RB7. Katode ovih LED dioda spojene su na uzemljenje, dok je anoda svake diode spojena na jedan od RB izvoda mikrokontrolera. Način na koji su spojene ove diode biti će detaljnije objašnjen u poglavlju 5.2.1 LED diode. Stanje ovih dioda, dakle, ovisi o stanju zasebnog RB izvoda za svaku diodu. Ukoliko je u nekom trenutku stanje izvoda mikrokontrolera logička 1 dioda spojena na taj izvod će svijetliti, dok će, ukoliko je stanje tog izvoda u nekom drugom trenutku logička 0, dioda biti ugašena. Cilj ovog projekta je tzv. „trčanje“ dioda. „Trčanje“ se postiže na način da, krenuvši od diode spojene na izvod RB0, u svakom slijedećem trenutku jedna dioda bude aktivna, sve do diode RB7, nakon koje se proces ponavlja. Programski kod ovog projekta prikazan je na kraju poglavlja 4.3, a njegov detaljni opis slijedi u nastavku teksta.

Opis projekta, odnosno njegovog programskog koda, valja krenuti od njegove glavne funkcije. Kao i svaka funkcija i glavna funkcija počinje izrazom begin (*hrv. započni*), a završava izrazom end (*hrv. završi*). Na početku glavne funkcije stanje registra TRISB postavljeno je u stanje 0, što znači da svi izvodi skupine RB djeluju kao izlazi mikrokontrolera. U mikroPascalu skupina izvoda, u ovom slučaju RB označava se sa portb, dok se svaki zaseban izvod, npr. RB0, označava sa portb.0. Izraz portb može se sagledati kao 8-bitni broj čiji svaki bit predstavlja jedan izvod, krenuvši od izvoda portb.0 koji predstavlja LSB, odnosno bit najmanje težine, do izvoda portb.7 koji predstavlja MSB, odnosno bit najveće težine. Izrazom „portb:=0“ početna stanja svih izvoda ove skupine postavljena su u

stanje 0. 8-bitni binarni zapis broja portb sada je 00000000. Prvi uvjet koji treba ispunit da bi se postiglo „trčanje“ je da u jednom trenutku bude upaljena samo jedna dioda, dakle, broj portb mora sadržavati binarnu znamenku 1 na samo jednom bitu u jednom trenutku. Drugi uvjet je da binarna znamenka 1 bude u svakom slijedećem trenutku na slijedećem bitu, krenuvši od LSB bita, te da se nakon MSB bita vraća na LSB bit kako bi se proces ponovio. Kako bi se ovaj proces stalno iznova ponavljao kod glavne funkcije izvodi se unutar beskonačne petlje koja počinje sa repeat (*hrv. ponavljaj*), a završava izrazom until (*hrv. sve dok*). Izraz „until 1=2“ označava uvjet nakon kojeg će se, ukoliko bude ispunjen, petlja prestati ponavljati. Promjena stanja 8-bitnog binarnog zapisa portb (Tabela 2), može se zapisati i dekadski (Tabela 2). kod dekadske promjene vidljivo je da je svako slijedeće stanje broja portb slijedeća potencija broja 2. Za dobivanje upravo ovakve dekadske promjene stanja izvedena je funkcija Manual_pow.

Vrijeme proteklo tijekom izvršavanja programa (ms)	portb (binarni zapis)	portb (dekadski zapis)
0	00000001	1
500	00000010	2
1000	00000100	4
1500	00001000	8
2000	00010000	16
2500	00100000	32
3000	01000000	64
3500	10000000	128

Tabela 2 Promjena stanja 8-bitnog zapisa izvoda portb

Zbog 8 različitih stanja ova funkcija mora se izvesti 8 puta unutar for petlje čiji se brojač mijenja od 0 do 7. Varijabla x unutar ove funkcije tijekom svakog izvođenja imati će stanje 2, dok će varijabla y imati stanje jednako varijabli i u svakom trenutku izvođenja. Stanja ovih varijabli određena su zagradom prilikom pozivanja funkcije Manual_pow i zagradom prilikom njenog deklariranja. U prvom trenutku, kada je y=0 ova funkcija će dati rezultat 1 zbog izraza „result :=1“ i portb

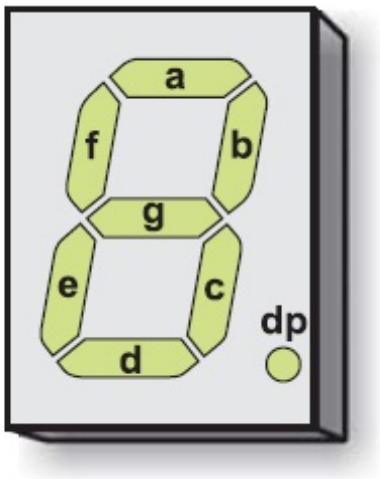
tako će poprimit vrijednost 1. Prije nego što for petlja krene ispočetka program će pričekati sa izvođenjem 500ms zbog izraza „delay_ms(500)“. Nakon toga i=1,pa je tako i y=1, te će rezultat funkcije Manual_pow biti x, odnosno 2. Nakon još 500ms biti će y=2, a tijek događaja unutar funkcije bit će taj da će result prvo poprimiti vrijednost 2 zbog izraza „result:=x“, a zatim će se izvršiti for petlja zbog koje će result, a tako i portb poprimit vrijednost 4. Zbog for petlje unutar glavne funkcije još će se pet puta promijeniti stanje na portb, a zaključno će biti stanje kada je i=7, odnosno y=7, a portb ima vrijednost 128. Nakon toga proces će zbog iste for petlje krenuti ispočetka.

Promjenom stanja portb skupine izvoda svakih 500ms po potencijama broja 2, od 1 do 128, zapravo znači da će nakon svakih 500ms slijedeći izvod iz skupine portb davati logičku 1, odnosno da će u svakom slijedećem trenutku svijetliti slijedeća dioda.

```
program Trcece_diode;   //NAZIV PROJEKTA
var i,j : byte;   //DEKLARIRANJE VARIJABLI
function Manual_pow(x,y: byte) : byte;   //DEKLARIRANJE FUNKCIJE Manual_pow
  begin   //POČETAK FUNKCIJE Manual_pow
    if y=0 then result:=1   //SLUČAJ y=0
    else   //OSTALI SLUČAJEVI (0 < y ≤ 7)
      begin
        result:=x;
        for j:=1 to (y-1) do
          begin
          result:=x*result;
          end;
      end;
  end;   // ZAVRŠETAK FUNKCIJE Manual_pow
begin   //POČETAK GLAVNE FUNKCIJE
  trisb:=0;   //IZVODI portb.0-portb.7 DOBIVANJU ULOGU IZLAZA
  portb:=0;
  repeat   //POČETAK repeat PETLJE
    for i:=0 to 7 do
      begin
        portb:=Manual_pow (2,i);   //POZIVANJE FUNKCIJE Manual_pow
        delay_ms(500);
      end;
  until 1=2;   //UVJET ZA ZAVRŠETAK repeat PETLJE
end.   //ZAVRŠETAK GLAVNE FUNKCIJE
```

4.4 Brojač 0-1000

Kao periferne jedinice, u ovom projektu koristiti će se multipleksirani 7-segmentni ekrani, točnije, njih četiri. Izvodi svakog od segmenata ovih ekrana spojeni su na jednu sabirnicu, zajedničku za sva četiri ekrana, koja je spojena na RB skupinu izvoda mikrokontrolera. Svaki segment ovih ekrana označen je slovom od a do g (Slika 4.3), dok je decimalna točka označena sa dp (Slika 4.3). Preko zajedničke sabirnice, izvodi segmenata ovih ekrana spojeni su abecednim redom od a do g na izvode RB0-RB6, dok je decimalna točka oznake dp spojena na izvod RB7. Svaki segment zapravo je jedna LED dioda koja će svijetliti ukoliko su zadovoljena dva uvjeta. Prvi uvjet odnosi se na ekran čiji je taj segment dio, koji će biti aktivan samo ukoliko je na njegov aktivacijski ulaz dovedena logička 1. Drugi se pak, uvjet, odnosi na sami segment na koji je potrebno dovesti logičku 1 kako bi svijetlio. Način na koji su spojeni ovi ekrani i njihovo aktiviranje i deaktiviranje detaljnije su pojašnjeni u poglavlju 5.2.3 Multipleksirani 7-segmentni ekrani.



Slika 4.3 Raspored segmenata na jednom od četiri korištena 7-segmentna ekrana [S4]

Kako je već spomenuto, za potrebe ovog projekta korištena su četiri multipleksirana 7-segmentna ekrana, čiji su segmenti zajedničkom sabirnicom spojeni na izvode RB0-RB7. Sva četiri ekrana biti će testirana projektom pisanim u mikroPascalu kojim će se, preko mikrokontrolera, na ovim ekranima izvoditi brojanje od 0 do 1000. Programski kod ovog projekta, čiji opis slijedi u nastavku teksta, nalazi se na kraju poglavlja 4.4

Ekrani su označeni brojevima od 1 do 4, gledajući s lijeva na desno, i oni će, gledajući istim redom, prikazivati tisućice, stotice, desetice i jedinice. Brojke će ispisivati na određenom ekranu preko izlaza RB0-RB7 označenih u mikroPascalu zajedničkim imenom portb, dok će se aktivacija ekrana vršiti preko izvoda RD4-RD7, označenih izrazima od portd.4 do portd.7. Samo brojanje od 0 do 1000, odnosno promjena znamenki na ekranima izvršeno je unutar tri hijerarhijski postavljene for petlje, koje mijenjaju stanje svojih varijabli od 0 do 9. Petlja kojom se mijenja varijabla stotice hijerarhijski je najviša petlja i mijenja se svaki stoti put. Hijerarhijski ispod nje nalazi se for petlja za promjenu desetica koja se mijenja svaki deseti put. Hijerarhijski najniža petlja je ona kojom se mijenjaju jedinice i ona se mijenja svaki puta, pa se zato unutar nje i nalazi glavni dio koji je namijenjen promjeni stanja na ekranima. U prošlom poglavlju već je spomenuto da se portb može prikazati kao 8-bitni binarni zapis (Tabela 3) čija svaka znamenka označava stanje na pojedinom izlazu RB, gdje je RB0 bit najmanje težine, a RB7 bit najveće težine. Međutim, pri zadavanju novog stanja za portb korišten je dekadski zapis (Tabela 3), koji zatim sam mikroPascal shvaća kao spomenuti 8-bitni binarni zapis.

Dio koda koji se nalazi unutar tri spomenute for petlje izvršiti će se 999 puta i tijekom cijelog tog procesa ekran na kojem se ispisuju tisućice mora prikazivati vrijednost 0, odnosno portb mora imati vrijednost 63. Već je spomenuto da su segmenti ovih ekrana spojeni na mikrokontroler zajedničkom sabirnicom, što znači da u jednom trenutku samo jedno stanje može biti na izvodima segmenata sva četiri ekrana i tu nastaje problem. Kod brojanja u većini slučajeva ekrani moraju imati različito stanje na izvodima svojih segmenata, odnosno prikazivati različiti broj u istom trenutku. Ovaj problem rješava se naizmjeničnim aktiviranjem ekrana tako da se najprije na portb postavi stanje za prikaz željenog broja, a zatim se aktivira samo ekran na kojem se želi prikazati taj broj, drži se aktiviranim neko kratko vrijeme i zatim se ugasi. Prvi primjer ovog proces je ekran za tisućice. Na portb postavljena je vrijednost 63, a nakon toga izrazom portd.4:=1 aktivira se ekran za prikaz tisućica, drži se aktiviranim 10ms i nakon toga se deaktivira.

portb (binarni zapis)	portb (dekadski zapis)	Broj prikazan na 7-segmentnom ekranu
00111111	63	0
00000110	6	1
01011011	91	2
01001111	79	3
01100110	102	4
01101101	109	5
01111101	125	6
00000111	7	7
01111111	127	8
01101111	111	9

Tabela 3 Stanje na 7-segmentnom ekranu u odnosu na stanje portb

Nakon što je postavljeno stanje 0 na prvi ekran, slijedi serija upita if (*hrv. ako*) kako bi se, ovisno o trenutnom stanju varijable jedinice, na skupinu izlaza portb postavilo odgovarajuće stanje. Iza ovih upita slijedi aktivacija ekrana koji prikazuje jedinice preko izlaza portd.7, zatim delay *(hrv. kašnjenje)* od 10ms, te se zatim ovaj ekran deaktivira. Nakon jedinica, gotovo identičan proces odvija se i u slučaju desetica, a zatim i stotica. Jedine su razlike upiti koji se ovaj put odnose na varijablu desetice, odnosno stotice i redni broj izvoda na portd gdje portd.6 predstavlja izvod za aktiviranje/deaktiviranje ekrana za prikaz desetica, a portd.5 izvod za aktiviranje/deaktiviranje ekrana za prikaz stotica.

Cjelokupni proces unutar tri već spomenute for petlje, kako je već rečeno ponovit će se 999 puta. Kako bi se prikazao zadnji broj (broj 1000) postavljena su tri upita if. Prvi upit propitkuje ima li varijabla stotice vrijednost 9, ukoliko ima prelazi se na drugi upit koji propitkuje ima li varijabla desetice vrijednost 9 i ukoliko ima prelazi se na treći upit. Treći upit propitkuje ima li varijabla jedinice vrijednost 9 i ukoliko ima znači da je na ekranima već ispisana vrijednost 999, te je potrebno ispisati vrijednost 1000, što se izvršava pomoću koda unutar ova tri upita. Kako bi se ispisao broj 1000 najprije je izrazom „portb:=6“ i aktivacijom ekrana izrazom portd.4:=1 na ekran koji prikazuje tisućice ispisana jedinica. Nakon 10ms ekran za prikaz tisućica je deaktiviran, izlazi portb su izrazom „portb:=63“ postavljeni u

zajedničko stanje 63 i zatim se redom aktiviraju i deaktiviraju ekrani za prikaz stotica, desetica i jedinica, te se na svakom od njih prikazuje broj 0. Iz razloga što se ovaj dio koda nalazi unutar beskonačne repeat petlje, proces ispisivanja broja 1000 izvoditi će se dok god se ne poništi program unutar mikrokontrolera.

```
program Multipleksirani_7_segm_ekrani;   //NAZIV PROJEKTA
var stotice, desetice, jedinice : byte;   //DEKLARIRANJE VARIJABLI
begin   //POČETAK GLAVNE FUNKCIJE
trisd:=0;   //IZVODI portd.0-portd.7 DOBIVANJU ULOGU IZLAZA
trisb:=0;   //IZVODI portb.0-portb.7 DOBIVANJU ULOGU IZLAZA
portd:=0;
portb:=0;
for stotice:=0 to 9 do   //FOR PETLJA ZA PROMJENU STOTICA
begin
  for desetice:=0 to 9 do   //FOR PETLJA ZA PROMJENU DESETICA
  begin
    for jedinice:=0 to 9 do   //FOR PETLJA ZA PROMJENU JEDINICA
    begin
      portb:=63;
      portd.4:=1;   //AKTIVACIJA EKRANA KOJI PRIKAZUJE TISUČICE
      delay_ms(20);
      portd.4:=0;   //DEAKTIVACIJA EKRANA KOJI PRIKAZUJE TISUĆICE
      if jedinice=0 then portb:=63;
      if jedinice=1 then portb:=6;
      if jedinice=2 then portb:=91;
      if jedinice=3 then portb:=79;
      if jedinice=4 then portb:=102;
      if jedinice=5 then portb:=109;
      if jedinice=6 then portb:=125;
      if jedinice=7 then portb:=7;
      if jedinice=8 then portb:=127;
      if jedinice=9 then portb:=111;
      portd.7:=1;   //AKTIVACIJA EKRANA KOJI PRIKAZUJE JEDINICE
      delay_ms(20);
      portd.7:=0;   //DEAKTIVACIJA EKRANA KOJI PRIKAZUJE JEDINICE
      if desetice=0 then portb:=63;
      if desetice=1 then portb:=6;
      if desetice=2 then portb:=91;
      if desetice=3 then portb:=79;
      if desetice=4 then portb:=102;
      if desetice=5 then portb:=109;
      if desetice=6 then portb:=125;
      if desetice=7 then portb:=7;
```

```
    if desetice=8 then portb:=127;
    if desetice=9 then portb:=111;
    portd.6:=1;   //AKTIVACIJA EKRANA KOJI PRIKAZUJE DESETICE
    delay_ms(20);
    portd.6:=0;   //DEAKTIVACIJA EKRANA KOJI PRIKAZUJE DESETICE
    if stotice=0 then portb:=63;
    if stotice=1 then portb:=6;
    if stotice=2 then portb:=91;
    if stotice=3 then portb:=79;
    if stotice=4 then portb:=102;
    if stotice=5 then portb:=109;
    if stotice=6 then portb:=125;
    if stotice=7 then portb:=7;
    if stotice=8 then portb:=127;
    if stotice=9 then portb:=111;
    portd.5:=1;   //AKTIVACIJA EKRANA KOJI PRIKAZUJE STOTICE
    delay_ms(20);
    portd.5:=0;   //DEAKTIVACIJA EKRANA KOJI PRIKAZUJE STOTICE
    if stotice=9 then
    begin
      if desetice=9 then
      begin
        if jedinice=9 then
        begin
          repeat
          portb:=6;
          portd.4:=1;
          delay_ms(5);
          portd.4:=0;
          portb:=63;
          portd.5:=1;
          delay_ms(5);
          portd.5:=0;
          portd.6:=1;
          delay_ms(5);
          portd.6:=0;
          portd.7:=1;
          delay_ms(5);
          portd.7:=0;
          delay_ms(5);
          until 1=2;
        end;
      end;
    end;
  end;
```

```
  end;
end;
end.   //ZAVRŠETAK GLAVNE FUNKCIJE
```

4.5 Brojač vremena

LCD ekran periferna je jedinica koja će biti korištena u ovom poglavlju, te će programski kod za mikrokontroler, pisan u mikroPascalu i ispisan na kraju ovog poglavlja, služiti upravo kako bi se prikazao način rada LCD ekrana. Radi se o ekranu dimenzija 20x2, što znači da ima dva retka, od kojih svaki može prikazati 20 znakova. Detaljni opis ovog LCD ekrana nalazi se u poglavlju 5.2.4 LCD ekran.

Funkcija koju će izvoditi programski kod u nastavku biti će brojač vremena koji korisnik može po želji pokretat i zaustavljat u bilo kojem trenutku tijekom izvođenja programa.

Interakcija korisnika sa brojačem vrši se preko dva tipkala spojena na mikrokontrolerske izvode RD6 i RD7, u kodu mikroPascala označene sa portd.6 i portd.7. Iz toga razloga je potrebno registar TRISD postaviti izrazom „trisd:=255“ u stanje 255. Time su izvodi mikrokontrolera od RD0 do RD7 dobili ulogu ulaza, što je bitno samo za spomenuta dva izvoda, RD6 i RD7, dok ostali izvodi ne igraju nikakvu ulogu u ovom projektu. Ova tipkala funkcioniraju na način da se njihovim pritiskom dovodi logička 1 na ulaz mikrokontrolera i upravo će se ta karakteristika koristiti u upitima pri pokretanju, odnosno zaustavljanju brojača. Detaljniji opis načina na koji su spojena ova tipkala prikazan je u poglavlju 5.2.2 Tipkala.

Za prikaz znamenki na ekranu korišten je 16-znakovni niz lcd čije je početno stanje postavljeno izrazom „lcd:='Vrijeme: 0 0'“. Naredbom lcd_init aktivira se LCD ekran, a izraz portb unutar zagrada znači da se ta aktivacija vrši preko izvoda skupine RB. MikroPascal ima već standardno predodređenu ulogu za svaki od portb ulaza pa nije potrebno određivanje uloge pojedinog ulaza. Točna uloga svakog od ovih izvoda, pri aktivaciji LCD ekrana, opisana je u poglavlju 5.2.4 LCD ekran. Ukoliko se LCD ekranu želi dati neka dodatna naredba koristi se izraz lcd_cmd i zagrada nakon njega, unutar koje se upisuje željena naredba. U ovom slučaju to je naredba za isključivanje pokazivača trenutnog položaja na LCD ekranu. Naredba lcd_out koristi se za ispisivanje sadržaja na LCD ekran. Prva dva broja unutar zagrade ove naredbe predstavljaju redni broj retka, odnosno stupca, od kojeg se želi započet ispis. Izraz nakon ovih brojki

predstavlja ime niza koji se želi ispisati, a to je u ovom slučaju znakovni niz lcd. Ovo stanje varijable biti će ispisano na ekranu dok god korisnik ne pritisne tipkalo spojeno preko ulaza portd.7 jer će tek tada uvjeta upita „if portd.7=1“ biti zadovoljen i ostatak programa će krenuti sa izvršavanjem. Brojanje vremena, odnosno promjena sekundi i minuta vrši se preko varijabli sec i min čija promjena stanja označava upravo promjenu sekundi, odnosno minuta. Kako bi se ta promjena prikazala na LCD ekranu potrebno je mjesta unutar 16-znakovnog niza lcd namijenjena prikazu minuta i sekundi zamijeniti vrijednostima ovih varijabli, ali to nije moguće učiniti direktno. Potrebno je najprije varijable min i sec prebaciti u neke znakovne nizove. To se izvodi pomoću funkcije ByteToStr unutar čije je zagrade najprije potrebno upisati varijablu koju se želi prebaciti u znakovni niz, a zatim i znakovni niz u koji se želi prebaciti ta varijabla. Ova funkcija uvijek prebacuje varijable u niz od tri znamenke, te jedinice, desetice i stotice ove varijable uvijek upisuje u niz na isti način (Tabela 4). Željeni nizovi u koji su prebačene varijable nazvani su min_lcd u slučaju varijable min i sec_lcd u slučaju varijable sec.

Znamenka varijable	stotica	desetica	jedinica
Redni broj znaka unutar znakovnog niza	0	1	2

Tabela 4 Raspodjela znamenki varijable unutar znakovnog niza nakon izvršene funkcije ByteToStr

Nakon što su varijable min i sec pretvorene u željene nizove ,potrebno je tim nizovima zamijeniti odgovarajuća mjesta unutar znakovnog niza lcd (Tabela 5) čiji će se znakovi ispisati na ekranu ponovnim ispisivanjem naredbe „lcd_out(1,1,lcd)“. Tijekom prvog izvršavanja ovog dijela koda, unutar petlje repeat, stanje na ekranu ostati će nepromijenjeno jer se stanja varijabli sec i min još uvijek nisu promijenila.

Znakovni niz	Redni broj znaka unutar niza															
lcd	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lcd_min	-	-	-	-	-	-	-	-	-	0	1	2	-	-	-	-
lcd_sec	-	-	-	-	-	-	-	-	-	-	-	-	-	0	1	2

Tabela 5 Zamjena odgovarajućih znakova unutar niza lcd odgovarajućim znakovima nizova lcd_min i lcd_sec

Varijabla sec jedinično se povećava tek izrazom „sec:=sec+1“, nakon čega se provjerava je li varijabla sec djeljiva sa 60, odnosno je li ostatak tog dijeljenja jednak broju 0. Ukoliko je varijabla zaista djeljiva sa 60, varijabla min se jedinično poveća, dok se varijabla sec postavi u stanje 0. Kako bi se varijabla sec mijenjala svake sekunde, a time i varijabla min svake minute, ispisan je izraz „delay_ms(1000)“ čime je određeno da će se ostatak koda izvršiti tek nakon što prođe jedna sekunda.

Petlja repeat postavljena je sa uvjetom until „portd.6=1“. Na ulazu portd.6 biti će logička 1 samo ukoliko je stisnuto tipkalo spojeno na taj ulaz i tada će se program zaustaviti, u suprotnom, program unutar repeat petlje će se nastaviti izvoditi i stanje na ekranu će se mijenjati čime će se vršiti brojanje vremena sve dok korisnik ne pritisne tipkalo spojeno na portd.6. Nakon zaustavljanja procesa brojanja u svakom slijedećem trenutku moguće je nastaviti brojanje pritiskom tipkala spojenog na portd.7.

```
program LCD_ekran;   //NAZIV PROJEKTA
var    sec, min  :  byte;   //DEKLARIRANJE VARIJABLI
       lcd  :  string[16];   //DEKLARIRANJE 16-ZNAKOVNOG NIZA lcd
       sec_lcd  :  string[3];   //DEKLARIRANJE 3-ZNAKOVNOG NIZA sec_lcd
min_lcd  :  string[3];   //DEKLARIRANJE 3-ZNAKOVNOG NIZA min_lcd
begin    //POČETAK GLAVNE FUNKCIJE
trisd:=255;    //IZVODI portd.0-portd.7 DOBIVANJU ULOGU ULAZA
portd:=0;
lcd:='Vrijeme:   0   0';   //POČETNO STANJE 16-ZNAKOVNOG NIZA lcd
Lcd_Init(portb);   //INICIRANJE LCD EKRANA PREKO portb
Lcd_Cmd(lcd_cursor_off);   //NAREDBA LCD EKRANU ZA ISKLJUČIVANJE POKAZIVAČA
Lcd_Out(1,1,lcd);   //NAREDBA ZA ISPISIVANJE NA LCD EKRANU
min:=0;
sec:=0;
```

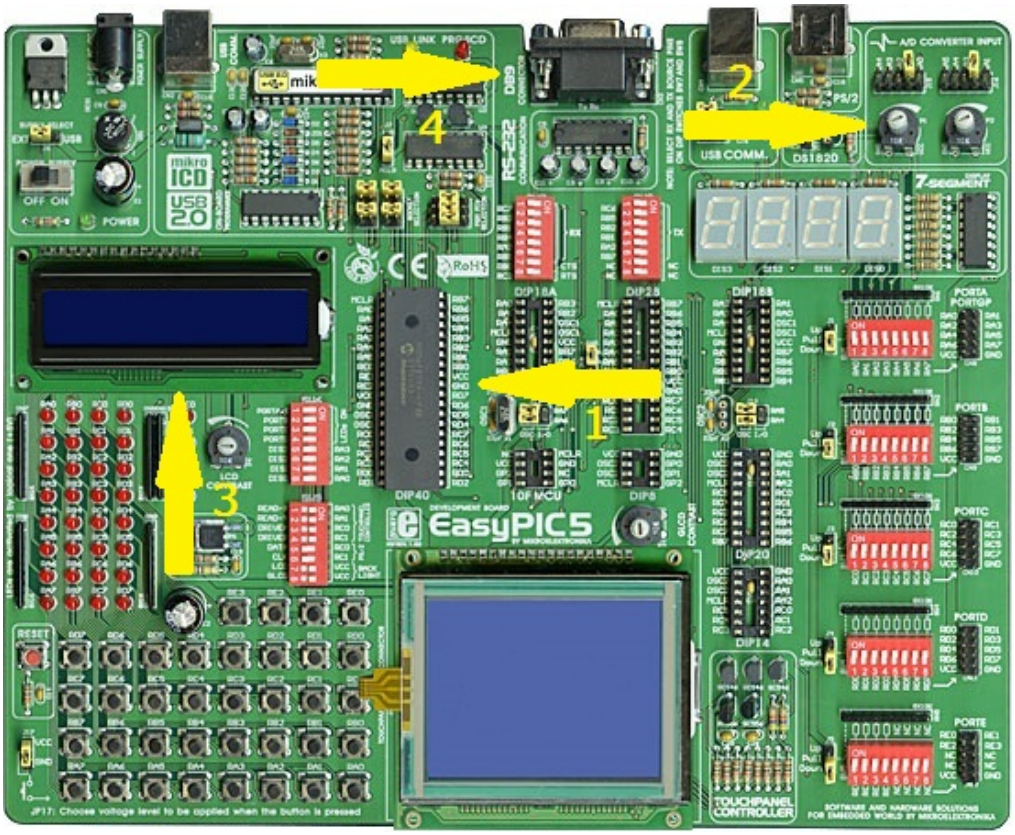
```
sec_lcd:='  0';   //POČETNO STANJE 3-ZNAKOVNOG NIZA sec_lcd
min_lcd:='  0';   //POČETNO STANJE 3-ZNAKOVNOG NIZA min_lcd
repeat
  if portd.7=1 then
  begin
    repeat
      ByteToStr(min, min_lcd);  //PRETVARANJE VARIJABLE min U 3-ZNAKOVNI NIZ
min_lcd
      ByteToStr(sec,sec_lcd);   //PRETVARANJE VARIJABLE sec U 3-ZNAKOVNI NIZ
sec_lcd
      lcd[9]:=min_lcd[0];
      lcd[10]:=min_lcd[1];
      lcd[11]:=min_lcd[2];
      lcd[13]:=sec_lcd[0];
      lcd[14]:=sec_lcd[1];
      lcd[15]:=sec_lcd[2];
      Lcd_Out(1,1,lcd);
      sec:=sec+1;
      if sec mod 60=0 then   //PROVJERA DJELJIVOSTI VARIJABLE sec SA 60
      begin
        min:=min+1;
        sec:=0;
      end;
      delay_ms(1000);
    until portd.6=1;
  end;
until 1=2;
end.   //ZAVRŠETAK GLAVNE FUNKCIJE
```

4.6 Mjerenje napona u realnom okruženju

Projekti u mikroPascalu o kojima će biti riječi u ovom poglavlju pisani su sa namjerom da se prikaže mogućnost Proteus programskog paketa da ostvari komunikaciju sa realnom okolinom. Za tu će namjenu, uz mikrokontroler spojen na razvojni sustav unutar Proteusa, biti potrebno koristiti i jedan u realnom okruženju koji će očitavati promjenu pada napona na jednom od svojih izvoda. Komunikacija između ova dva mikrokontrolera vršiti će se RS232 serijskom komunikacijom, odnosno preko komponente COMPIM koja se nalazi u knjižnici komponenata Proteus programskog paketa. Komponenta COMPIM i cjelokupni protokol serijske komunikacije detaljno su objašnjeni u poglavlju 5.2.5 RS232 serijska komunikacija.

4.6.1 Mikrokontroler u realnom okruženju

Mikrokontroler u realnom okruženju, smješten je na razvojni sustav EasyPIC5 (Slika 4.4), na slici označen brojem 1 i isto je serije PIC18F448, poput mikrokontrolera unutar Proteusa. Kroz mikrokontrolersku aplikaciju pisanu u mikroPascalu, potrebno mu je zadati niz naredbi pomoću kojih će on mjeriti promjenu pada napona na jednom od svojih izvoda, tu promjenu slati serijskom komunikacijom mikrokontroleru unutar Proteusa, ali ju radi usporedbe rezultata i prikazivati na LCD ekranu koji se također nalazi na EasyPIC5 razvojnom sustavu. Brojem 2 (Slika 4.4), označen je potenciometar kojim se mijenja pad napona na RA2 izvodu mikrokontrolera, u rasponu 0-5V. LCD ekran na kojem se prikazuje promjena stanja izvoda RA2 označen je brojem 3 (Slika 4.4), dok je priključak preko kojeg se vrši serijska komunikacija sa računalom na kojem se nalazi Proteus označen sa brojem 4 (Slika 4.4). Kod mikrokontrolerske aplikacije namijenjene mikrokontroleru u realnom okruženju koji će biti opisan u nastavku, ispisan je na kraju ovog poglavlja.



Slika 4.4 Razvojni sustav EasyPIC5 [S5]

Već je spomenuto da se mjeri pad napona na potenciometru spojenom na izvod RA2, što znači da je najprije izvode RA potrebno postaviti kao ulaze odgovarajućom vrijednošću registra trisa. Slijedeća bitna stvar je postavljanje registra adcon1 u logičko stanje 0, čime je određeno da su ulazi RA digitalni. LCD ekran na ovom razvojnom sustavu nije spojen na izvode mikrokontrolera kao što je to inače običaj, pa je naredbom Lcd_Config i odgovarajućim izrazima u zagradi potrebno to prenijeti mikrokontroleru. Nakon toga se isključuje pokazivač na ekranu i naredbom Lcd_Out na prvo mjesto prvog retka ispisuje se željeni znakovni niz. Ovim naredbama određeno je početno stanje LCD ekrana nakon čega je potrebno inicirati serijsku komunikaciju izrazom Usart_Init.

Mikrokontroler PIC18F448 unutar sebe sadrži ADC modul, koji služi kao analogno digitalni pretvornik, za ulaze RA0 do RA5 koji mogu biti ili digitalni ili analogni. U slučaju ove mikrokontrolerske aplikacije već je određeno da će se signal koji dolazi na ulaze RA čitati kao digitalni, pa je tako ovim modulom potrebno pretvoriti analognu vrijednost pada napona na potenciometru u digitalnu vrijednost. Potenciometar je spojen na izvod RA2 koji je zapravo drugi kanal ADC pretvornika. Izrazom „adc_rd:=ADC_read(2)“ vrijednost tog drugog kanala pridodaje se varijabli adc_rd. Nakon toga ta se varijabla pretvara iz tipa word (*hrv. riječ*) u znakovni niz text, te se zatim taj znakovni niz ispisuje na LCD ekranu.

Posljednji dio aplikacije odnosi se na serijsku komunikaciju. Izrazom „Usart_Write_Text(text)“, serijskom se komunikacijom šalje znakovni niz text. Nakon njega šalje se i znakovni niz „OK“, koji će mikrokontroleru unutar Proteusa služiti kao provjera valjanosti trenutno dobivenih podataka.

```
program Mjerenje_napona_posiljatelj;   //NAZIV PROJEKTA
var adc_rd : word;   //DEKLARIRANJE VARIJABLE adc_rd TIPA word
    text : string[5];   //DEKLARIRANJE 5-ZNAKOVNOG NIZA text
begin   //POČETAK GLAVNE FUNKCIJE
trisa  := 255;   //IZVODI porta.0-porta.5 DOBIVANJU ULOGU ULAZA
Adcon1 := 0;   //KONFIGURIRANJE ANALOGNIH PINOVA U DIGITALNE
Lcd_Config(PORTB,3,2,1,0,PORTB,4,7,5);   //RUČNA KONFIGURACIJA PINOVA ZA
INICIRANJE LCD EKRANA
Lcd_Cmd(LCD_CURSOR_OFF);   //NAREDBA LCD EKRANU ZA ISKLJUČIVANJE
POKAZIVAČA
Lcd_Out(1,1,'Napon [0-1024]');   //NAREDBA ZA ISPISIVANJE NA LCD EKRANU
Usart_Init(2400);   //INICIRANJE SERIJSKE KOMUNIKACIJE PRI 2400 bps
```

```
while TRUE do
  begin
    adc_rd := ADC_read(2);   //UZMI VRIJEDNOST SA 2. KANALA ADC PRETVORNIKA
    WordToStr(adc_rd,text);    //PRETVARANJE VARIJABLE adc_rd IZ TIPA word U
5-
ZNAKOVNI NIZ text
    Lcd_Out(2,1,text);   //NAREDBA ZA ISPISIVANJE NA LCD EKRANU
    Usart_Write_Text(text);   //SLANJE ZNAKOVNOG NIZA text SERIJSKOM
KOMUNIKACIJOM
    Usart_Write_Text('OK');   //SLANJE IZRAZA „OK" SERIJSKOM KOMUNIKACIJOM
    delay_ms(500);
  end;
end.   //ZAVRŠETAK GLAVNE FUNKCIJE
```

4.6.2 Mikrokontroler u okruženju Proteus programskog paketa

Mikrokontroler u okruženju Proteus programskog paketa spojen je na već spomenuti razvojni sustav za mikrokontrolerske aplikacije, a njegov zadatak je primanje podataka koji se putem RS232 serijske komunikacije šalju sa mikrokontrolera u realnom okruženju, te njihovo ispisivanje na LCD ekranu unutar Proteusa. Mikrokontrolerska aplikacija ispisana za ovaj zadatak biti će objašnjena u nastavku, a njezin je kod ispisan na kraju ovog poglavlja.

Serijsku komunikaciju i LCD ekran potrebno je inicirati na početku glavne funkcije izrazima Usart_Init, odnosno Lcd_init. Nakon ovih naredbi zadana je ona za ispisivanje znakovnog niza „Napon [mV]“, gdje je prvi stupac prvog retka LCD ekrana određeno kao početno. Slijedećom naredbom ispisuje se znakovni niz koji sadrži samo prazna polja, i to u prvi stupac drugog retka LCD ekrana, a sve kako bi se prikrilo nepredvidivo ispisivanje znakova pri početnom zaprimanju podataka putem RS232 serijske komunikacije.

Početkom beskonačne petlje koja slijedi počinje i dio koda koji će se izvršavati tijekom cijelog rada mikrokontrolera. Prvi se unutar petlje nazire upit „if Usart_Data_Ready = 1“ čime se propitkuje jesu li zaprimljeni ikakvi podaci i ukoliko jesu izvršava se kod unutar upita. Sada slijedi izraz „Usart_Read_text(txt,'OK')“ koji upućuje mikrokontroler na čitanje podataka zaprimljenih putem serijske komunikacije sve do izraza „OK“ i zatim njihovo spremanje u znakovni niz txt. Taj znakovni niz sada je potrebno izrazom „StrToInt(txt)“ pretvorit u varijablu tipa integer i pridodat tu vrijednost varijabli napon, kako bi zatim varijablu napon mogli pomnožit sa 5 i dobiti broj koji

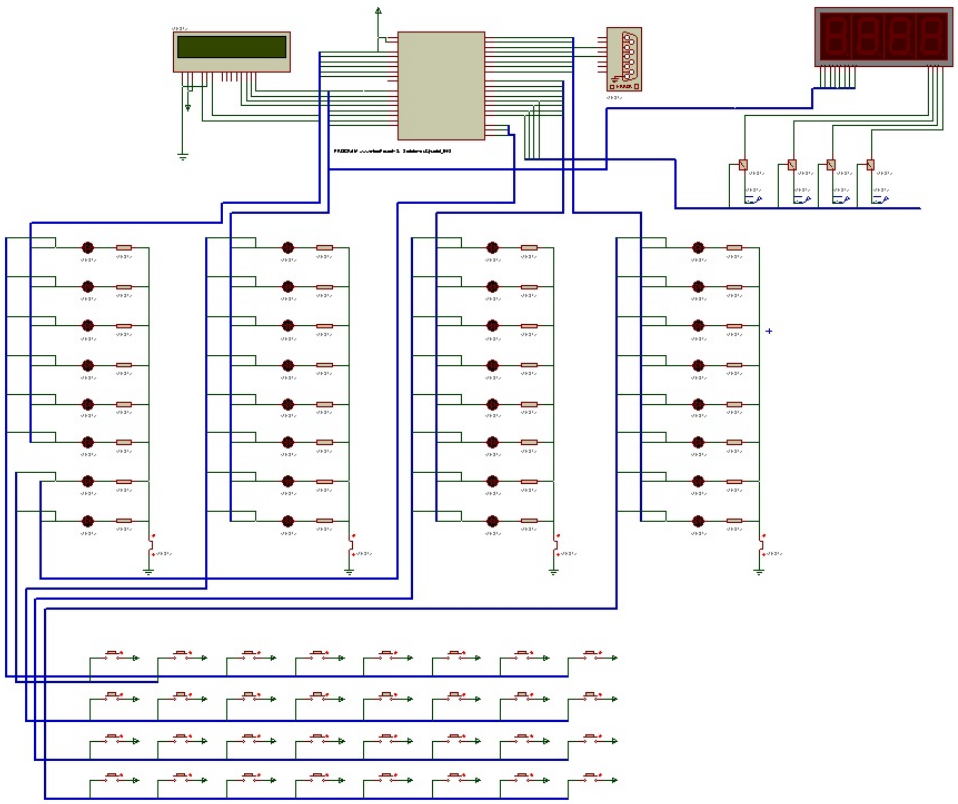
predstavlja pad napona između 0V i 5V zapisan u mV. Taj broj opet se sprema u varijablu napon, koju je potrebno pretvorit u znakovni niz txt kako bi se ona mogla ispisat na LCD ekranu. Ispisivanjem znakovnog niza txt na LCD ekranu izrazom „lcd_out(2,1,txt)“ kompletirana je ova mikrokontrolerska aplikacija, te je izvršen njezin ciljani zadatak.

```
program Mjerenje_napona_primatelj;   //NAZIV PROJEKTA
var txt: string[6];   //DEKLARIRANJE 6-ZNAKOVNOG NIZA txt
var napon: integer;   //DEKLARIRANJE VARIJABLE napon TIPA integer
begin   //POČETAK GLAVNE FUNKCIJE
Usart_Init(2400);   // INICIRANJE SERIJSKE KOMUNIKACIJE PRI 2400 bps
Lcd_init(portb);   //INICIRANJE LCD EKRANA PREKO portb
LCD_Cmd(LCD_CURSOR_OFF);   //NAREDBA LCD EKRANU ZA ISKLJUČIVANJE
POKAZIVAČA
lcd_out(1,1,'Napon [mV]');   //NAREDBA ZA ISPISIVANJE NA LCD EKRANU
lcd_out(2,1,'                ');   //NAREDBA ZA ISPISIVANJE NA LCD EKRANU
repeat
  if Usart_Data_Ready = 1 then    //AKO SU ZAPRIMLJENI PODACI
    begin
      Usart_Read_text(txt,'OK');   //ČITANJE PODATAKA ZAPEIMLJENIH SERIJSKOM
KOMUNIKACIJOM
      napon := StrToInt(txt);   //PRETVARANJE ZNAKOVNOG NIZA txt U VARIJABLU
napon TIPA integer
      napon:=napon*5;
      IntToStr(napon,txt);   //PRETVARANJE VARIJABLE NAPON U ZNAKOVNI NIZ txt
      lcd_out(2,1,txt);   //NAREDBA ZA ISPISIVANJE NA LCD EKRANU
    end;
until  1=2
end.   //ZAVRŠETAK GLAVNE FUNKCIJE
```

5. Simulacija razvojnog sustava za mikrokontrolerske aplikacije

5.1 Razvojni sustav za mikrokontrolerske aplikacije

Razvojni sustav za mikrokontrolerske aplikacije, kako mu i samo ime kaže namijenjen je simulaciji mikrokontrolerskih aplikacija, u ovom slučaju pisanih u mikroPascalu. Korisnici mikroPascala i ostalih programskih prevoditelja suočeni su sa činjenicom da, ukoliko žele izrađivati ili samo simulirati i najjednostavnije mikrokontrolerske aplikacije, moraju imati programator za mikrokontrolere preko kojeg se kod ispisan u mikroPascalu ubacuje na sam mikrokontroler, ali i sve ostale periferne komponente s kojima se želi vršiti interakcija. Kako bi se izbjegla zavisnost o hardveru tijekom projektiranja mikrokontrolerskih sustava, odnosno njihovoj simulaciji, izrađen je razvojni sustav za simulaciju mikrokontrolerskih aplikacija koji je cijelim svojim dijelom izrađen unutar programskog paketa Proteus (Slika 5.1).

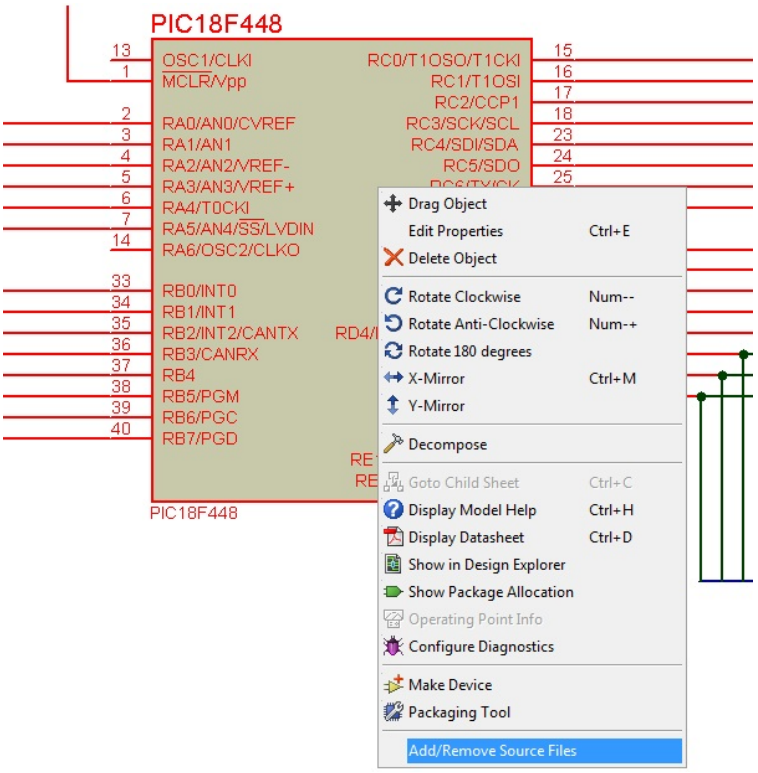


Slika 5.1 Razvojni sustav za mikrokontrolerske aplikacije

Najvažnija komponenta ovog razvojnog sustava jest sam mikrokontroler, u ovom slučaju već spomenuti mikrokontroler serije PIC18F448. Cijeli razvojni sustav zapravo je zamišljen kao niz perifernih jedinica istovremeno spojenih na odgovarajuće izvode mikrokontrolera, gdje korisnik ubacivanjem mikrokontrolerske aplikacije određuje s kojim će jedinicama u kojem trenutku vršiti interakciju.

Dakle, nakon što se ispiše željeni kod unutar mikroPascala i nakon što mikroPascal prevede taj kod, unutar datoteke tog mikroPascal projekta pojaviti će se datoteka naziva jednakog nazivu projekta, ali nastavka .hex. Ova datoteka predstavlja isti onaj kod ispisan od strane korisnika u mikroPascalu, no taj kod je sam mikroPascal preveo u jezik razumljiv mikrokontroleru i spremio ga u ovu datoteku. Naredbe unutar ove datoteke zapravo određuju funkciju mikrokontrolera.

Dodavanje datoteka nastavka .hex u sam mikrokontroler unutar Proteusa izvedeno je jednostavnim pritiskom desne tipke miša na označeni mikrokontroler (Slika 5.2) i odabirom opcije „Add/Remove Source files“ (*hrv. Dodaj/Ukloni izvornu datoteku*), nakon čega se otvara prozor unutar kojeg se odabire izvorna datoteka. Nakon odabira izvorne datoteke potrebno je samo pokrenuti simulaciju unutar Proteusa, i mikrokontroler počinje sa izvršavanjem svoje funkcije, odnosno interakcijom sa željenim perifernim komponentama.

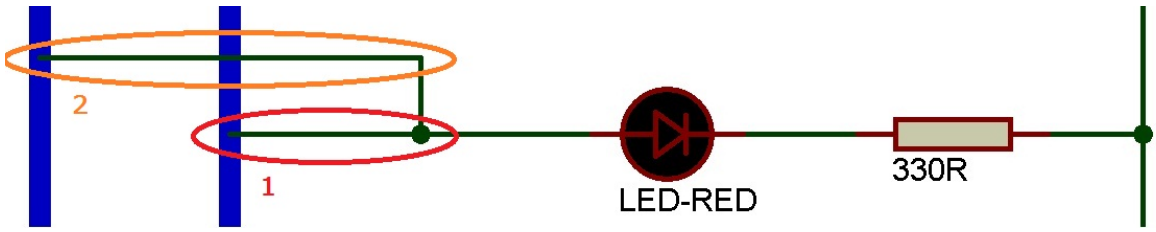


Slika 5.2 Dodavanje nove .hex datoteke u mikrokontroler

5.2 Periferne komponente razvojnog sustava za mikrokontrolerske aplikacije

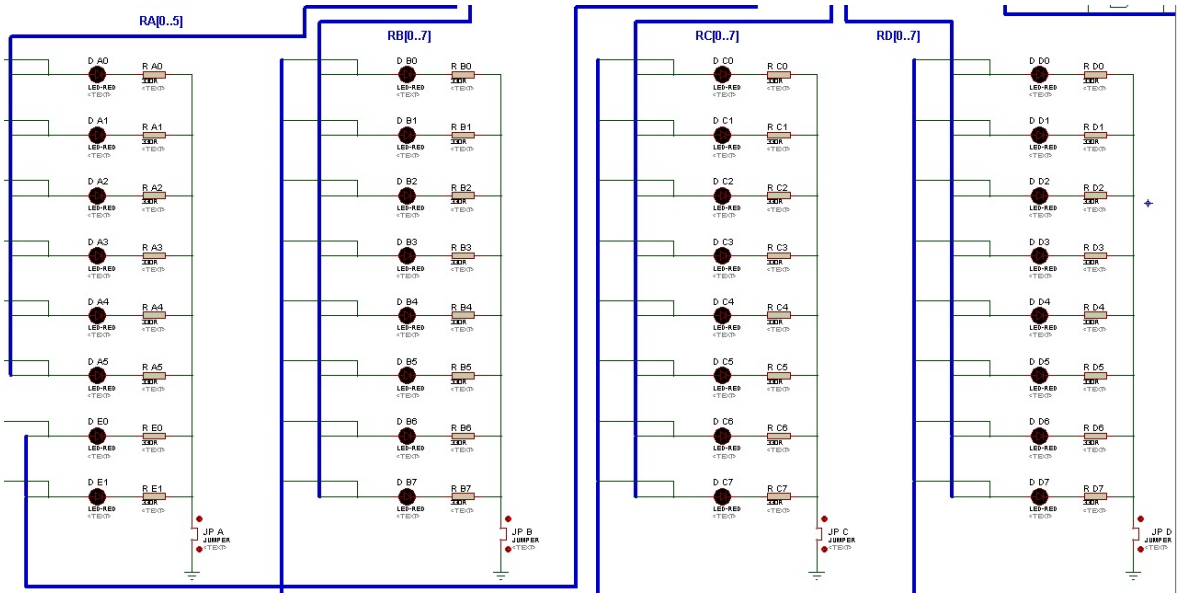
5.2.1 LED diode

LED diode su poluvodički svjetlosni elementi koje je potrebno propusno polarizirati kako bi svijetlile. Ta propusna polarizacija ostvaruje se preko podatkovnih izlaza korištenog mikrokontrolera. Anode ovih dioda spojene su sabirnicama na odgovarajuće izvode mikrokontrolera kao što je označeno brojem 1 (Slika 5.3), ali su spojene i na tipkala, također preko sabirnica, kao što je označeno brojem 2 (Slika 5.3). Sama tipkala detaljnije će biti objašnjena u poglavlju 5.2.2 Tipkala. Ukoliko se na anodu neke od dioda preko odgovarajućeg izvoda mikrokontrolera ili preko odgovarajućeg tipkala dovede logička 1 ta će dioda biti propusno polarizirana i svijetliti će.



Slika 5.3 Način spajanja LED diode

Na shemi razvojnog sustava za mikrokontrolerske aplikacije (Slika 5.4), LED diode su podijeljene u četiri skupine po osam dioda. Gledajući sa lijeva na desno i od gore prema dolje (Slika 5.4), u prvoj skupini nalaze se diode koje su spojene na izvode RA0-RA5 i RE0-RE1. U drugoj skupini nalaze se diode spojene na izvode RB0-RB7, u trećoj skupini diode spojene na RC0-RC7, dok se u četvrtoj skupini nalaze diode spojene na izvode RD0-RD7. Svaka od ovih dioda, dakle, ovisi o izvodu mikrokontrolera na koji je spojena. Kako bi se na diodu uopće mogla dovesti logička 1, izvod na koji je spojena mora imati ulogu izlaza mikrokontrolera, što se određuje TRIS registrima spomenutima u poglavlju 4.2 PIC18F448 mikrokontroler.

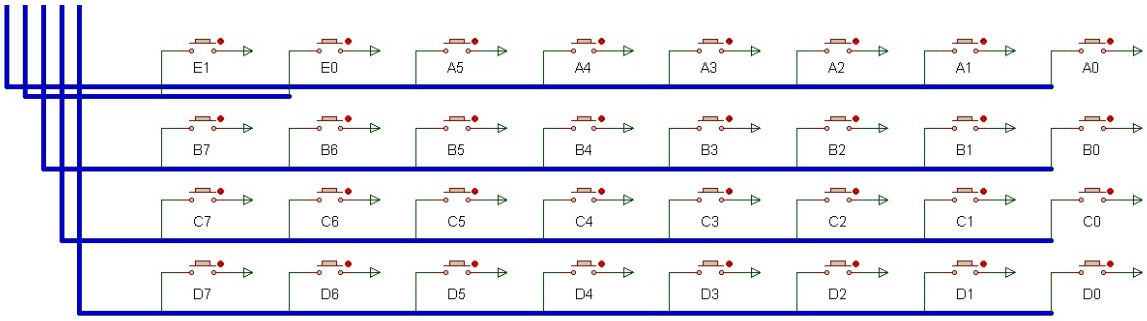


Slika 5.4 LED diode na shemi razvojnog sustava za mikrokontrolerske aplikacije

Način na koji su ove diode spojene daje im ulogu svojevrsnih indikatora koji u svakom trenutku pokazuju koji je izlaz mikrokontrolera aktivan ili koja su tipkala u tom trenutku stisnuta, ali je pritom s njima moguće ostvariti i različite interakcije preko različitih mikrokontrolerskih aplikacija.

5.2.2 Tipkala

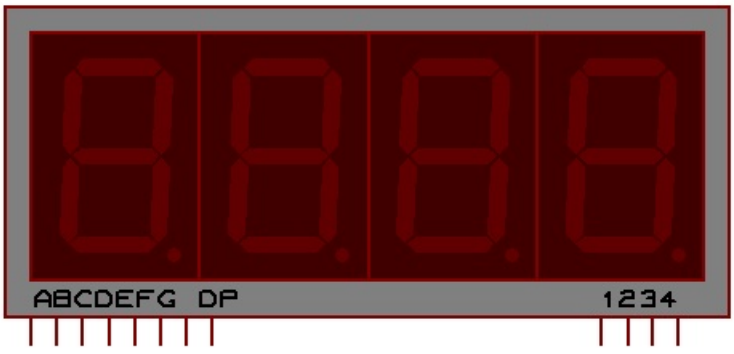
Tipkala su, kao i diode, spojena na svaki od podatkovnih izvoda mikrokontrolera, ali ne direktno. Izvod svakog od ovih tipkala sabirnicama je spojen na anodu odgovarajuće mu diode (Slika 5.5), preko izvoda označenog brojem 2 (Slika 5.3). Drugi izvod svakog tipkala spojen je na izvor od +5V istosmjernog napona. To znači da će, ukoliko je tipkalo stisnuto, dioda svijetliti. Još važnije za napomenuti jest da se dovođenjem pozitivnog napona na anodu diode, preko izvoda označenog sa brojem 2 (Slika 5.3), ujedno taj pozitivan napon preko izvoda označenog sa brojem 1 (Slika 5.3) preko sabirnice dovodi na izvod mikrokontrolera na koji je spojena ta dioda. Ukoliko korisnik želi koristiti tipkala kao elemente dodatne interakcije sa sustavom tijekom same simulacije, najprije je potrebno postaviti odgovarajuće izvode mikrokontrolera u ulogu ulaza, a zatim i postaviti uvjet unutar mikrokontrolerske aplikacije, da se ukoliko je tipkalo pritisnuto, odnosno ukoliko je na ulaz mikrokontrolera na koji je spojeno to tipkalo dovedena logička 1, neki proces pokrene, izvrši ili zaustavi.



Slika 5.5 Tipkala na shemi razvojnog sustava za mikrokontrolerske aplikacije

5.2.3 Multipleksirani 7-segmentni ekrani

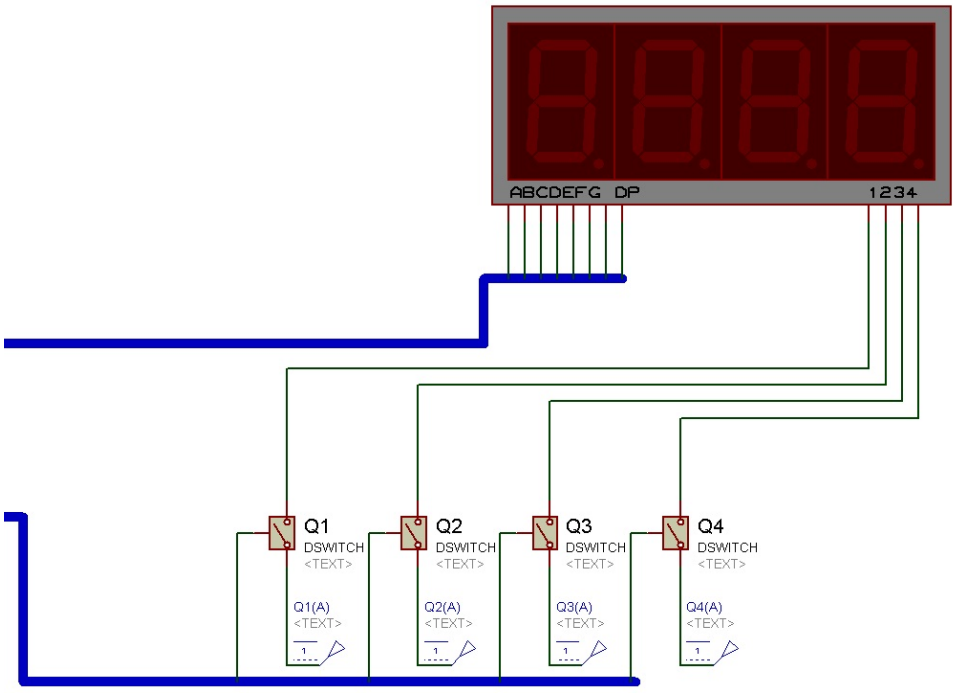
Multipleksirani 7-segmentni ekrani su kao periferne komponente razvojnog sustava predstavljeni su samo jednom komponentom (Slika 5.6) koja se nalazi u knjižnici komponenti Proteus programskog paketa. Ova komponenta sadrži četiri 7-segmentna ekrana, od kojih je svaki segment predstavljen jednom LED diodom. Svaki ekran sadrži sedam segmenata, odnosno LED dioda, za prikaz broja te dodatni segment za prikaz decimalne točke. Izvodi svakog od segmenata, sva četiri ekrana, spojeni su na zajedničku sabirnicu, čiji su izvodi predstavljeni slovima A-G i izvodom DP koji predstavlja izvod diode decimalne točke. Izvodi označeni brojevima 1, 2, 3 i 4 predstavljaju aktivacijske ulaze svakog od ekrana, krenuvši s lijeva na desno.



Slika 5.6 Komponenta multipleksiranih 7-segmentnih ekrana

Izvodi komponente multipleksiranih 7-segmentnih ekrana označeni slovima od A do G i izvod DP spojeni su redom preko zajedničke sabirnice na RB0-RB7 izvode mikrokontrolera (Slika 5.7). Već je poznato da svaki od ovih izvoda predstavlja izvod diode jednog od segmenata unutar ekrana. Kako bi ta dioda svijetlila na njen je izvod potrebno preko izlaza mikrokontrolera dovesti logičku 1.

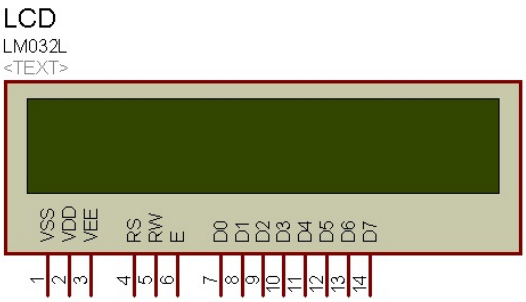
Za prikaz nekog željenog broja na jednom od ekrana potrebno je odrediti odgovarajuću kombinaciju logičkih 0 i 1, kako bi svijetlile upravo diode za prikaz tog broja. Problem kod multipleksiranih 7-segmentnih ekrana je zajednička sabirnica za sva četiri ekrana, preko koje se u jednom trenutku može dovesti samo jedna kombinacija logičkih 0 i 1 za prikaz broja na ekranu. Problem zajedničke sabirnice rješava se naizmjeničnim aktiviranjem i deaktiviranjem ovih ekrana preko njihovih aktivacijskih ulaza. Na svaki od ova četiri aktivacijska ulaza, od kojih svaki predstavlja jedan ekran postavljene su digitalne sklopke, Q1-Q4 (Slika 5.7) koje su zatvorene samo ukoliko je na njihov kontrolni ulaz dovedena logička 1. Ovi kontrolni ulazi sklopki Q1-Q4 spojeni su, preko sabirnice, redom na izvode mikrokontrolera RD4-RD7 i preko njih se vrši aktivacija. Na drugi izvod ovih sklopki spojen je izvor koji stalno daje logičku 1. Ta logička 1 doći će na aktivacijski ulaz željenog ekrana, odnosno aktivirat će željeni ekran, samo ukoliko je sklopka na koju je spojen zatvorena. U trenutku kada se pošalje sa izvoda RB0-RB7 neka željena kombinacija 0 i 1, preko aktivacijskog ulaza aktivira se samo ekran za koji je namijenjena ta kombinacija. Ekran se drži aktiviran neko optimalno vrijeme koje je dovoljno da podaci stignu na ekran, ali i da čovjekovo oko ipak ne primijeti aktiviranje i deaktiviranje ekrana.



Slika 5.7 Način spajanja komponente multipleksiranih 7-segmentnih ekrana

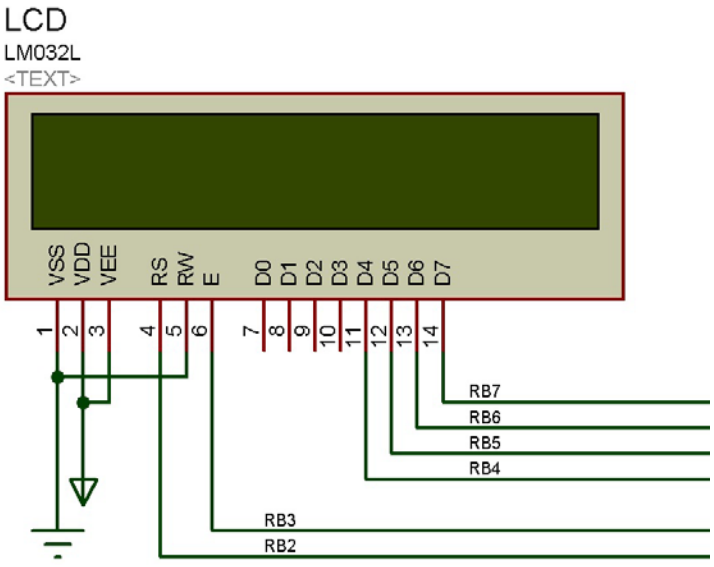
5.2.4 LCD ekran

LCD ekran za prikaz standardni znakova vjerojatno je najviše komponenta za vizualni prikaz podatak. U knjižnici komponenti Proteus programskog paketa nalazi se više izvedbi LCD ekrana. Za potrebe razvojnog sustava za mikrokontrolerske aplikacije odabrana je komponenta LCD ekrana veličine 20x2, odnosno sa dva retka i po dvadeset znakova u svakom retku (Slika 5.8).



Slika 5.8 Komponenta LCD ekrana

Komunikacija LCD ekrana sa mikrokontrolerom vrši se preko 4-bitne sabirnice (Slika 5.9), gdje su izvodi mikrokontrolera, označeni sa RB4-RB7, spojeni na izvode LCD ekrana označene sa D4-D7. Nazivi ostalih izvoda mikrokontrolera, njihova uloga i oznaka izvoda mikrokontrolera na koji su spojeni navedeni su u tabeli (Tabela 6), dok je način na koji su spojeni prikazan na slici (Slika 5.9).



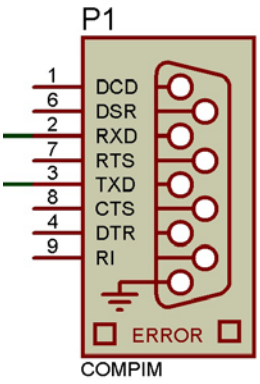
Slika 5.9 Način spajanja LCD ekrana

Naziv izvoda LCD ekrana	Značenje izvoda LCD ekrana	Izvod na koji je spojen
Vss	Uzemljenje	Uzemljenje
Vdd	Napajanje	+5V
Vee	Napon između Vss i Vdd	+5V
RS	Odabir registra (1 = podatkovni ulaz , 0 = instrukcijski ulaz)	RB2 (izvod mikrokontrolera)
RW	Način rada - Read/Write (*hrv. Čitaj/Zapiši*) (1 = očitaj podatke sa LCD ekrana, 0 = zapiši podatke)	Uzemljenje
E	Aktivacijski signal	RB3

Tabela 6 Naziv, značenje i izvod mikrokontrolera na koji je spojen pojedini izvod LCD ekrana

5.2.5 RS232 serijska komunikacija

RS232 serijska komunikacija omogućava prijenos podataka u binarnom obliku i često se koristi za komunikaciju između mikrokontrolera i računala. Komponenta unutar Proteusa koja omogućava serijsku komunikaciju sa realnom okolinom je komponenta COMPIM (Slika 5.10). Ova komponenta zapravo predstavlja priključak serijskog kabela sa 9 izvoda u simulacijskoj okolini Proteusa.

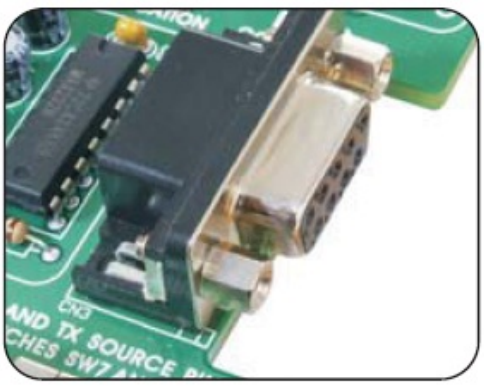


Slika 5.10 Komponenta COMPIM

Od 9 izvoda ove komponente, za potrebu serijske komunikacije realne okoline i razvojnog sustava za mikrokontrolerske aplikacije bitna su samo dva. Izvod RX je onaj izvod preko kojeg podaci idu iz smjera Proteusa prema realnoj okolini i taj je izvod spojen na RC7 izvod mikrokontrolera. Drugi izvod je TX i njime se podaci iz realne okoline šalju na razvojni sustav za mikrokontrolerske aplikacije unutar Proteusa. Jedan od priključaka je, dakle, komponenta COMPIM koja predstavlja fizički priključak 9-izvodnog serijskog kabela (Slika 5.11) na računalo. Na drugi kraj ovog kabela moguće je priključiti bilo koju komponentu koja je u mogućnosti ostvariti serijsku komunikaciju.

Slika 5.11 Kabel za serijsku komunikaciju [S6]

U ovom slučaju na drugi je kraj kabela priključen već spomenuti razvojni sustav EasyPIC5, odnosno njegov RS232 modul za serijsku komunikaciju (). RX i TX izvodi ovog modula spojeni su na mikrokontroler koji se nalazi na tom razvojnom sustavu, odnosno redom na njegove izvode RC7 i RC6.



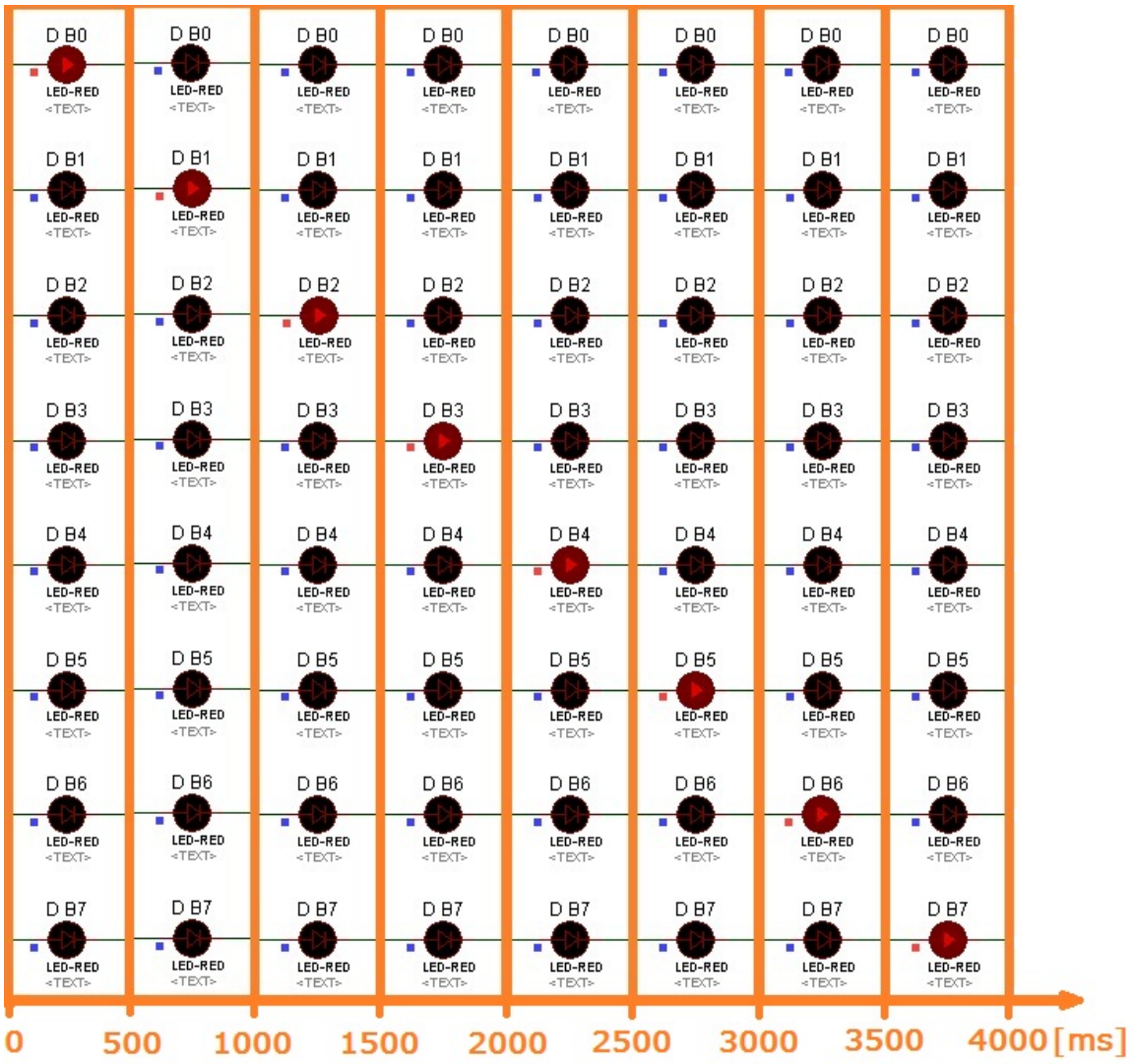
Slika 5.12 RS232 modul na razvojnom sustavu EasyPIC5 [S7]

5.3 Prikaz simulacije komponenti razvojnog sustava pomoću odgovarajućih mikrokontrolerskih aplikacija

5.3.1 „Trčeće diode“

Mikrokontrolerska aplikacija kojom se izvode takozvano „trčanje“ dioda, detaljno je opisana u poglavlju 4.3 „Trčeće“ diode. LED diode, koje predstavljaju periferne komponente za čiju je namjenu pisana ova mikrokontrolerska aplikacija, detaljno su opisane u poglavlju 5.2.1 LED diode.

Nakon detaljnog opisa spomenute mikrokontrolerske aplikacije i LED dioda, vrijedi slikovno prikazati samo izvođenje ove aplikacije unutar razvojnog sustava simuliranog u Proteusu, odnosno ponašanje LED dioda tijekom tog izvođenja (Slika 5.13).



Slika 5.13 Simulacija mikrokontrolerske aplikacije "Trčeće diode"

5.3.2 Brojač 0-1000

Brojanje od 0 do 1000 izvodi se na komponenti multipleksiranih 7-segmentnih ekrana, koja je ujedno i periferna komponenta razvojnog sustava za mikrokontrolerske aplikacije, opisana u poglavlju 5.2.3 Multipleksirani 7-segmentni ekrani. Mikrokontrolerska aplikacija čiji se kod izvršava unutar mikrokontrolera tijekom izvođenja simulacije ove komponente, opisan je u poglavlju 4.4 Brojač 0-1000.

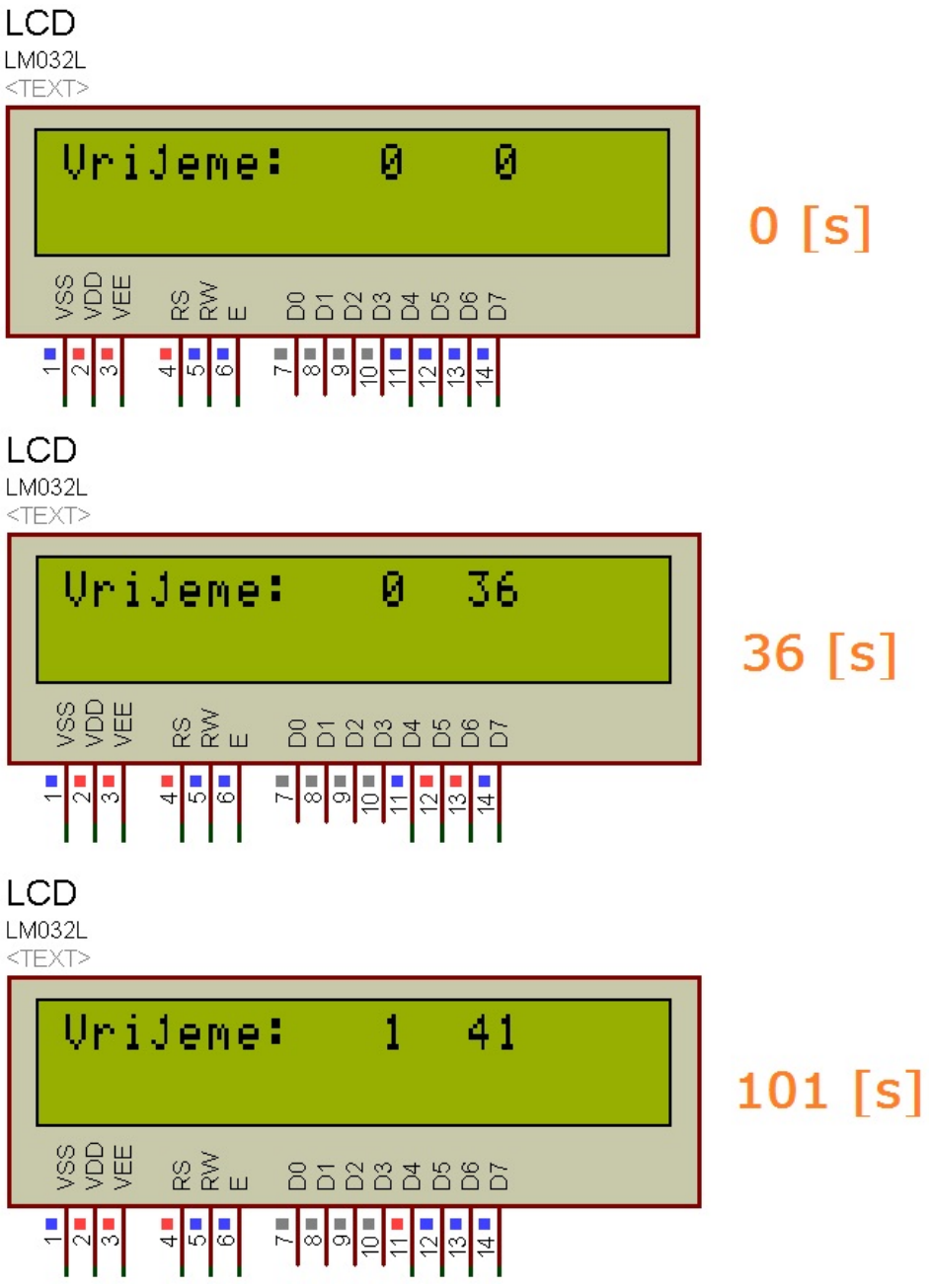
Proizvoljno određeni trenuci simulacije ove komponente unutar razvojnog sustava za mikrokontrolerske aplikacije i stanja njezinih ekrana u tim trenucima prikazani su na slici (Slika 5.14).

Slika 5.14 Simulacija mikrokontrolerske aplikacije "Brojač 0-1000"

5.3.3 Brojač vremena

Brojanje vremena izvodi se mikrokontrolerskom aplikacijom „Brojač vremena“, opisanom u poglavlju 4.5 Brojač vremena. Ovom aplikacijom ujedno se to brojanje i prikazuje na LCD ekranu, kao dijelu razvojnog sustava za mikrokontrolerske aplikacije koji je opisan u poglavlju 5.2.4 LCD ekran.

Proizvoljni trenuci izvođenja spomenute mikrokontrolerske aplikacije, tijekom simulacije razvojnog sustava unutar Proteusa prikazani su na slici (Slika 5.15).



Slika 5.15 Simulacija mikrokontrolerske aplikacije „LCD ekran“

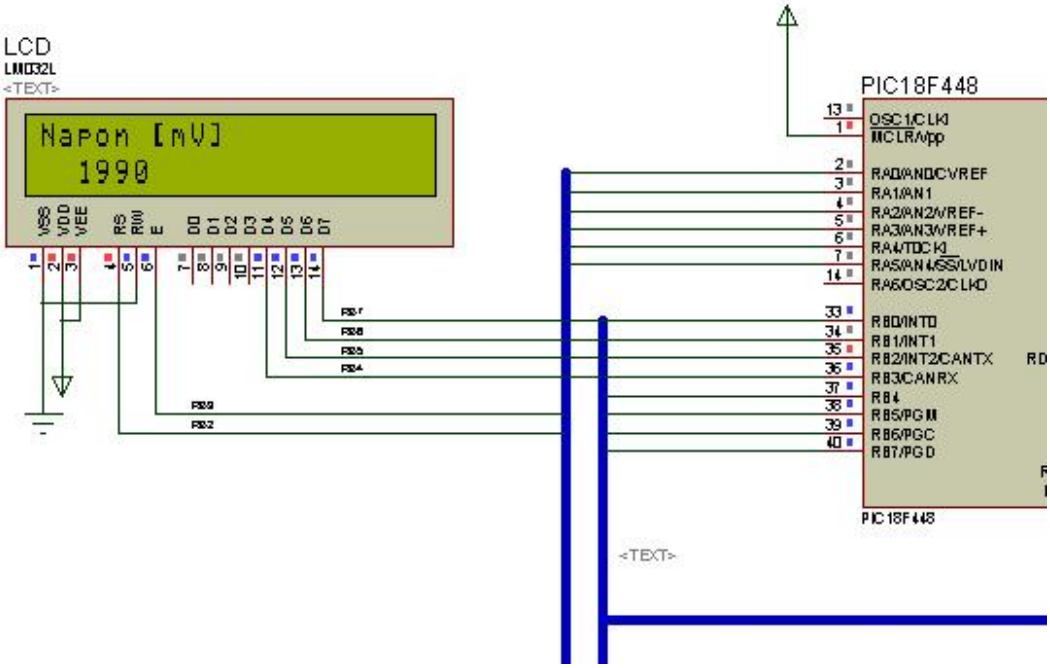
5.3.4 Mjerenje napona u realnom okruženju

Mjerenje napona u realnom okruženju izvodi se kako bi se prikazala mogućnost Proteus programskog paketa da komunicira sa realnim okruženjem preko RS232 serijske komunikacije. Ova vrsta komunikacije, skupa sa načinom na koji je spojena na razvojni sustav za mikrokontrolerske aplikacije, prikazana je u poglavlju 5.2.5 RS232 serijska komunikacija.

Mjerenje napona se vrši uz korištenje dva mikrokontrolera, jednog u realnom okruženju i jednog u okruženju Proteusa. Mikrokontroler u realnom okruženju izvodi mikrokontrolersku aplikaciju opisanu u poglavlju 4.6.1 Mikrokontroler u realnom okruženju, a njegov zadatak je mjerenje promjene pada napona na jednom od svojih ulaza čija se vrijednost mijenja zakretanjem potenciometra sadržanog na razvojnom sustavu EasyPIC5, na kojem se nalazi i sam mikrokontroler. Uz to, ovaj mikrokontroler ima zadaću ispisivati tu promjenu na LCD ekranu (Slika 5.16) i slati ju preko modula za serijsku komunikaciju.

Slika 5.16 LCD ekran na razvojnom sustavu EasyPIC5

Mikrokontroler u okruženju Proteus programskog paketa, čije su naredbe zadane mikrokontrolerskom aplikacijom opisanom u poglavlju 4.6.2 Mikrokontroler u okruženju Proteus programskog paketa, ima zadatak zaprimanja podataka poslanih iz realnog okruženja putem serijske komunikacije, te ispisivanje tih podataka na LCD ekranu, smještenom na razvojni sustav za mikrokontrolerske aplikacije. Podatke koji su zaprimljeni serijskom komunikacijom, prije prikazivanja na LCD ekranu, ovaj mikrokontroler množi sa brojem 5 kako bi se vrijednost prikazanog broja mogla izraziti u mV.



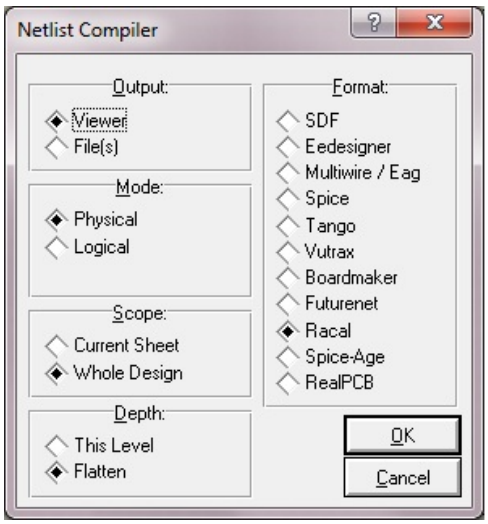
Slika 5.17 LCD ekran na razvojnom sustavu za mikrokontrolerske aplikacije

6. Programska veza između razvojne okoline programskog paketa Proteus i razvojne okoline udaljenog laboratorija

6.1 Netlist prevoditelj

Netlist prevoditelj je opcija Proteus programskog paketa koja omogućava generiranje tekstualne datoteke, unutar koje se nalazi popis međusobnog spajanja svih izvoda na trenutno korištenoj shemi. Ta datoteka naziva se Netlist datoteka i ona je zapravo popis mreža, gdje mreža predstavlja grupu međusobno spojenih izvoda.

Ovoj se opciji Proteusa pristupa naredbom „Netlist Compiler“ (*hrv. Netlist prevoditelj*), unutar izbornika „Tools“ (*hrv. Alati*). Nakon toga se otvara prozor u kojem su ponuđene različite opcije pri generiranju netlist datoteke (Slika 6.1).



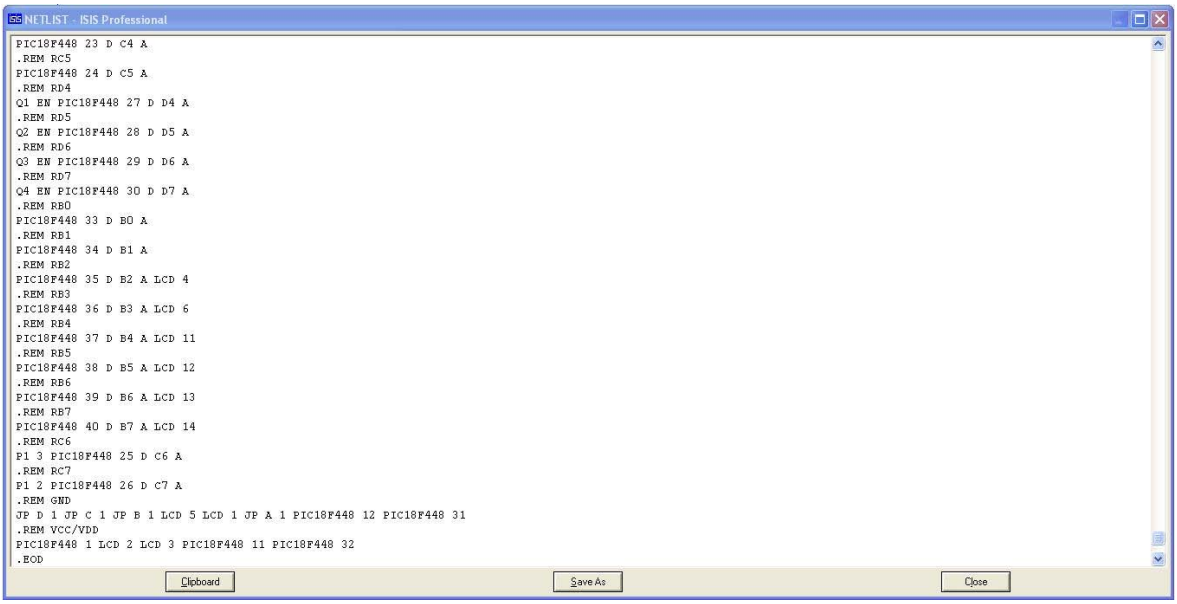
Slika 6.1 Različite opcije za generiranje Netlist datoteke

U polju nazvanom Mode (*hrv. Mod*) moguće je odabrati fizički ili logički zapis izvoda. U fizičkom su zapisu izvodi komponenti predstavljeni njihovim brojevima, dok su u logičkom zapisu izvodi predstavljeni svojim imenima. Međutim, među ovim opcijama, najviše pažnje potrebno je posvetiti polju Format kojim se odabire format zapisa unutar ove datoteke. Više je različitih formata kako bi netlista bila korisna i za programe drugih proizvođača. Od ostalih opcija bitno je spomenuti i polje Scope (*hrv. opseg*). Ono nam nudi dvije opcije od kojih je

uobičajena „Whole design“ (*hrv. Cijeli dizajn*) i njome je u generiranoj netlisti obuhvaćen cijeli dizajn. Kod opcije „Current sheet“ (*hrv. Trenutna podloga*) unutar netliste će se naći samo podloga na koju korisnik trenutno ima pogled, dok će hijerarhijski različite podloge bit izostavljene.

Nakon što su odabrane željene opcije, potrebno je samo još pritisnuti tipku „OK“ i netlista je generirana (Slika 6.2), te ju je nakon toga moguće spremiti pritiskom na tipku „Save as“ (*hrv. Spremi kao*).

Opcije korištene pri generiranju netliste korištene u poglavlju 6.3 Programska realizacija veze između razvojne okoline programskog paketa Proteus i razvojne okoline udaljenog laboratorija, označene su na slici (Slika 6.1), dok je izgled formata imena Racal koji je pritom korišten prikazan na drugoj slici ().



Slika 6.2 Generirana netlista

6.2 Veza između razvojne okoline programskog paketa Proteus i razvojne okoline udaljenog laboratorija

Cijela ideja veze između razvojne okoline programskog paketa Proteus i razvojne okoline udaljenog laboratorija sastoji se od tri koraka. U prvom koraku korisnik nacrta shemu unutar Proteus programskog paketa u kojoj je na neki mikrokontroler spojena jedna ili više komponenti koja se može naći i u sustavu udaljenog laboratorija, te generira netlistu te sheme. U drugom se koraku ta netlista čita aplikacijom pisanom u C programskom jeziku koja ju zatim i prevodi u format kojim se služi kontroler za prespajanje u sustavu udaljenog laboratorija.

Nakon što je kontroler zaprimio potrebne naredbe u odgovarajućem formatu može se izvesti treći korak, a to je prespajanje i izvođenje sheme iz Proteusa u sustavu udaljenog laboratorija.

Komponenta koja će biti korištena pri prikazu funkcioniranja veze ovih razvojnih okolina je LCD ekran spojen na mikrokontroler. Ovaj je LCD ekran spojen na neke izvode mikrokontrolera, koje je u bilo kojem drugom slučaju korisnik mogao spojiti na bilo koje izvode mikrokontrolera. Sada je pitanje na koje je izvode spojen LCD ekran u sustavu udaljenog laboratorija i na koji se način kontroleru za prespajanje daju naredbe. Oblik naredbi koje očekuje kontroler za prespajanje u sustavu udaljenog laboratorija slijedećeg su izgleda:

sxxxyyyzzz

Naredbe za prespajanje počinju slovom s, što označava naredbu spoji. Sljedeće tri znamenke, označene s xxx, kazuju koji analogni multipleksor unutar osnovnog modula za Softwireing želimo aktivirati. Druge tri znamenke, označene s yyy, određuju koji kanal odabranog analognog multipleksora želimo spojiti sa njegovim zajedničkim izvodom. Posljednje tri znamenke, označene sa zzz, definiraju na koji izvod sabirnice želimo spojiti prethodno definirani signal. Ulazni argumenti xxx kreću se u rasponu od 001 do 040, za yyy u rasponu od 001 do 005, a za zzz u rasponu od 001 do 008[4].

Korisnik je unutar Proteusa, kako je već rečeno, slobodan spojiti, u ovom slučaju LCD ekran, na bilo koji izvod mikrokontrolera. Ovisno o rednom broju izvoda mikrokontrolera na koji je spojen neki izvod LCD ekrana, aplikacijom pisanom u C programskom jeziku, potrebno je promijeniti broj xxx (Tabela 7) u ovisnosti o rednom broju izvoda mikrokontrolera, yyy u slučaju LCD ekrana uvijek ostaje isti (004), dok je argument zzz potrebno promijeniti ovisno o rednom broju izvoda LCD ekrana (Tabela 8). Argumente izmijenjene na prethodni način sada je moguće poslati kontroleru za prespajanje u zapisu sxxxyyyzzz, nakon čega će se izvršiti i samo prespajanje.

Izvod mikrokontrolera	xxx (Udaljeni laboratorij)	Izvod mikrokontrolera	xxx (Udaljeni laboratorij)
1	0	21	26
2	1	22	27
3	2	23	20
4	3	24	21
5	4	25	22
6	5	26	23
7	6	27	28
8	32	28	29
9	33	29	30
10	34	30	31
11	NAP+	31	NAP+
12	NAP-	32	NAP-
13	0	33	8
14	7	34	9
15	16	35	10
16	17	36	11
17	18	37	12
18	19	38	13
19	24	39	14
20	25	40	15

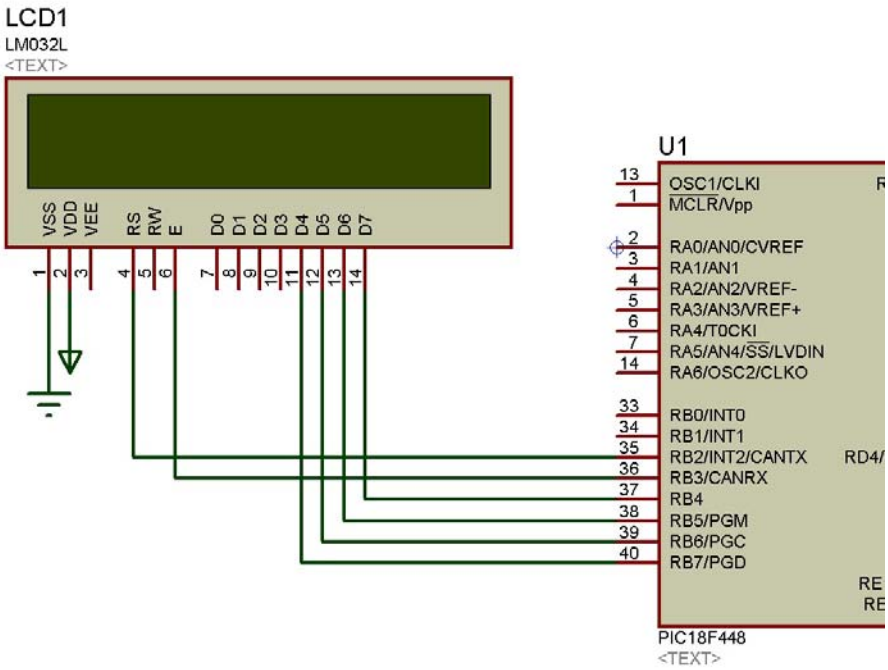
Tabela 7 Promjena argument xxx u ovisnosti o rednom broju izvoda mikrokontrolera

Izvod LCD ekrana	xxx (Udaljeni laboratorij)	Izvod LCD ekrana	xxx (Udaljeni laboratorij)
4	3	10	-
5	-	11	5
6	4	12	6
7	-	13	7
8	-	14	8
9	-		

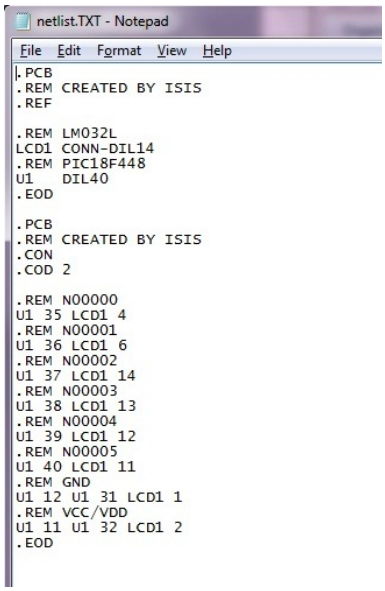
Tabela 8 Promjena argument zzz u ovisnosti o rednom broju izvoda LCD ekrana

6.3 Programska realizacija veze između razvojne okoline programskog paketa Proteus i razvojne okoline udaljenog laboratorija

U svrhu prikazivanja programske realizacije, na mikrokontroler PIC18F448 spojen je LCD ekran (Slika 6.3) identičan onome u poglavlju 5.2.4 LCD ekran, te je generirana netlista tog spoja (Slika 6.4).



Slika 6.3 Spoj LCD ekrana u svrhu prikazivanja programske realizacije



Slika 6.4 Generirana netlista

Prespajanje ovog spoja u sustavu udaljenog laboratorija izvršeno je programskom aplikacijom pisanom u C programskom jeziku čiji je kod prikazan na kraju ovog poglavlja. Ova se aplikacija pokreće dvostrukim klikom na datoteku „Aplikacija_za_prevodenje.exe“. Nakon što je otvorena, aplikacija traži upis lokacije .txt datoteke koju je potrebno prevest u format prepoznatljiv kontroleru za prespajanje unutar udaljenog laboratorija, a zatim i lokaciju na koju će se spremit prevedena .txt datoteka. Ukoliko su upisane nepostojeće lokacije aplikacija ispisuje izraz „Datoteka ne postoji!“.

U C programskom jeziku čitanje podataka iz formatirane datoteke vrši se funkcijom fscanf(). Ona čita zadani broj znakova nakon trenutne pozicije zamišljenog pokazivača unutar datoteke i vraća vrijednost količine vraćenih varijabli. Ukoliko se pročitani izraz ne poklapa sa zadanim, zbog while funkcije fscanf() ide dalje sve dok ne dođe do odgovarajućeg izraza. Nakon funkcije while započinje funkcija for unutar koje se nalazi ostatak koda. Taj će se kod izvršiti šest puta jer LCD ekran i mikrokontroler imaju šest međusobno spojenih izvoda. Unutar for petlje najprije se u prevedenu datoteku sprema znak „s“ kao početak izraza sxxxyyyzzz. Switch funkcijom koja slijedi provjerava se vrijednost varijable izvod_uC, čija je vrijednost prethodno bila pročitana iz datoteke funkcijom fscanf(). Kako varijabla izvod_uC predstavlja izvod mikrokontrolera, u odnosu na njenu vrijednost odabire se određeni case (*hrv. slučaj*) i pretvara u tom izvodu odgovarajuću vrijednost xxx, te ju zapisuje u datoteku. Nakon xxx u datoteku se na mjesto yyy upisuje izraz 004. Slično kao i za vrijednost varijable izvod_uC vrši se provjera i za vrijednost varijable izvod_LCD, kao i odabir odgovarajućeg slučaja u odnosu na njenu vrijednost. Vrijednost te varijable zatim se upisuje u datoteku nakon yyy, na mjesta namijenjena argumentu zzz čime je kompletiran izraz sxxxyyyzzz za spoj jednog izvoda mikrokontrolera sa izvodom LCD ekrana i for petlja kreće sa slijedećim izvršavanjem.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
int main()   //POČETAK GLAVNE FUNKCIJE
{
int izvod_uC, izvod_LCD, i;   //DEKLARIRANJE VARIJABLI
char otvaranje[150], spremanje[150];   //DEKLARIRANJE POLJA
printf("Upišite lokaciju datoteke koju želite prevest:");   //ISPIS
scanf("%s", otvaranje);   //SPREMANJE UPISANOG U ZNAKOVNI NIZ otvaranje
printf("Upišite lokaciju na koju želite spremiti prevedenu datoteku:");
```

```
scanf("%s", spremanje);   //SPREMANJE UPISANOG U ZNAKOVNI NIZ zatvaranje
FILE *f_citanje, *f_pisanje;   //DEKLARIRANJE POKAZIVAČA NA DATOTEKU
f_citanje=fopen(otvaranje,"r");   //POKAZIVAČ NA MJESTO OTVARANJA
f_pisanje=fopen(spremanje,"w");   //POKAZIVAČ NA MJESTO SPREMANJA
if(f_citanje==NULL || f_pisanje==NULL)   //UKOLIKO JE DOŠLO DO GREŠKE PRI
OTVARANJU
    {     printf("\nDatoteka ne postoji!\n");   //ISPIS    }
for (i=0;i<6;i++) {
while(fscanf(f_citanje, "U1 %d LCD1 %d\n", &izvod_uC, &izvod_LCD) !=2 )
{fscanf(f_citanje, "%*c\n");}   //PRESKAKANJE NEPOTREBNIH IZRAZA U DATOTECI
fprintf(f_pisanje, "s");   //UPISIVANJE ZNAKA S U DATOTEKU
switch(izvod_uC)
     {  case 1 : fprintf(f_pisanje, "000"); break;
        case 2 : fprintf(f_pisanje, "001"); break;
        case 3 : fprintf(f_pisanje, "002"); break;
        case 4 : fprintf(f_pisanje, "003"); break;
        case 5 : fprintf(f_pisanje, "004"); break;
        case 6 : fprintf(f_pisanje, "005"); break;
        case 7 : fprintf(f_pisanje, "006"); break;
        case 8 : fprintf(f_pisanje, "032"); break;
        case 9 : fprintf(f_pisanje, "033"); break;
        case 10 : fprintf(f_pisanje, "034"); break;
        case 11 : fprintf(f_pisanje, "000"); break;
        case 12 : fprintf(f_pisanje, "000"); break;
        case 13 : fprintf(f_pisanje, "000"); break;
        case 14 : fprintf(f_pisanje, "007"); break;
        case 15 : fprintf(f_pisanje, "016"); break;
        case 16 : fprintf(f_pisanje, "017"); break;
        case 17 : fprintf(f_pisanje, "018"); break;
        case 18 : fprintf(f_pisanje, "019"); break;
        case 19 : fprintf(f_pisanje, "024"); break;
        case 20 : fprintf(f_pisanje, "025"); break;
        case 21 : fprintf(f_pisanje, "026"); break;
        case 22 : fprintf(f_pisanje, "027"); break;
        case 23 : fprintf(f_pisanje, "020"); break;
        case 24 : fprintf(f_pisanje, "021"); break;
        case 25 : fprintf(f_pisanje, "022"); break;
        case 26 : fprintf(f_pisanje, "023"); break;
        case 27 : fprintf(f_pisanje, "028"); break;
        case 28 : fprintf(f_pisanje, "029"); break;
        case 29 : fprintf(f_pisanje, "030"); break;
        case 30 : fprintf(f_pisanje, "031"); break;
        case 31 : fprintf(f_pisanje, "000"); break;
        case 32 : fprintf(f_pisanje, "000"); break;
        case 33 : fprintf(f_pisanje, "008"); break;
        case 34 : fprintf(f_pisanje, "009"); break;
        case 35 : fprintf(f_pisanje, "010"); break;
        case 36 : fprintf(f_pisanje, "011"); break;
        case 37 : fprintf(f_pisanje, "012"); break;
        case 38 : fprintf(f_pisanje, "013"); break;
        case 39 : fprintf(f_pisanje, "014"); break;
        case 40 : fprintf(f_pisanje, "015"); break;     }
fprintf(f_pisanje, "004");   //UPISIVANJE IZRAZA 004 U DATOTEKU
switch(izvod_LCD)
     {  case 1 : fprintf(f_pisanje, "000\n"); break;
        case 2 : fprintf(f_pisanje, "000\n"); break;
        case 3 : fprintf(f_pisanje, "000\n"); break;
        case 4 : fprintf(f_pisanje, "003\n"); break;
        case 5 : fprintf(f_pisanje, "000\n"); break;
        case 6 : fprintf(f_pisanje, "004\n"); break;
        case 7 : fprintf(f_pisanje, "000\n"); break;
        case 8 : fprintf(f_pisanje, "000\n"); break;
        case 9 : fprintf(f_pisanje, "000\n"); break;
        case 10 : fprintf(f_pisanje, "000\n"); break;
        case 11 : fprintf(f_pisanje, "005\n"); break;
        case 12 : fprintf(f_pisanje, "006\n"); break;
        case 13 : fprintf(f_pisanje, "007\n"); break;
        case 14 : fprintf(f_pisanje, "008\n"); break;     }  }   }
```

7. Zaključak

Ovim je završnim radom detaljno opisan način rada programskog paketa Proteus. Taj je opis započeo simuliranjem osnovnih sklopova digitalne logike, čija je simulacija poslužila kao uvod u složenije sustave digitalne elektronike, odnosno mikrokontrolere. Za prikaz rada složenijeg mikrokontrolerskog sustava, u Proteusu je simuliran razvojni sustav za mikrokontrolerske aplikacije. U svrhu testiranja tog razvojnog sustava ispisane su mikrokontrolerske aplikacije namijenjene za interakciju sa perifernim komponentama tog sustava, a zatim je slikovno prikazana i opisana sama simulacija tih mikrokontrolerskih aplikacija. U posljednjem poglavlju ovoga rada opisana je programska veza sustava udaljenog laboratorija sa razvojnom okolinom Proteus programskog paketa, koja korisniku omogućava da nakon što spoji neku shemu unutar Proteusa, njen rad testira u sustavu udaljenog laboratorija.

Simulacijska okolina Proteus programskog paketa kao i razvojna okolina udaljenog laboratorija za razvoj mikrokontrolerskih sustava pronalaze veoma značajnu ulogu u edukacijskom procesu, te su stoga svakako vrijedni daljnjeg istraživačkog rada.

8. Literatura

[1] Proteus VSM

http://www.labcenter.com/products/vsm_overview.cfm

str. 7

[2] ISIS

http://www.labcenter.com/products/vsm_overview.cfm

str. 10 - 13

[3] Stephen Brown, Zvonko Vranesic, "Fundamentals of Digital Logic with VHDL Design, 2 Edition", McGraw-Hill, SAD, 2004.

str. 16 – 33

[4] Paolo Zenzerović, „Udaljeni laboratorij za razvoj mikroprocesorskih sustava: idejno rješenje i projektiranje razvojne okoline“, Tehnički fakultet, Rijeka, 2010.

str. 69

[S1] Primjer načina spajanja unutar Proteus Example (*hrv. primjer*) datoteke

[S2] Grafičko sučelje mikroPascal-a

http://www.mikroe.com/en/compilers/mikrobasic/pro/pic/images/screenshots/mb-pro-ide-01.png

[S3] Dijagram izvoda PIC18F448 mikrokontrolera

http://www.datasheetarchive.com/PIC18F448-datasheet.html

[S4] Raspored segmenata na jednom od četiri korištena 7-segmentna ekrana

http://www.mikroe.com/pdf/easypic5/easypic5_manual.pdf

[S5] Razvojni sustav EasyPIC5

http://www.mikroe.com/img/development-tools/pic/easypic5/gallery/easypic5_550_1.jpg

[S6] Kabel za serijsku komunikaciju
http://www.nairaland.com/nigeria/topic-182209.2720.html

[S7] RS232 modul na razvojnom sustavu EasyPIC5
http://www.mikroe.com/pdf/easypic5/easypic5_manual.pdf