

Modeliranje i simulacija mikroelektronskih komponentata i kola

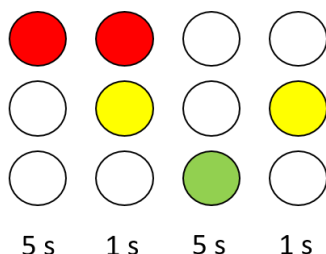
ZBIRKA REŠENIH ARDUINO PROJEKATA

Miloš Marjanović | Katedra za mikroelektroniku, Elektronski fakultet Niš | 2020.

Digitalni izlazni/ulazni pinovi

PROJEKAT 1.

Simulirati rad saobraćajnog semafora u programu Proteus korišćenjem Arduina i LED dioda. Napisati program za Arduino koji kontrolira rad semafora po algoritmu prikazanom na slici 1.

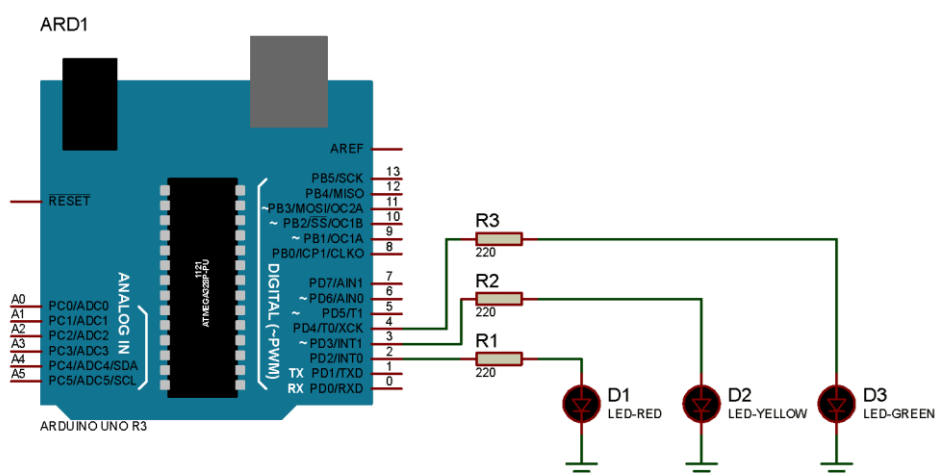


Slika 1. Ilustracija saobraćajnog semafora

Rešenje:

Električna šema saobraćajnog semafora u Proteusu data je na slici 2. Iskorišćeno je **Arduino Uno** razvojno okruženje sa **ATMega328P** mikrokontrolerom. Arduino ima **14 digitalnih pinova, označenih sa 0-13**, kao i 6 analognih pinova. Digitalni pinovi se mogu koristiti kao ulazni i izlazni. Pinovi 0 i 1 rezervisani su za serijsku UART komunikaciju; ukoliko se u programu koristi serijska komunikacija ove pinove treba osloboditi za ovu namenu. Neki od digitalnih pinova (3, 5, 6, 9, 10, 11) pored oznake broja imaju simbol ~ koji označava da se ovi pinovi mogu koristiti kao **PWM pinovi**.

LED diode povezane su preko otpornika na pinove mikrokontrolera. Iskorišćeni otpornici od $220\ \Omega$ ograničavaju struju kroz LED diode. Crvena dioda kontroliše se pinom 2, žuta pinom 3, a zelena pinom 4. U ovom projektu, svi ovi pinovi biće konfigurisani kao digitalni izlazni. Katode svih LED-ova vezane su na masu (*ground*).



Slika 2. Električna šema saobraćajnog semafora

Arduino projekat naziva se **sketch**. Otvaranjem novog sketch-a kreirane su dve funkcije: `void setup()` i `void loop()`. Funkcija **`void setup()`** izvršava se samo jednom, kada mikrokontroler

dobije napajanje, tako da se u okviru ove funkcije piše deo koda koji se odnosi na konfigurisanje pinova, inicijalizaciju komunikacije, itd. Glavni korisnički program piše se u okviru funkcije **void loop()**, koja predstavlja beskonačnu petlju. Arduino programski jezik je **case sensitive**, što znači da pravi razliku između velikih i malih slova. Svaka linija koda završava se sa tačka-zarez (;).

Pre ovih funkcija deklarišu se konstante i promenljive. U ovom projektu formirane su tri celobrojne konstante koje čuvaju vrednosti pinova na koje su povezane određene LED diode: crvenaPin, zutaPin, zelenaPin.

U okviru funkcije void setup(), sva tri pina su konfigurisana kao izlazna, što znači da će mikrokontroler na ove pinove slati napon logičke nule (0 V) ili napon logičke jedinice (5 V). Za konfiguraciju pinova koristi se funkcija **pinMode(pin,MOD)**. Argumenti funkcije su broj pina i TIP, koji može biti ulazni – INPUT, izlazni – OUTPUT ili ulazni sa internim *pull-up* otpornikom – INPUT_PULLUP. MOD pina mora biti napisan velikim slovima.



pinMode(pin,MOD)

Konfiguriše specificirani pin kao ulazni ili izlazni

pin: broj pina čiji se mod podešava

MOD: INPUT,OUTPUT ili INPUT_PULLUP

Glavni program, u skladu sa algoritmom sa slike 1, napisan je u okviru beskonačne petlje void loop(). Za postavljanje deklariranih pinova na odgovarajuće naponske nivoe koristi se funkcija **digitalWrite(pin,VREDNOST)**. Argumenti funkcije su broj pina i VREDNOST, koji može biti nivo logičke nule (0V) – LOW ili nivo logičke jedinice (5V) – HIGH. NIVO pina mora biti napisan velikim slovima.



digitalWrite(pin,VREDNOST)

Postavlja digitalni pin na određeni naponski nivo

pin: broj digitalnog pina

VREDNOST: HIGH ili LOW

Za kreiranje vremenskog kašnjenja iskorišćena je funkcija **delay(vreme)**. Argument funkcije je vreme izraženo u milisekundama (ms).

```
const int crvenaPin=2;
const int zutaPin=3;
const int zelenaPin=4;

void setup() {
  pinMode(crvenaPin,OUTPUT);
  pinMode(zutaPin,OUTPUT);
  pinMode(zelenaPin,OUTPUT);
}

void loop() {
  digitalWrite(crvenaPin,HIGH);
  delay(5000);
  digitalWrite(zutaPin,HIGH);
  delay(1000);
  digitalWrite(crvenaPin,LOW);
  digitalWrite(zutaPin,LOW);
  digitalWrite(zelenaPin,HIGH);
```

```

delay(5000);
digitalWrite(zelenaPin, LOW);
digitalWrite(zutaPin, HIGH);
delay(1000);
digitalWrite(zutaPin, LOW);
}

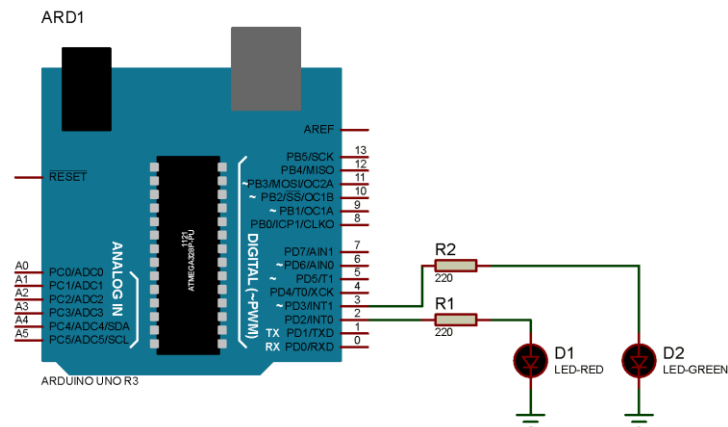
```

PROJEKAT 2.

Simulirati rad semafora za pešake u programu Proteus korišćenjem Arduina i LED dioda. Napisati program za Arduino koji naizmenično uključuje crvenu i zelenu LED diodu na svakih 5 sekundi.

Rešenje:

Električna šema semafora za pešake u Proteusu data je na slici 3.



Slika 3. Električna šema semafora za pešake

```

const int crvenaPin=2;
const int zelenaPin=3;
const int kasnjenje=5000;
int crvenaStatus=LOW;
int zelenaStatus=HIGH;

void setup() {
    pinMode(crvenaPin, OUTPUT);
    pinMode(zelenaPin, OUTPUT);
}

void loop() {
    crvenaStatus=!crvenaStatus;
    zelenaStatus=!zelenaStatus;
    digitalWrite(crvenaPin, crvenaStatus);
    digitalWrite(zelenaPin, zelenaStatus);
    delay(kasnjenje);
}

```

U ovom projektu kreirane su celobrojne promenljive koje čuvaju stanje određenih LED-ova. Na početku svakog ciklusa u beskonačnoj petlji vrši se promena trenutnog statusa LED-a primenom **operatora inverzije na promenljivu (!)**. Ako je prethodno stanje bilo LOW,



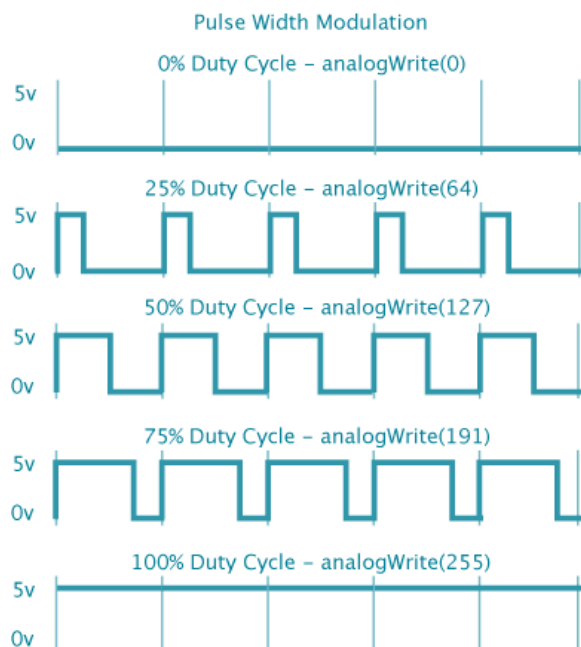
`analogWrite(pin, vrednost)`

Generiše povorku impulsa određene širine (duty cycle-a)- PWM signal na zadati pin
pin: broj pina (int vrednost)
vrednost: duty cycle signala, od 0 do 255 (int vrednost)

```
const int crvenaPin=11, zelenaPin=10, plavaPin=9;
void setup() {
}

void loop() {
  boja(255,0,0); //crvena
  delay(500);
  boja(0,255,0); //zelena
  delay(500);
  boja(0,0,255); //plava
  delay(500);
  boja(255,255,0); //zuta
  delay(500);
  boja(0,255,255); //cijan
  delay(500);
  boja(255,0,255); //magenta
  delay(500);
  boja(255,255,255); //bela
  delay(500);
}

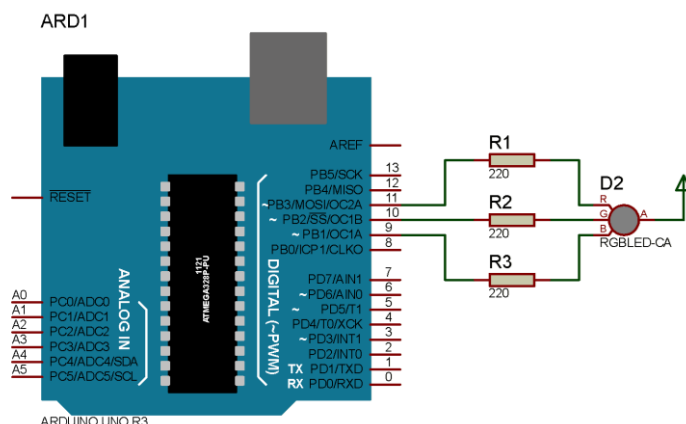
void boja(int R, int G, int B){
  analogWrite(crvenaPin,R);
  analogWrite(zelenaPin,G);
  analogWrite(plavaPin,B);
}
```



Slika 5. PWM signal definisan funkcijom `analogWrite()`

U slučaju RGB diode sa zajedničkom anodom, električna šema prikazana je na slici 6. Da bi kolo ispravno radilo, zbog suprotne logike potrebno je promeniti funkciju `analogWrite()`.

```
void boja(int R, int G, int B){
    analogWrite(crvenaPin,255-R);
    analogWrite(zelenaPin,255-G);
    analogWrite(plavaPin,255-B);
}
```



Slika 6. Električna šema Arduina i RGB diode sa zajedničkom anodom

PROJEKAT 4.

Simulirati kolo za kontrolu 10-segmentnog LED bargraph-a pomoću Arduina u programu Proteus. Napisati program za Arduino koji redom uključuje segmente u oba smera, pri čemu se uključuje segment i , posle 50 ms i segment $i+1$, da bi se posle 50 ms isključio i -ti segment, a nakon 100 ms od tog trenutka se ova sekvenca ponavlja.

Rešenje:

Električna šema kola za kontrolu 10-segmentnog LED bargraph-a pomoću Arduina prikazana je na slici 7. Svaki segment bargraph-a povezan je na odgovarajući digitalni pin mikrokontrolera preko otpornika (220 Ω) za ograničenje struje. Iskorišćen je jedan otporni niz (resistor network) sa 8 otpornika i 2 diskretna otpornika. Bargraph je u konfiguraciji sa zajedničkom katodom.

Kako su segmenti bargraph-a povezani redom od pina 2 do pina 11, efikasniji način za deklarisanje ovih pinova kao izlaznih je korišćenje **for petlje**. U suprotnom, bilo bi potrebno napisati 10 linija koda, umesto 2. For petlja definisana je za celobrojnu promenljivu i , koja ima početnu vrednost 2 i izvršava se dok je ispunjen uslov $i < 12$, pri čemu se u svakom prolasku kroz petlju ova promenljiva inkrementira za 1. U okviru for petlje, korišćenjem funkcije `pinMode()` pin i se deklarise kao izlazni.



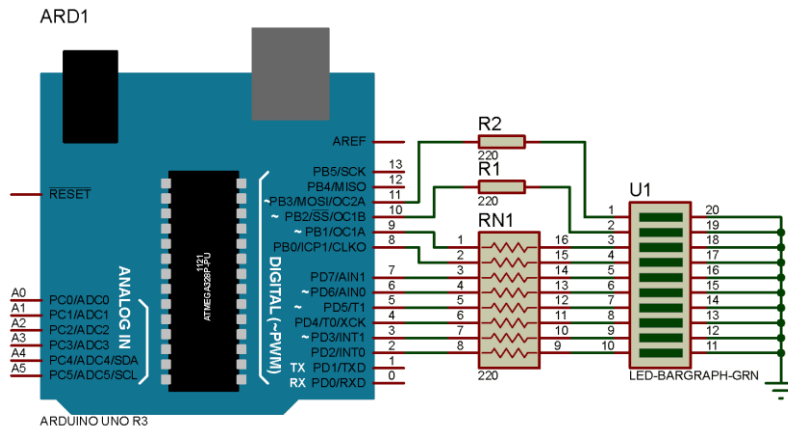
```
for(inicijalizacija;uslov;inkrement) {
    //naredbe
}
```

For petlja se koristi za ponavljanje bloka naredbi zadatih između {}

inicijalizacija: dešava se prvo i tačno jednom

uslov: prilikom svakog prolaska kroz petlju testira se uslov; ako je uslov ispunjen naredbe u petlji se izvršavaju i vrši inkrement; kada uslov nije ispunjen, petlja se završava

inkrement: izvršava se svaki put prilikom prolaska kroz petlju kada je uslov ispunjen



Slika 7. Električna šema Arduina i LED bargraph-a

```
const int kasnjenje=50;

void setup() {
  for(int i=2;i<12;i++){
    pinMode(i,OUTPUT);
  }
}

void loop() {
  for(int i=2;i<11;i++){
    digitalWrite(i,HIGH);
    delay(kasnjenje);
    digitalWrite(i+1,HIGH);
    delay(kasnjenje);
    digitalWrite(i,LOW);
    delay(kasnjenje*2);
  }
  for(int i=11;i>2;i--){
    digitalWrite(i,HIGH);
    delay(kasnjenje);
    digitalWrite(i-1,HIGH);
    delay(kasnjenje);
    digitalWrite(i,LOW);
    delay(kasnjenje*2);
  }
}
```

Glavni program realizovan je kroz dve for petlje. Prva for petlja inkrementira broj pina koji se postavlja u stanje logičke jedinice, čeka se 50 ms, zatim se u stanje logičke jedinice postavlja i+1 pin, čeka se 50 ms, zatim se isključuje i-ti pin (postavlja se u stanje logičke nule), čeka se 100 ms, a zatim se petlja ponovo izvršava sve dok ne bude ispunjen uslov i<11. To znači da će u poslednjem prolazu kroz for petlju, biti uključen pin 10, zatim posle 50ms i pin 11, pa će posle 50ms, pin 10 biti isključen i čeka se još 100 ms. Na ovaj način svi segmenti bargrapha biće uključeni.

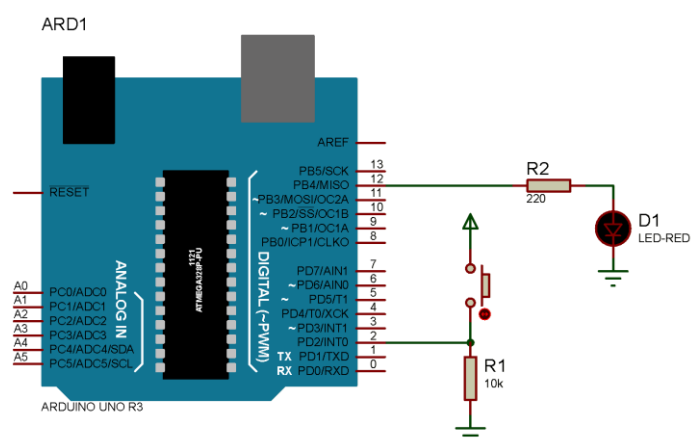
Za suprotni smer, for petlja polazi od i=11, i u svakom prolazu kroz for petlju i se dekrementuje za 1, dok je ispunjen uslov i>2. U svakom prolazu, i-ti segment se postavlja u stanje logičke jedinice, čeka se 50 ms, uključuje se i-ti segment, pa se posle 50 ms, i-ti isključuje i čeka se još 100 ms.

PROJEKAT 5.

Simulirati kolo u programu Proteus zasnovano na Arduino sa tasterom u *pull-down* konfiguraciji i LED diodom. Napisati program za Arduino koji isčitava vrednost sa tastera i u zavisnosti od stanja tastera, uključuje diodu ako je taster pritisnut, odnosno isključuje diodu ako nije pritisnut.

Rešenje:

Električna šema kola zasnovanog na Arduino sa tasterom u *pull-down* konfiguraciji i LED diodom prikazana je na slici 8. Taster je povezan na digitalni pin 2, preko koga će se isčitavati stanje tastera. Kada taster nije pritisnut, pin 2 biće preko *pull-down* otpornika na 0V (stanja logičke nule), dok kada je taster pritisnut, pin 2 biće na 5V (stanje logičke jedinice). LED je preko otpornika za ograničenje struje povezana na pin 12.



Slika 8. Električna šema Arduino kola sa tasterom u *pull-down* konfiguraciji i LED diodom

Pin 2, gde je vezan taster, deklarise se kao ulazni - INPUT korišćenjem funkcije `pinMode()`. Dok će pin 12 istom funkcijom deklarise kao izlazni. U okviru glavne petlje, stanje tastera upisuje se u celobrojnu promenljivu `taster` korišćenjem funkcije `digitalRead(pin)`. Argument funkcije `digitalRead()` je broj pina sa koga se isčitava stanje. U promenljivu `taster` biće upisana vrednost HIGH, ako je taster pritisnut, odnosno LOW, ako taster nije pritisnut.



```
digitalRead(pin)
```

Čita vrednost sa određenog digitalnog pina, koja može biti HIGH ili LOW
pin: broj pina

Korišćenjem naredbe uslovnog prelaska **if**, ispituje se stanje tastera i donosi odluka šta se izvršava ako je taster pritisnut, a šta ako nije pritisnut. Ispitivanje uslova vrši se **operatorom poređenja (==)**. Ako je taster pritisnut (pin u stanju HIGH), uključuje se LED dioda slanjem napona logičke jedinice na pin 12, korišćenjem funkcije `digitalWrite()`. Inače, ako nije pritisnut (pin u stanju LOW), LED će biti isključen, što se postiže postavljanjem pina 12 u stanje logičke nule, korišćenjem funkcije `digitalWrite()`.



```
if(uslov1){
//naredbe1
}
else if(uslov2){
//naredbe2
}
else{
//naredbe3
}
```

If naredba proverava uslov i izvršava set naredbi ako je uslov ispunjen
uslov: logički izraz

If...else omogućava veću kontrolu nad kodom od osnovne if naredbe; ako je ispunjen uslov1, izvršavaju se naredbe1, inače ako je ispunjen i uslov2, izvršavaju se naredbe2, inače se izvršavaju naredbe3; *else if* blok može se koristiti sa i bez poslednjeg else, i obrnuto

```
const int tasterPin=2;
const int ledPin=12;
int taster;

void setup() {
  pinMode(tasterPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  taster=digitalRead(tasterPin);
  if(taster==HIGH){
    digitalWrite(ledPin, HIGH);
  }
  else{
    digitalWrite(ledPin, LOW);
  }
}
```

Isti efekat može se postići bez korišćenja naredbe if, na kraći način. Jednostavno, u promenljivu taster se upiše status tastera korišćenjem digitalRead(), a zatim se funkcijom digitalWrite(ledPin,taster) podesi stanje LED-a. S obzirom da je u promenljivu taster upisano stanje HIGH ili LOW, ovo je kraći i brži način. U slučaju da je iskorišćen taster u *pull-up* konfiguraciji, a da se želi isti efekat funkcija za podešavanje LED-a bi bila sa korišćenjem operatora inverzije: digitalWrite(ledPin,!taster).

```
taster=digitalRead(tasterPin);
digitalWrite(ledPin,taster);
```

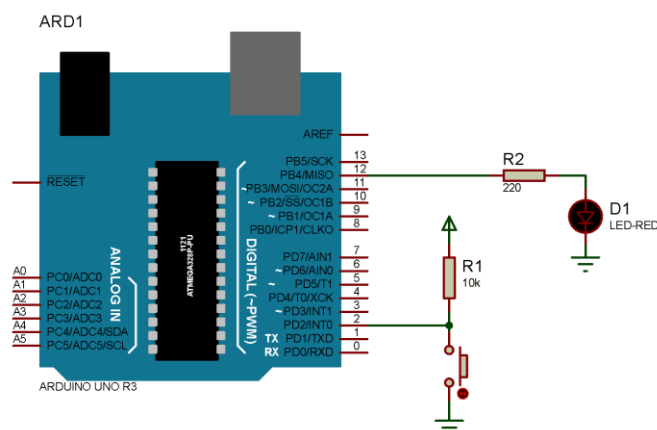
PROJEKAT 6.

Simulirati kolo u programu Proteus zasnovano na Arduino sa tasterom u *pull-up* konfiguraciji i LED diodom. Napisati program za Arduino koji na svaki pritisak tastera menja stanje LED-a (uključeno/isključeno).

Rešenje:

Električna šema Arduino kola sa tasterom u *pull-up* konfiguraciji i LED diodom data je na slici 9. U ovom slučaju, kada taster nije pritisnut, pin 2 će preko *pull-up* otpornika biti u stanju logičke jedinice, dok kada se taster pritisne, pin 2 odlazi u stanje logičke nule.

Pre svega deklariraju se tri promenljive: *preTaster*, koja će čuvati prethodno stanje tastera, *sadTaster*, koja će čuvati trenutno stanje tastera i *ledStatus*, koja će čuvati trenutni status LED diode (ova promenljiva se inicijalno postavlja na LOW). U funkciji *void setup()* će pored deklaracije pinova biti postavljen LED pin na trenutno stanje, LOW i korišćenjem funkcije *digitalRead()*, biće isčitano stanje tastera, koje se upisuje prvo u promenljivu *preTaster*, a zatim u promenljivu *sadTaster*.



Slika 9. Električna šema Arduino kola sa tasterom u *pull-up* konfiguraciji i LED diodom

U glavnom programu, prvo se isčitava trenutno stanje tastera korišćenjem funkcije *digitalRead()* i upisuje se u promenljivu *sadTaster*. Zatim se korišćenjem *if* naredbe porede trenutno i prethodno stanje korišćenjem **operatora nejednakosti (!=)**. Ako su isčitana različita stanja, to znači da se stanje tastera promenilo, pa se dalje ispituje da li je trenutno stanje tastera LOW, što znači da je taster pritisnut. Ako je ispunjen ovaj uslov, menja se stanje LED-a korišćenjem operatora inverzije (!) i najnovije stanje LED-a se podešava korišćenjem funkcije *digitalWrite()*. Na kraju, neophodno je da promenljiva *preTaster* uzme vrednost promenljive *sadTaster*, da bi prilikom sledećeg prolaska kroz petlju bilo moguće utvrditi da li je stanje tastera promenjeno.

```
const int tasterPin=2;
const int ledPin=12;
int preTaster;
int sadTaster;
int ledStatus=LOW;

void setup() {
  pinMode(tasterPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledStatus);
  preTaster=digitalRead(tasterPin);
  sadTaster=digitalRead(tasterPin);
}
```

```

void loop() {
  sadTaster=digitalRead(tasterPin);
  if(sadTaster!=preTaster){
    if(sadTaster==LOW){
      ledStatus=!ledStatus;
      digitalWrite(ledPin,ledStatus);
    }
    preTaster=sadTaster;
  }
}

```

Usled efekta poskakivanja signala sa tastera (*bouncing*), ovaj kod moguće je poboljšati primenom nekog od algoritama za **debouncing**. Najjednostavnija mogućnost je da se nakon što se utvrdi da je došlo do promene stanja tastera, sačeka neko vreme, na primer 200 ms, a zatim ponovo isčita taster, ako se utvrdi da je stanje tastera i dalje LOW, to znači da je stanje stabilno – taster je sigurno pritisnut.

```

...
if(sadTaster!=preTaster){
  delay(200);
  sadTaster=digitalRead(tasterPin);
  if(sadTaster==LOW){
...

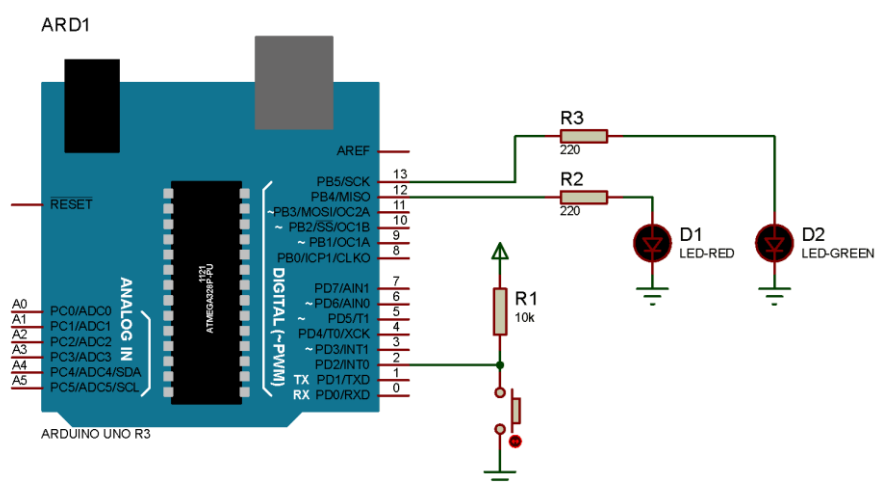
```

PROJEKAT 7.

Simulirati kolo u programu Proteus zasnovano na Arduino sa tasterom u *pull-up* konfiguraciji i dve LED diode. Napisati program za Arduino koristeći mehanizam prekida (interrupt-a) koji će na svaki pritisak tastera zameniti stanja LED-ovima. Na početku je crvena LED uključena, a zelena LED isključena.

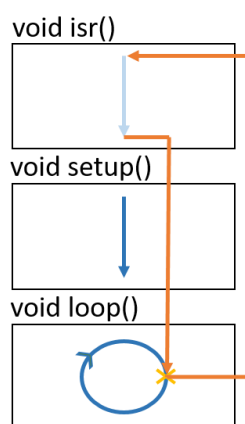
Rešenje:

Električna šema kola Arduino sa tasterom u *pull-up* konfiguraciji sa dve LED diode prikazana je na slici 10.



Slika 10. Električna šema kola Arduino sa tasterom i dve LED diode

Da bi se izbeglo neprekidno isčitavanje stanja tastera, tj. *polling*, što i nije pouzdana metoda biće korišćen **mehanizam prekida (interrupt)**. Interrupt je signal koji primi mikrokontroler i tada prekida izvršenje koda koji je do tada izvršavao i izvršava posebnu funkciju koja vrši servisiranje prekida (**ISR – interrupt service routine**). Nakon što mikrokontroler završi sa izvršenjem ISR-a, nastavlja izvršenje programa od tačke u kojoj je stao kada je bio prekinut (slika 11). ISR se poziva automatski po prijemu signala prekida i nikada ne vraća rezultat, tako da se deklarise kao void. Signal prekida može se generisati interno (tajmer stigao do određenog vremena) ili eksterno. Eksterne prekide mogu izazvati komponente (na primer tasteri) povezani na određene digitalne ulazne pinove koji mogu raditi sa interruptima. **Arduino pinovi 2 i 3 omogućavaju rad sa interruptom**. Stanja pina koja mogu dovesti do prekida su: pin je na logičkoj jedinici, pin je na logičkoj nuli, stanje pina se menja sa logičke nule na jedinicu i obrnuto.



Slika 11. Ilustracija principa interrupt-a

Na početku su deklarisan konstante i promenljive. Kada se deklarise globalna promenljiva čija će se vrednost menjati u više funkcija potrebno je da bude deklarisan kao **volatile**. U funkciji void setup() deklarisan su pinovi i postavljena početna stanja LED-ova, ali je ovde pinu gde je povezan taster dodeljena prekidna rutina pozivanjem funkcije **attachInterrupt(digitalPinToInterrupt(pin),ISR,mod)**. Argumenti funkcije su: broj pina kome se dodeljuje prekid, ISR – naziv funkcije u koju će program ući nakon pojave prekida, kao i mod- koja pojava na pinu uzrokuje prekid. Modovi mogu biti: LOW – pin ima vrednost logičke nule, RISING – vrednost pina se promenila sa logičke nule na logičku jedinicu, FALLING – vrednost pina se promenila sa logičke jedinice na logičku nulu, CHANGE – vrednost pina se promenila. U ovom slučaju, pin kome se dodeljuje prekid je pin 2, funkcija u koju će program ući nakon prekida je tasterISR i to će se desiti svaki put kada se stanje na pinu promeni iz stanja logičke jedinice u stanje logičke nule (taster pritisnut). Logička promenljiva (tipa boolean) tasterFleg predstavlja vezu između glavnog programa i ISR-a. Inicijalno se postavlja u stanje *false* – taster nije pritisnut.



```
attachInterrupt(digitalPinToInterrupt(pin), ISR, MOD)
```

Deklarisanje mehanizma prekida (interrupta)

pin: broj pina (2 ili 3)

ISR: naziv funkcije koja se poziva kada dođe do prekida

MOD: definiše šta će okinuti interrupt (LOW, RISING, FALLING, CHANGE)

Prilikom pisanja ISR funkcija treba voditi računa da one budu što kraće i efikasnije. U ovom slučaju kada se desi prekid usled pritiska tastera, poziva se ISR funkcija tasterISR i u njoj se promenljiva tasterFleg setuje na vrednost *true*, što će glavnom programu signalizirati da je taster pritisnut.

```
const int tasterPin=2;
const int crvenaPin=12;
const int zelenaPin=13;
int taster;
int crvenaStatus=HIGH;
int zelenaStatus=LOW;
volatile boolean tasterFleg;

void setup() {
    pinMode(tasterPin, INPUT);
    pinMode(crvenaPin, OUTPUT);
    pinMode(zelenaPin, OUTPUT);
    digitalWrite(crvenaPin, crvenaStatus);
    digitalWrite(zelenaPin, zelenaStatus);
    attachInterrupt(digitalPinToInterrupt(tasterPin), tasterISR, FALLING;
    tasterFleg=false;
}

void tasterISR() {
    tasterFleg=true;
}

void loop() {
    if(tasterFleg==true) {
        crvenaStatus=!crvenaStatus;
        zelenaStatus=!zelenaStatus;
        digitalWrite(crvenaPin, crvenaStatus);
        digitalWrite(zelenaPin, zelenaStatus);
        tasterFleg=false;
    }
}
```

U glavnom programu, ako je taster pritisnut, što signalizira vrednost promenljive tasterFleg, zameniće se trenutni statusi LED-ova korišćenjem operatora inverzije (!) i korišćenjem funkcije digitalWrite(), biće postavljene odgovarajuće vrednosti logičkih stanja na pinove. Na kraju, nakon što se izvrši deo koda uslovljen prekidom, promenljiva tasterFlag postavlja se na vrednost *false*.

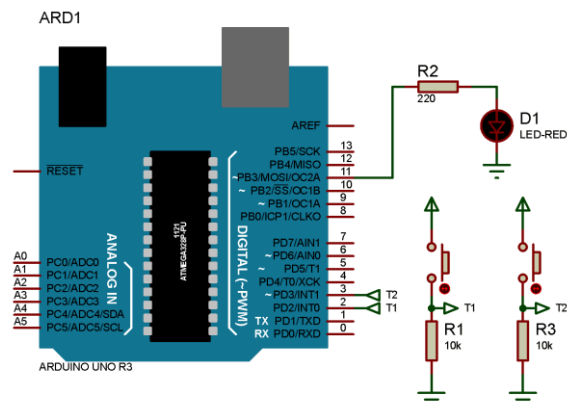
PROJEKAT 8.

Simulirati kolo u programu Proteus zasnovano na Arduinu sa dva tastera u *pull-down* konfiguraciji i jednim LED-om. Napisati program za Arduino koristeći mehanizam prekida (interrupt-a) kojim će se menjati intenzitet osvetljenosti LED-a. Na svaki pritisak tastera Up intenzitet LED-a se povećava za 10%, a na svaki pritisak tastera Down intenzitet LED-a se smanjuje za 10%.

Rešenje:

Električna šema Arduina sa dva tastera u *pull-down* konfiguraciji i LED-om prikazana je na slici 12. Taster Down povezan je na pin 2, dok je taster Up povezan na pin 3. S obzirom da je

potrebno menjati intenzitet LED-a, što se postiže PWM signalom, izabran je PWM digitalni pin 11.



Slika 12. Električna šema Arduina sa dva tastera u *pull-down* konfiguraciji i LED-om

Inicijalna vrednost celobrojne promenljive intenzitet je 50, pri čemu je ova promenljiva deklarirana kao volatile, jer se njena vrednost menja u više funkcija. Pored deklaracije pinova u funkciji void setup() podešena su i dva prekida koja će izazvati promena napona sa signala logičke nule, na signal logičke jedinice na dva tastera. Usled prekida biće pozvana downISR(), odnosno upISR().

```
const int downTaster=2;
const int upTaster=3;
const int ledPin=11;
volatile int intenzitet=50;

void setup() {
    pinMode(downTaster, INPUT);
    pinMode(upTaster, INPUT);
    pinMode(ledPin, OUTPUT);
    attachInterrupt(digitalPinToInterrupt(downTaster), downISR, RISING);
    attachInterrupt(digitalPinToInterrupt(upTaster), upISR, RISING);
}

void downISR() {
    if(intenzitet>0){
        intenzitet-=10;
    }
}

void upISR() {
    if(intenzitet<100){
        intenzitet+=10;
    }
}

void loop() {
    analogWrite(ledPin, 255*intenzitet/100);
}
```

Ako je pritisnut Down taster, desiće se prekid i biće pozvana funkcija koja će ako je vrednost promenljive intenzitet veća od 0, smanjiti vrednost ove promenljive za 10. Ako je pritisnut

taster Up, desiće se prekid i biće pozvana funkcija koja će ako je vrednost promenljive intenzitet manja od 100, povećati vrednost ove promenljive za 10.

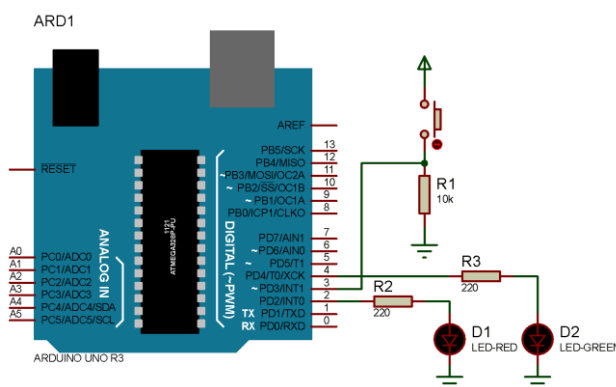
U glavnom programu u beskonačnoj petlji, na LED pin se šalje vrednost $255 \cdot \text{intenzitet} / 100$, što određuje duty cycle PWM signala koji kontroliše intenzitet LED-a. Minimalna vrednost koja se šalje je 0, a maksimalna 255, što odgovara duty cycle-u od 0 do 100% (slika 5).

PROJEKAT 9.

Simulirati kolo u programu Proteus zasnovano na Arduinou sa dve LED diode i jedim tasterom u *pull-down* konfiguraciji. Napisati program za Arduino korišćenjem tajmera koji će treptati izabranu LED diodu sa periodom 0.5 s. Izbor diode koja treperi vrši se tasterom.

Rešenje:

Električna šema Arduina sa dve LED diode i tasterom u *pull-down* konfiguraciji prikazana je na slici 13. Ukoliko se stanje tastera isčitava tehnikom prekida, potrebno ga je povezati na pin 2 ili pin 3, kao što je ovde urađeno.



Slika 13. Električna šema Arduina sa dve LED diode i tasterom u *pull-down* konfiguraciji

U ovom projektu biće korišćen tajmer (**Timer**), periferija mikrokontrolera čija je uloga merenje vremena; na svaki taktni signal brojač tajmera se inkrementira. Posle svakog inkrementiranja vrši se poređenje stanja brojača sa vrednošću upisanom u registar. Kada brojač dostigne vrednost upisanu u registar, brojač se resetuje i procesoru se šalje prekidni signal. Tajmeri se koriste kada je potrebno izvršavanje periodičnih zadataka, na primer čitanje senzora, osvežavanje displeja, čitanje tastature... Arduino ima tri tajmera: Timer 0 – koji je osmobitni i njega koriste funkcije `delay()`, `milis()`, `micros()`, Timer 1 – koji je 16-to bitni i pristupa mu se korišćenjem funkcije iz biblioteke `TimerOne`, Timer 2 – koristi se u drugim Arduino bibliotekama.

Uključivanje biblioteke postiže se preko header fajlova, korišćenjem `#include<naziv_biblioteke.h>`, na početku koda. Sledi deklarisanje konstanti i promenljivih. Promenljive čija se vrednost menja kroz više funkcija deklariraju se kao *volatile*. Promenljiva aktivna je logičkog tipa (*boolean*), što znači da ona može uzimati vrednosti *true* (tačno), *false* (netačno), koristiće se da čuva vrednost koja će LED dioda biti aktivna, tj. koja će treptati. Ako je vrednost *false*, treptaće zeleni LED, a ako je vrednost *true*, treptaće crveni LED. Konstanta period čuva vrednost vremena reseta brojača tajmera u mikrosekundama, zbog

vrednosti deklarirane se kao unsigned long. Naime, promenljiva tipa int može uzimati vrednosti od -32768 do 32767, pa se za promenljive čija vrednost prelazi ovaj broj koristi unsigned long. Za ovaj tip podatka karakteristično je da je uvek pozitivan jer je neoznačen (unsigned) i može imati vrednost od 0 do $2^{32}-1$.

```
#include <TimerOne.h>
const int tasterPin=3;
const int crvenaPin=2;
const int zelenaPin=4;
volatile int taster;
volatile int crvenaStatus=LOW;
volatile int zelenaStatus=LOW;
volatile boolean aktivna;
const unsigned long period=500000;

void setup() {
  pinMode(crvenaPin,OUTPUT);
  pinMode(zelenaPin,OUTPUT);
  pinMode(tasterPin,INPUT);
  digitalWrite(crvenaPin,crvenaStatus);
  digitalWrite(zelenaPin,zelenaStatus);
  aktivna=false;
  attachInterrupt(digitalPinToInterrupt(tasterPin),tasterISR,RISING);
  Timer1.initialize(period);
  Timer1.attachInterrupt(tajmerISR);
}

void tasterISR(){
  aktivna= !aktivna;
}

void tajmerISR(){
  if(aktivna==false){
    zelenaStatus=!zelenaStatus;
    digitalWrite(zelenaPin,zelenaStatus);
  }
  else{
    crvenaStatus=!crvenaStatus;
    digitalWrite(crvenaPin,crvenaStatus);
  }
}

void loop() {
}
```

U okviru funkcije void setup() deklarirani su pinovi na koje su vezani LED-ovi kao izlazni, pin gde je povezan taster deklarisan je kao ulazni. Inicijalno obe LED diode su isključene, ali je vrednost logičke promenljive aktivna postavljena na *false*, što znači da će prvo početi da treperi zeleni LED. Prekidna rutina tasterISR izvršavaće se svaki put kada signal na pinu tasterPin pređe sa vrednosti logičke nule, na vrednost logičke jedinice, dakle svaki put kada se taster pritisne. Ovde je inicijalizovan tajmer korišćenjem funkcije **Timer1.initialize(vrednost)**. Argument funkcije je vrednost do koje će brojati tajmer pre nego što izazove prekid (period). Ova vrednost definisana je napred kao konstanta; u ovom slučaju, 500000 mikrosekundi = 0.5 s. Na kraju je tajmeru dodeljena prekidna rutina, korišćenjem funkcije **Timer1.attachInterrupt(naziv_ISR)**, argument funkcije je naziv prekidne

rutine koja će se izvršavati svaki put kada tajmer odbroji zadato vreme. U ovom slučaju na svakih 0.5 s, izvršavaće se prekidna rutina `tajmerISR()`.



```
#include <TimerOne.h>
Uključivanje biblioteke za korišćenje Timer1
Timer1.initialize(vrednost)
Inicijalizacija tajmera Timer1
vrednost: period tajmera u mikrosekundama
Timer1.attachInterrupt(funkcija)
Dodeljivanje prekidne funkcije tajmeru Timer1
funkcija: naziv ISR-a
Timer1.setPeriod(vrednost)
Promena perioda tajmera Timer1
vrednost: period tajmera u mikrosekundama
```

Na svaki pritisak tastera, izvršavaće se prekidna rutina `tasterISR()`, koja će samo korišćenjem operatora inverzije (!) promeniti vrednost promenljive `aktivna`, *false* u *true* i obrnuto. Na svakih 0.5 s izvršavaće se prekidna rutina `tajmerISR()`, koja će proveravati vrednost promenljive `aktivna`. Ako je vrednost `aktivna` jednako *false*, korišćenjem operatora inverzije promeniće se status zelenog LED-a, LOW u HIGH i obrnuto i postaviće se taj status na odgovarajući pin. Ako je vrednost `aktivna` jednako *true*, korišćenjem operatora inverzije promeniće se status crvenog LED-a i postaviće se taj status na odgovarajući pin.

Na ovaj način postiže se treptanje LED-ova sa zadatim periodom bez dodatnog zauzimanja procesora za vreme dok ne istekne 0.5 s. Napomenimo, korišćenje prekida omogućava paralelizaciju procesa, odnosno omogućava da se dva ili više procesa "odigrava" istovremeno. Na taj način se rasterećuje centralna procesorska jedinica, a koriste se periferalne. Funkcija `void loop()` je prazna, tj. procesor je slobodan da izvršava druge operacije. U trenutku kada se taster pritisne, dioda koja je bila aktivna, ostaje u poslednjem stanju, a počinje treptanje druge- do tada neaktivne diode. Na primer, ako je treptala crvena i u trenutku pritiska tastera je bila u stanju HIGH, ostaje uključena i počinje da treperi zeleni LED.

Napomenimo, vrednost perioda tajmera može se menjati u toku izvršenja programa korišćenjem funkcije **`Timer1.setPeriod(vrednost)`**, gde je vrednost promenljiva koja sadrži željeni period brojača tajmera izražen u mikrosekundama. Ako je vrednost deklarirana na 100000, u sledećem primeru, vrednost će se uvećati za 10000, pa se preko ove funkcije prosleđuje nova vrednost perioda: 110000.

```
vrednost+=10000;
Timer1.setPeriod(vrednost);
```

Serijska UART komunikacija

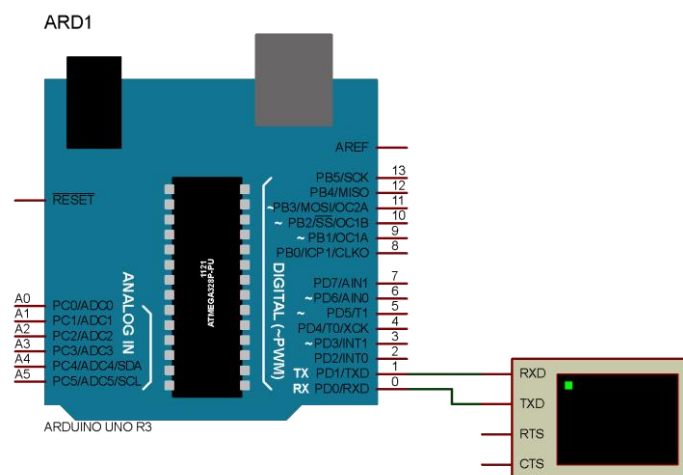
PROJEKAT 10.

Simulirati kolo Arduinoa povezano na virtuelni terminal u programu Proteus. Napisati program za Arduino koji na virtuelni terminal šalje:

- Zadati string,
- Broj karaktera u zadatom stringu,
- String dobijen spajanjem dva stringa,
- String dobijen spajanjem stringa i broja,
- Poruku o tome da li su dva stringa jednaka, poređenje vršiti funkcijama strcmp i strcmp,
- U beskonačnoj petlji generisati vrednosti dve promenljive korišćenjem funkcije random, odštampati generisane brojeve, potom im zameniti mesta i ponovo prikazati rezultat. Zamena vrednosti u promenljivama vrši se u funkciji korišćenjem referenci.

Rešenje:

Simulaciona šema Arduinoa povezanog na virtuelni terminal u program Proteus prikazana je na slici 14. *Virtuelni terminal nalazi se u sekciji Instruments. **Za serijsku UART komunikaciju kod Arduinoa rezervisani su digitalni pinovi 0 i 1, i to 0 (RX pin) prijem podataka, odnosno 1 (TX pin) za slanje podataka.*** TX pin Arduinoa vezuje se na RX pin virtuelnog terminala, a RX pin Arduinoa na TX pin virtuelnog terminala.



Slika 14. Simulaciona šema Arduinoa povezanog na virtuelni terminal

Da bi se otvorio serijski port i podesio *baud rate* (brzina prenosa podataka u bitovima u sekundi) koristi se funkcija **Serial.begin(brzina)**. Argument ove funkcije je brzina prenosa podataka. S obzirom da se podrazumeva da Arduino šalje podatke preko USB-a, baud rate treba podesiti tipično na 9600, što nije jedina moguća brzina prenosa.

Funkcija za štampanje na serijskom monitoru (virtuelnom terminalu) je **Serial.print()**, odnosno **Serial.println()**. **Serial.print()** štampa zadatu vrednost i ostavlja kursor na kraju

odštampanog rezultata, dok funkcija `Serial.println()` štampa rezultat i postavlja kursor u novi red. Argumenti ovih funkcija mogu biti promenljive i poruke zadate u vidu stringova. Na primer: `Serial.print(broj)` će odštampati vrednost promenljive `broj`, dok će `Serial.print("broj")` odštampati: `broj`.



`Serial.begin(brzina)`

Inicijalizuje serijsku UART komunikaciju uz podešavanje brzine prenosa podataka
brzina: brzina prenosa podataka u bitovima u sekundi (baud rate)

`Serial.print(vrednost)`

Štampanje zadate vrednosti na serijski port (terminal)

vrednost: vrednost koja se štampa, može biti bilo kog tipa (int, float, String, ...)

`Serial.println(vrednost)`

Štampanje zadate vrednosti na serijski port (terminal) i prelazak u novi red

vrednost: vrednost koja se štampa, može biti bilo kog tipa (int, float, String, ...)

```
String tekst1="Modeliranje i simulacija";
String tekst2=" mikroelektronskih komponenata i kola.";

void zameniMesta(int &vr1, int &vr2);

void setup() {
    Serial.begin(9600);

    // rad sa stringovima
    Serial.print("Naziv predmeta je: ");
    Serial.print(tekst1);
    Serial.println(".");
    Serial.print("U nazivu predmeta je: ");
    Serial.print(tekst1.length());
    Serial.println(" karaktera.");
    tekst1.concat(tekst2);
    Serial.println("Pun naziv predmeta je:");
    Serial.println(tekst1);

    //konvertovanje broja u string
    int a=327;
    String broj="Broja a je: ";
    broj+=a;
    Serial.println(broj);

    //poredjenje stringova
    char string1[]="model";
    char string2[]="modeliranje";
    if(strcmp(string1,string2)==0){
        Serial.println("Stringovi su isti");
    }
    else{
        Serial.println("Stringovi nisu isti!");
    }
    if(strncmp(string1,string2,5)==0){
        Serial.println("Stringovi su isti");
    }
    else{
        Serial.println("Stringovi nisu isti!");
    }
}

void loop() {
```

```

// vraćanje više od 1 vrednosti iz funkcije korišćenjem referenci
int x=random(10);
int y=random(10);
Serial.print("Izvućeni su brojevi: ");
Serial.print(x);
Serial.print(", ");
Serial.println(y);
zameniMesta(x,y);
Serial.print("Kada zamene mesta, dobijamo: ");
Serial.print(x);
Serial.print(", ");
Serial.println(y);
delay(2000);
}

void zameniMesta(int &vr1, int &vr2){
    int temp;
    temp=vr1;
    vr1=vr2;
    vr2=temp;
}

```

U ovom projektu prvo će biti deklarirana dva stringa pod nazivima tekst1 i tekst2. Nakon inicijalizacije serijske komunikacije u okviru funkcije void setup(), biće korišćenjem Serial.print("Naziv predmeta je: ") odštampan tekst zadat između navodnika, pa će u nastavku istog reda biti odštampana vrednost stringa tekst1, i simbol ., nakon čega kursor prelazi u novi red.

U novom redu biće prikazan poruka U nazivu predmeta je:, pa će korišćenjem funkcije **tekst1.length()**, biti određen **broj karaktera u stringu** tekst1.

Konkatenacija, ili **spajanje stringova** vrši se funkcijom **tekst1.concat(tekst2)**. Ova funkcija stringu tekst1 dodaje sadržaj stringa tekst2 i to čuva u stringu tekst1. Korišćenjem Serial.println() biće prikazana vrednost spojenih stringova.

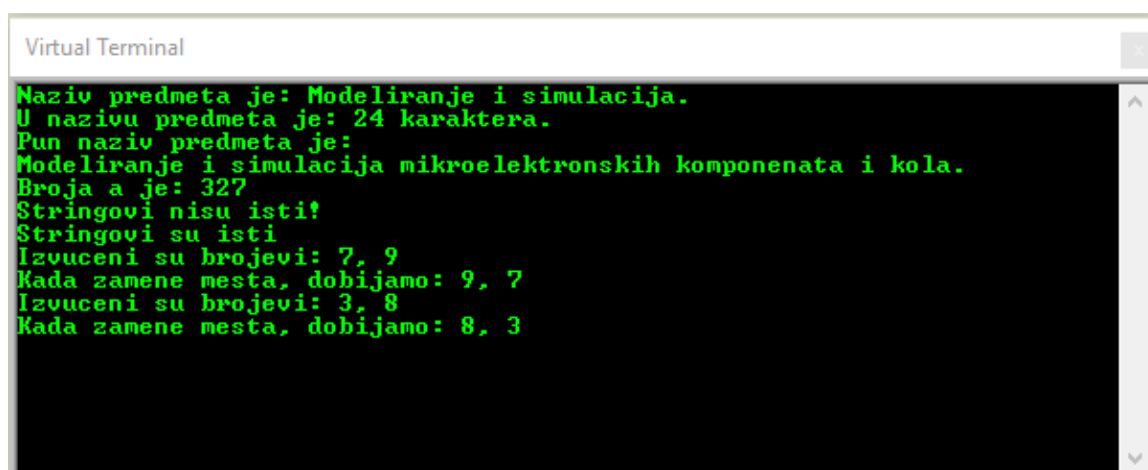
Konvertovanje broja u string demonstrirano je u nastavku koda: deklarirana je celobrojna promenljiva a i dodeljena joj neka vrednost. Nakon toga deklariran je string broj i izvršeno spajanje stringa i broja korišćenjem broj+=a. Ista funkcionalnost dobila bi se i sa: **broj.concat(a)**. Korišćenjem Serial.println() prikazan je sadržaj stringa nastao spajanjem stringa i broja.

Poređenje stringova izvršeno je pomoću dve funkcije. Pre svega su deklarirani stringovi kao nizovi karaktera. Formirana su dva niza string1[] i string2[], tipa char (karakter). Za poređenje stringova može se koristiti funkcija **strcmp(string1,string2)==0**, koja vraća *true* vrednost ako su sadržaji stringova string1 i string2 jednaki. S obzirom da se u zadatom primeru stringovi razlikuju, funkcija će vratiti vrednost *false*, tj. biće štampana poruka da stringovi nisu isti, korišćenjem Serial.println(). S druge strane kada se koristi funkcija za poređenje **strncmp(string1,string2,n)**, biće poređen sadržaj prvih n-zadatih karaktera u stringovima i ako se podudaraju, poređenje se zaustavlja i funkcija vraća vrednost *true*. U ovom primeru, proverava se prvih 5 karaktera u zadatim stringovima. S obzirom da je prvih 5 karaktera isto, biće štampana poruka da su stringovi isti, iako nisu.

Primer ispravnog pisanja funkcija koje treba da vrate više od jedne vrednosti dat je u beskonačnoj petlji. U tom slučaju potrebno je pisati funkcije korišćenjem reference. **Funkcije sa referencama moraju biti deklarisanе na početku programa**, pre nego što se pozovu. Pre svega, deklarisanе su dve celobrojne promenljive x i y koje uzimaju vrednost koju vraća funkcija **random(10)**. Funkcija random vraća slučajno generisanu cifru od 0 – 9, zato što je prosleđen argument 10. Izvučeni brojevi se štampaju na virtuelnom terminalu, a zatim se poziva funkcija za zamenu mesta brojevima, funkcija ima dva argumenta x,y. Nakon izvršenja funkcije, štampa se dobijeni rezultat i čeka 2 s, a nakon toga se petlja ponavlja.

Funkcija `zameniMesta(&vr1,&vr2)`, prihvata promenljive i smešta ih u **promenljive sa referencom (&)**. To znači da će promene vrednosti promenljivih u okviru funkcije takođe biti izvršene u promenljivama koje su prosleđene kada je funkcija pozvana. Dakle, promene u `&vr1` su vidljive u x, odnosno promene u `&vr2` su vidljive u y. Unutar funkcije pravi se pomoćna celobrojna promenljiva temp i vrši se zamena vrednosti u promenljivama. Temp uzme vrednost vr1, u vr1 se upiše vrednost vr2, a onda u vr2 vrednost skladištena u temp. Na ovaj način zamenjene su vrednosti promenljivama `&vr1` i `&vr2`, odnosno x i y.

Prikaz rezultata izvršenja programa u virtuelnom terminalu dat je na slici 15.



```
Virtual Terminal
Naziv predmeta je: Modeliranje i simulacija.
U nazivu predmeta je: 24 karaktera.
Pun naziv predmeta je:
Modeliranje i simulacija mikroelektronskih komponenata i kola.
Broja a je: 327
Stringovi nisu isti!
Stringovi su isti
Izvučeni su brojevi: 7, 9
Kada zamene mesta, dobijamo: 9, 7
Izvučeni su brojevi: 3, 8
Kada zamene mesta, dobijamo: 8, 3
```

Slika 15. Prikaz rezultata izvršenja programa u virtuelnom terminalu

PROJEKAT 11.

Simulirati kolo Arduina povezano na virtuelni terminal u programu Proteus. Napisati program za Arduino koji na virtuelni terminal šalje:

- Ostatak pri deljenju dva cela broja,
- Poruku o parnosti broja,
- Vrednost zadatih brojeva posle primene funkcije za ograničenje vrednosti,
- Kvadrat broja,
- Koren broja,
- Zaokružene vrednosti brojeva,
- Vrednosti izvršenja trigonometrijskih funkcija,
- Rezultat pomeranja bitova,
- Zadati broj u različitim brojnim sistemima.

Rešenje:

Simulaciona šema Arduina povezanog na virtuelni terminal u program Proteus prikazana je na slici 14.

Na početku su deklarirane dve celobrojne promenljive x i y . U okviru funkcije `void setup()` napisane su sve funkcionalnosti programa, pre svega je inicijalizovana serijska komunikacija. Prvo se na virtuelni terminal šalje **ostatak pri deljenju brojeva**, koristi se operator $x\%y$. Zatim se ispituje parnost broja x , ispituje se da li je ostatak pri deljenju $x\%2$ jednak nuli, ako jeste to znači da je broj paran, u suprotnom broj je neparan.

Za **ograničenje vrednosti promenljive na određeni opseg** koristi se funkcija **`constrain(x,min,max)`**. Argumenti funkcije su: x – promenljiva čija vrednost treba da bude ograničena, min – donja vrednost za ograničenje, max – gornja vrednost za ograničenje. Ako je vrednost x manje ili jednako od min , funkcija vraća min ; ako je x između min i max vrednosti, funkcija vraća x ; ako je x veće ili jednako od max , funkcija vraća max . U ovom primeru deklarirane su dve celobrojne promenljive u koje se upisuje vrednost koju vraća funkcija `constrain()`. U prvom slučaju, `ogranicena1=constrain(x,10,20)`, s obzirom da je $x=22$, a to je veće od 20, u promenljivu `ogranicena1` biće upisana gornja vrednost, tj. 20. U drugom slučaju, `ogranicena2=constrain(y,10,20)`, s obzirom da je $y=10$, što je jednako sa min , u promenljivu `ogranicena2` biće upisano 10.

Za matematičku operaciju **kvadriranja** koristi se funkcija **`pow(a,b)`**, koja će vratiti vrednost a^b . U ovom primeru formirana je promenljiva z u koju se upisuje rezultat izvršenja funkcije `pow(y,2)`, a pošto je $y=10$, dobiće se $z=100$. Korišćenjem ove funkcije moguće je naći i vrednost korena, koji nije kvadratni. Na primer, $5\sqrt{2}$ (peti koren iz 2) možemo napisati kao $2^{1/5}$, odnosno `pow(2,1.0/5)`. Promenljiva koja prihvata rezultat ove funkcije mora biti float, kako bi bio sačuvan tačan rezultat. Za traženje **kvadratnog korena** koristi se funkcija **`sqrt()`**.

Za **zaokruživanje float brojeva** koriste se funkcije **`floor(x)`**, koja broj x zaokružuje na manji ceo broj, odnosno funkcija **`ceil(x)`**, koja broj x zaokružuje na veći ceo broj. Broj x može biti i negativan, tako će na primer funkcijom `ceil(-1.1)` broj biti zaokružen na -1.0.

Trigonometrijske funkcije kao što su **`sin(ugao)`**, **`cos(ugao)`** vratiće sinus, odnosno kosinus ugla, pri čemu $ugao$ mora biti izražen u radijanima. Ako korisnik unese $ugao$ izražen u stepenima, potrebno ga je konvertovati u radijane, zato je u ovom primeru formirana float promenljiva `radijan=stepen*PI/180`. **PI** je predefinisana konstanta u Arduinu ($PI=3.14$).

Za **pomeranje na nivou bita** koriste se **operatori $>>$ i $<<$** , koji pomeraju broj za zadati broj bitova u desno, odnosno levo. U ovom primeru deklarirana je promenljiva $a=6$, broj a je u binarnom sistemu `%0110`. Korišćenjem operatora `a<<1`, binarna vrednost a se pomera za jedno mesto u 1, pa se dobija `%1100`, što odgovara broju 12 u dekadnom brojnom sistemu, zato će u promenljivu `rez` biti upisano 12. Slično, korišćenjem operatora `a>>2`, binarna vrednost broja a se pomera sa dva mesta u desno, pa se dobija `%0001`, što odgovara broju 1 u dekadnom sistemu, zato će u promenljivu `rez` biti upisano 1.

```
int x=22;
int y=10;
void setup() {
```

```

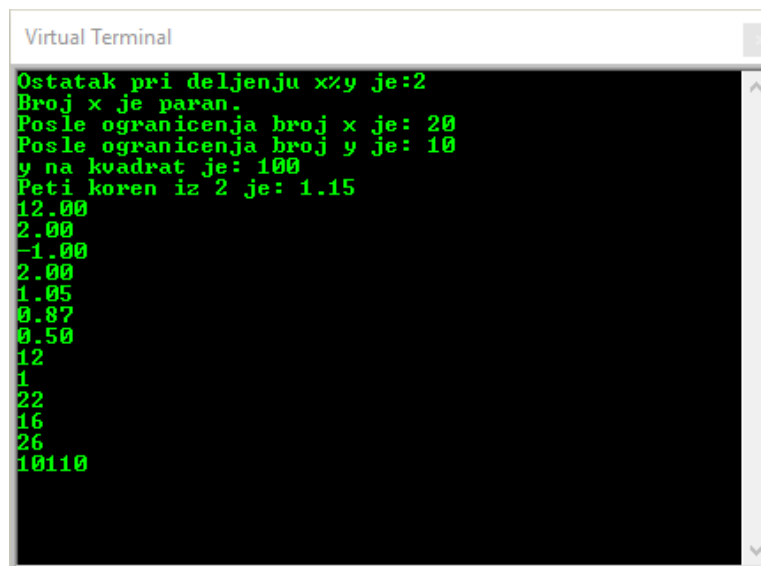
Serial.begin(9600);
//ostatak pri deljenju
Serial.print("Ostatak pri deljenju x%y je:");
Serial.println(x%y);

if(x%2==0){
    Serial.println("Broj x je paran.");
}
else{
    Serial.println("Broj x je neparan.");
}
//ogranicenje vrednosti
int ogranicena1=constrain(x,10,20);
Serial.print("Posle ogranicenja broj x je: ");
Serial.println(ogranicena1);
int ogranicena2=constrain(y,10,20);
Serial.print("Posle ogranicenja broj y je: ");
Serial.println(ogranicena2);
//kvadriranje
int z=pow(y,2);
Serial.print("y na kvadrat je: ");
Serial.println(z);
float pwr=pow(2,1.0/5);
Serial.print("Peti koren iz 2 je: ");
Serial.println(pwr);
//korenovanje
Serial.println(sqrt(144));
//zaokruživanje float brojeva
Serial.println(floor(2.2));
Serial.println(ceil(-1.1));
Serial.println(ceil(1.989));
//korišćenje trigonometrijskih funkcija
float stepen=60;
float radijan=stepen*PI/180;
Serial.println(radijan);
Serial.println(sin(radijan));
Serial.println(cos(radijan));
//pomeranje bitova: 6=%0110, %1100=12 %0001=1
int a=6;
int rez=a<<1;
Serial.println(rez);
rez=a>>2;
Serial.println(rez);
//broj u različitim brojnim sistemima
Serial.println(x,DEC);
Serial.println(x,HEX);
Serial.println(x,OCT);
Serial.println(x,BIN);
}

void loop() {
}

```

Prilikom korišćenja funkcije Serial.print() moguće je navesti **brojni sistem u kome se želi prikaz zadate promenljive**. Ovde je ilustrovano kao se broj x štampa u dekadnom brojnom sistemu DEC, heksadecimalnom sistemu HEX, oktalnom OCT i binarnom BIN. Na slici 16 dat je prikaz rezultata izvršenja opisanog programa.



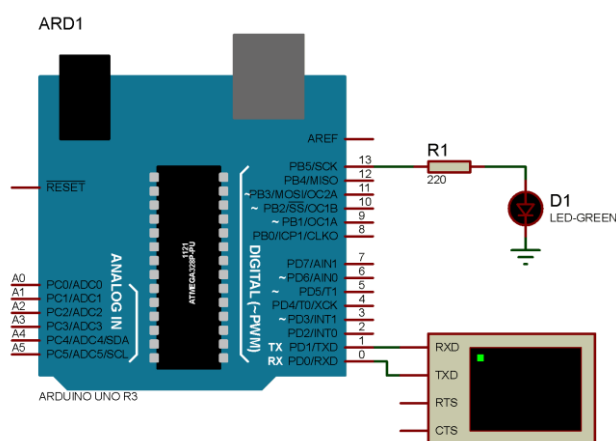
Slika 16. Prikaz rezultata izvršenja programa u virtuelnom terminalu

PROJEKAT 12.

Simulirati korišćenjem programa Proteus kolo Arduino na koji je povezana jedna LED dioda i vitruelni terminal. Napisati program za Arduino koji isčitava uneti karakter sa tastature i u zavisnosti od unete vrednosti menja period treptanja LED diode od 0- 900 ms. Korisnik unosi pomoću tastature cifru od 0 – 9, što treba da odgovara periodu treptanja od 0 – 900 ms.

Rešenje:

Električna šema Arduino sa LED diodom i virtuelnim terminalom data je na slici 17.



Slika 17. Električna šema Arduino sa LED diodom i virtuelnim terminalom

Nakon deklarisanja promenljive koja će čuvati period treptanja LED-a, u funkciji void setup() inicijalizovana je serijska komunikacija i deklarisan je LED pin kao izlazni. U glavnom programu, korišćenjem funkcije **Serial.available()** proverava se da li je stigao bar jedan podatak sa tastature, ako jeste u promenljivu tip karakter (char) t upisuje se vrednost pristiglog podatka korišćenjem funkcije **Serial.read()**.



Serial.available()

Vraća broj bajtova (karaktera) dostupnih za čitanje sa serijskog porta; podaci koji stignu se čuvaju u prijemnom baferu mikrokontrolera (64 bajta)

Serial.read()

Čita podatke stigle preko serijskog porta

Zatim se ispituje, ako je u promenljivu `t` upisan karakter koji je veći ili jednak '0' ili manji ili jednak '9' vrši se konverzija tog karaktera u celobrojnu vrednost i to se upisuje u promenljivu `blic`, pa se ta vrednost konačno pomnoži sa 100, da bi promenljiva `blic` čuvala period treptanja LED-a u milisekundama. Sve vrednosti u promenljivoj tipa karakter su u skladu sa ASCII vrednostima, zato će `t-'0'` dati celobroju vrednost koja odgovara unetom karakteru. Da bi se proverilo da li uneti karakter ispunjava zadate uslove koristi se **operator logičkog I (&&)**. Slično se može koristiti **operator logičkog I (||)**. Treba napomenuti da *uneti karakteri mogu biti bilo koji sa tastature (velika, mala slova, specijalni znakovi), a Arduino ih prepoznaje po ASCII vrednosti.*

Kada je u promenljivoj `blic` upisan period treptanja LED-a, na serijski terminal šalje se poruka o podešenom periodu i poziva se funkcija `blicFunkcija()` koja će izvršiti uključivanje/isključivanje LED-a sa zadatim periodom. U ovoj funkciji koriste se ranije objašnjene funkcije `digitalWrite()` i `delay()`.

```
const int ledPin=13;
int blic=0;
void setup() {
    Serial.begin(9600);
    pinMode(ledPin,OUTPUT);
}

void loop() {
    if(Serial.available()){
        char t=Serial.read();
        if(t>='0' && t<='9'){
            blic=t-'0';
            blic=blic*100;
        }
    }
    Serial.print("Podesen period treptanja je: ");
    Serial.print(blic);
    Serial.println(" ms.");
    blicFunkcija();
}

void blicFunkcija(){
    digitalWrite(ledPin,HIGH);
    delay(blic);
    digitalWrite(ledPin,LOW);
    delay(blic);
}
```

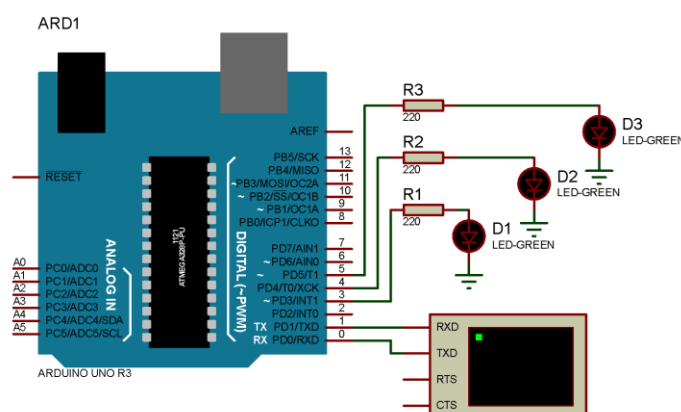
PROJEKAT 13.

Simulirati kolo Arduina sa tri LED diode i virtuelnim terminalom pomoću programa Proteus. Napisati program za Arduino koji vrši parsovanje (razdvajanje) stringa unetog preko tastature na dva dela. Smatra se da je kraj unetog stringa kada korisnik pritisne Enter.

Očekuje se da korisnik unosi ključnu reč “uključiti” ili “isključiti” i posle razmaka redni broj pina koji želi da uključi/isključi. Na primer: uključiti 4 (što će pin 4 postaviti na HIGH) ili isključiti 5 (što će pin 5 postaviti na LOW). Obezbediti da komande ne budu case sensitive. Ukoliko korisnik unese nepoznatu komandu, prikazati odgovarajuću poruku.

Rešenje:

Simulaciona šema Arduina sa tri LED-a i virtuelnim terminalom prikazana je na slici 18.



Slika 18. Električna šema Arduina sa tri LED-a i virtuelnim terminalom

String *komanda* se deklarise na početku i u njega će biti smeštena uneta komanda. U okviru funkcije void setup() deklarisan je serijska komunikacija i podešeni pinovi 3, 4, 5 kao izlazni.

U glavnom programu, ako je pristigao karakter, isčitava se i upisuje u promenljivu c, tipa char. Ako je unet **Enter** ('\\r'), prikazuje se unet string i poziva funkcija za parsovanje (razdvajanje) stringova, da bi se iza toga obrisao prethodno uneti string, koji je obrađen. Inače, ako korisnik unosi karaktere oni se jedan po jedan smeštaju u string, redom kako pristižu, pomoću komanda=komanda+c, tj. komanda+=c. Napomenimo, string je niz karaktera.

Funkcija za parsovanje kao argument prihvata uneti string komanda, koji se smešta u string kom. U okviru funkcije deklarise se dva stringa deo1 i deo2 koji će čuvati različite delove unetog stringa posle razdvajanja. U string deo1 upisuje se prvi deo stringa kom, od početka do pojave razmaka u unetom stringu. To se postiže funkcijom **kom.substring(0, kom.indexOf(" "))**. Pojava razmaka u stringu dobija se korišćenjem funkcije kom.indexOf(" ") koja vraća indeks, redni broj karaktera u nizu koji je prazno mesto. U string deo2 upisuje se drugi deo stringa kom, od mesta gde je bio razmak i plus jedna pozicija, pa do kraja unetog stringa. To se postiže funkcijom **kom.substring(kom.indexOf(" ")+1)**. Da nije uzeto +1 mesto, u string deo2 bilo bi upisano "drugi_deo_stringa", dakle u taj deo stringa ušao bi i razmak.

Nakon što je uneti string razdvojen na dva dela, prvi koji čuva akciju koju treba preduzeti (uključiti/isključiti) i drugi koji čuva broj pina nad kojim treba preduzeti zadatu akciju, sledi ispitivanje da li je u deo1 uneto "uključiti". Da bi se obezbedilo da komande nisu case sensitive, koristi se funkcija **equalsIgnoreCase()**. Ako je string deo1 jednak uključiti, bez obzira

da li je naredba uneta velikim/malim slovima ili kombinacijom velikih i malih slova, u promenljivu pin upisuje se broj pina koji treba da se bude uključen. To se postiže konvertovanjem stringa deo2 u ceo broj, što se postiže funkcijom **deo2.toInt()**. Korišćenjem funkcije digitalWrite(pin,HIGH), uneti pin se postavlja na logičku jedinicu. Inače, ako je string deo1 jednak "iskljuci", tada se u promenljivu pin upisuje broj pina koji treba da bude podešen na logičku nulu, što se postiže funkcijom konvertovanja stringa deo2 u broj – deo2.toInt(). Na kraju, ako nije uneta ispravna komanda, na virtuelni terminal šalje se poruka "Nepoznata komanda".

```
String komanda;
void setup() {
    Serial.begin(9600);
    pinMode(3,OUTPUT);
    pinMode(4,OUTPUT);
    pinMode(5,OUTPUT);
}

void loop() {
    if(Serial.available()) {
        char c=Serial.read();
        if(c=='\r') {
            Serial.print("Uneli ste: ");
            Serial.println(komanda);
            parsovanje(komanda);
            komanda="";
        }
        else{
            komanda+=c;
        }
    }
}

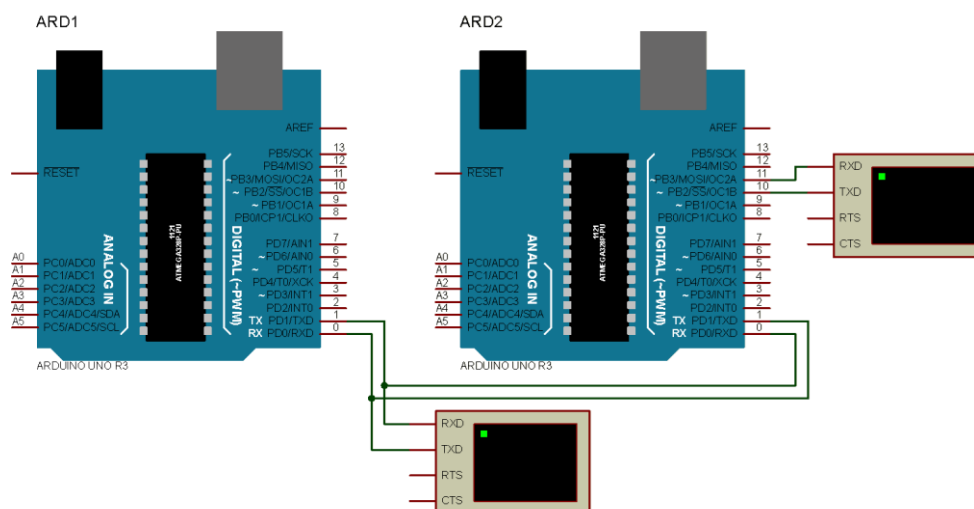
void parsovanje(String kom) {
    String deo1;
    String deo2;
    deo1=kom.substring(0, kom.indexOf(" "));
    deo2=kom.substring(kom.indexOf(" ")+1);
    if(deo1.equalsIgnoreCase("ukljuci")) {
        int pin=deo2.toInt();
        digitalWrite(pin,HIGH);
    }
    else if(deo1.equalsIgnoreCase("iskljuci")) {
        int pin=deo2.toInt();
        digitalWrite(pin,LOW);
    }
    else{
        Serial.println("Nepoznata komanda");
    }
}
```

PROJEKAT 14.

Simulirati kolo sa dva Arduina koja komuniciraju preko UART-a korišćenjem programa Proteus. Napisati programe za Arduino: prvi Arduino šalje poruku kao string na svaku sekundu, a drugi Arduino prima poruku i prosleđuje je na softverski UART.

Rešenje:

Simulaciona šema dva Arduina koja komuniciraju preko UART-a data je na slici 19.



Slika 19. Električna šema dva Arduina koja komuniciraju preko UART-a

Prenos podataka preko magistrala (skupa žica), tj. **komunikacija** može biti **paralelna** ili **serijska**. U paralelnoj komunikaciji prenosi se veliki broj podataka istovremeno, najčešće preko 8, 16 ili 32 linija. Iako je ovaj tip komunikacije brz, zahteva veliki broj konekcija, pa se zato uglavnom sreće unutar integrisanih kola. Za povezivanje uređaja uglavnom se koristi serijska komunikacije gde se podaci prenose bit po bit. Brzina prenosa podataka je manja nego kod paralelnog prenosa, ali se koristi manji broj linija veze (1-4). Da bi se postigla sinhronizacija u prenosu podataka jedna od linija na magistrali je takt (clock). To su sinhronne serijske magistrale. Postoje i asinhronne serijske magistrale, gde nema prenosa taktnog signala, već sami uređaju koju komuniciraju imaju integrisana kola za sinhronizaciju. Komunikacija može biti: **half-duplex**, odvija se u oba smera, ali ne istovremeno (jedan uređaj priča, drugi sluša); **full-duplex**, odvija se istovremeno u oba smera; **simplex**, odvija se u jednom smeru. Neke od serijskih magistrala su: **UART, I2C, SPI**.

UART (Universal Asynchronous Receiver-Transmitter) je asinhrona serijska magistrala, koja može biti full-duplex, half-duplex i simplex, a kod Arduina se najčešće koristi full-duplex sa dve linije veza: TX (transmit line) – preko koje se šalju podaci i RX (receive line) – preko koje se primaju podaci. TX i RX pinovi kod Arduina su 1 i 0.

ARD1 će slati poruku ka ARD2. Zato je u okviru funkcije void setup() inicijalizovana serijska komunikacija i Timer1. Na svaku sekundu, što je podešeno preko tajmera, desiće se prekid i u prekidnoj rutini biće poslata poruka u vidu stringa.

```
#include <TimerOne.h>
void setup() {
    Serial.begin(9600);
    Timer1.initialize(1000000);
    Timer1.attachInterrupt(porukaISR);
}
void loop() {
}
```

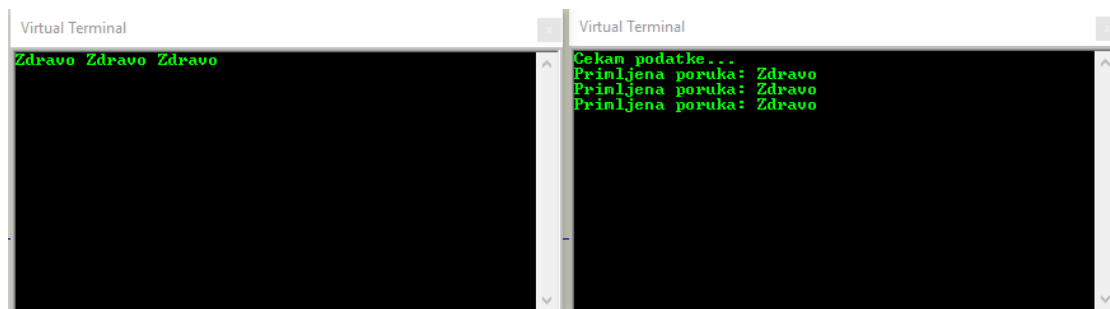
```
void porukaISR() {  
    Serial.print("Zdravo ");  
}
```

ARD2 treba da prihvati poruku i pošalje je na virtuelni terminal. S obzirom da ARD2 preko hardverskog UART-a na pinovima 0 i 1 prima podatke od ARD1, za prikaz primljene poruke biće iskorišćeni, tzv. **softverski UART**. Naime, razvijena je biblioteka **SoftwareSerial.h** koja omogućava slanje i prijem podataka preko drugih digitalnih pinova. U ovom projektu izabrani su pinovi 10 za RX i 11 za TX, tako je pin 10 vezan za TX od virtuelnog terminala, odnosno 11 za RX od virtuelnog terminala.

Pre funkcije void setup() deklarirane su dve promenljive: i, tipa char koja će prihvatati karaktere koji stižu preko UART-a iz ARD1 i poruka, tipa string, koji će čuvati primljenu poruku. U okviru void setup() deklarirani su baud-rate za oba UART-a i preko softverskog UART-a je poslata poruka "Cekam podatke..." na virtuelni terminal.

U glavnom programu, kada pristigne karakter preko hardverskog UART-a smešta se u promenljivu i. Ispituje se da li je pristigao karakter – prazno mesto (space), ako jeste na virtuelni terminal se preko softverskog UART-a šalje pristigla poruka i briše se upravo prikazana poruka iz stringa. Inače, ako još nije stigao kraj stringa – prazno mesto, u string poruka se upisuje karakter po karakter, redom kao što pristižu preko hardverskog UART-a. Na ovaj način se rekonstruiše poruka koja je stigla. Prikaz rezultata u virtuelnom monitoru dat je na slici 20.

```
#include <SoftwareSerial.h>  
SoftwareSerial mojUART(10,11);  
char i;  
String poruka;  
void setup() {  
    Serial.begin(9600);  
    mojUART.begin(9600);  
    mojUART.println("Cekam podatke...");  
}  
void loop() {  
    if(Serial.available()){  
        i=Serial.read();  
        if(i==' '){  
            mojUART.print("Primljena poruka: ");  
            mojUART.println(poruka);  
            poruka="";  
        }  
        else{  
            poruka+=i;  
        }  
    }  
}
```



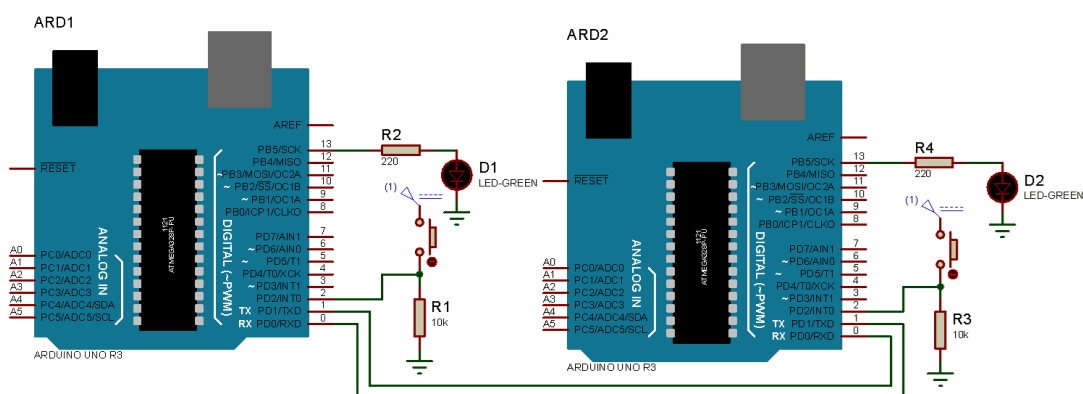
Slika 20. Prikaz rezultata izvršenja programa u ARD1 (levo) i ARD2 (desno)

PROJEKAT 15.

Simulirati kolo sa dva Arduina koja komuniciraju preko UART-a u programu Proteus. Na svaki od Arduina vezan je po jedan taster u *pull-down* konfiguraciji i jedna LED dioda. Napisati program za Arduino korišćenjem mehanizma prekida kojim se tasterom menja stanje LED-u na drugom Arduinou. Isti program učitati na oba mikrokontrolera.

Rešenje:

Simulaciona šema dva Arduina sa tasterom u *pull-down* konfiguracija i LED-om koji međusobno komuniciraju preko UART-a prikazana je na slici 21.



Slika 21. Električna šema dva Arduina sa tasterom i LED-om koji komuniciraju preko UART-a

Na početku su deklarisan konstante i promenljive koje čuvaju stanje tastera, LED-a, poruku koja se šalje/prima. U funkciji void setup() inicijalizovana je serijska komunikacija, deklarisan pinovi i podešena ISR koja se vezuje za promenu na tasteru sa stanja logičke nule na stanje logičke jedinice.

Kada se pritisne taster, dolazi do prekida glavnog programa i izvršava se ISR tasterISR() u okviru koje se vrši debouncing tastera tako što se sačeka određeno vreme, a zatim ponovo isčita stanje tastera. Ako je stanje tastera posle tog vremena i dalje HIGH, smatra se da je u stabilnom stanju i tada se na serijski terminal šalje '1', što će signalizirati drugom Arduinou da treba da uključi/isključi LED.

U beskonačnoj petlji void loop() se čeka prijem poruke preko UART-a od drugog mikrokontrolera. Kada informacija stigne upisuje se u promenljivu poruka, tipa char. Zatim

se proverava da je pristigla poruka jednaka '1', ako jeste menja se vrednost promenljive ledStanje, korišćenjem operatora inverzije (!), i novo stanje prosleđuje na pin korišćenjem funkcije digitalWrite().

S obzirom da su komponente povezane na iste pinove na oba Arduina, isti kod će raditi na oba mikrokontrolera. Kada se pritisne taster na prvom Arduinu, menja se stanje LED-a na drugom i obrnuto, kada se pritisne taster na drugom Arduino, menja se stanje LED-a na prvom.

```
const int ledPin=13;
const int tasterPin=2;
int tasterStanje;
int ledStanje=LOW;
char poruka;
int debounceVreme=200;
void setup() {
    Serial.begin(9600);
    pinMode(ledPin,OUTPUT);
    pinMode(tasterPin,INPUT);
    digitalWrite(ledPin,ledStanje);
    attachInterrupt(digitalPinToInterrupt(tasterPin),tasterISR,RISING);
}

void tasterISR(){
    delay(debounceVreme);
    tasterStanje=digitalRead(tasterPin);
    if(tasterStanje==HIGH){
        Serial.print('1');
    }
}

void loop() {
    if(Serial.available()){
        poruka=Serial.read();
        if(poruka=='1'){
            ledStanje=!ledStanje;
            digitalWrite(ledPin,ledStanje);
        }
    }
}
```

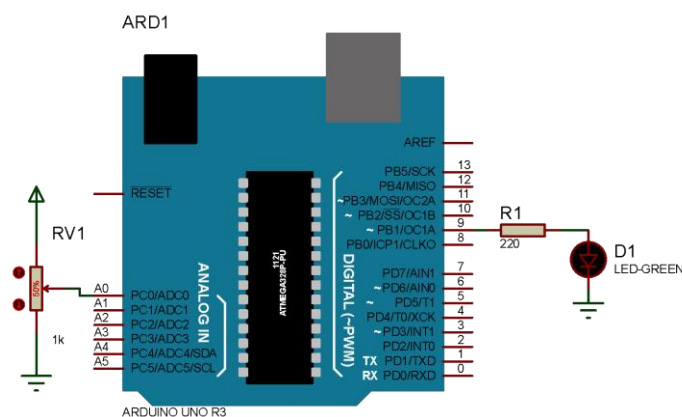

Analogni ulazni pinovi

PROJEKAT 16.

Simulirati kolo Arduino sa potencijetrom i LED diodom korišćenjem programa Proteus. Napisati program za Arduino koji isčitava vrednost sa potencijetra i u zavisnosti od očitane vrednosti menja PWM signal na LED-u.

Rešenje:

Simulaciona šema Arduino sa potencijetrom i LED diodom data je na slici 22. Vrednost napona na potencijetru menja se od 0 – 5V, tako da se za isčitavanje analogne vrednosti napona mora koristiti analogni ulaz mikrokontrolera. Arduino ima **6 analognih ulaznih pinova (A0-A5)**, na svakom od njih je **10-bitni AD (analogno-digitalni) konvertor**, tako da se prilikom isčitavanja dobija vrednost od 0 – 1023 ($1024=2^{10}$). Maksimalni napon koji sme da se dovede na analogni pin je 5V. **Arduino analogni pinovi mogu biti samo ulazni, tako da ih nije potrebno deklarirati funkcijom `pinMode()`**. LED dioda vezana je na PWM pin, s obzirom da je potrebno menjati širinu trajanja impulsa na LED-u.



Slika 22. Električna šema Arduino sa potencijetrom i LED diodom

Nakon deklarisanja konstanti i promenljivih i podešavanja pina gde je povezan LED, u glavnom programu deklarirše se celobrojna promenljiva pot u koju se upisuje vrednost pročitana sa analognog pina. Za čitanje analognih pinova koristi se funkcija **`analogRead(pin)`**. Argument ove funkcije je redni broj pina sa koga se vrši čitanje. Kao što je napomenuto, rezultat izvršenja ove funkcije je broj od 0 do 1023, jer Arduino ima 10-tobitni ADC.



```
analogRead(pin)
```

Čita vrednost sa određenog analognog pina (A0-A5)

pin: broj analognog pina

S obzirom da se PWM signal zadaje preko osmootne vrednosti (0-255), potrebno je vrednost očitane sa potencijetra preskalirati na ovaj opseg. Za skaliranje se može koristiti funkcija **`map(vrednost, vr1, vr2, sk1, sk2)`**, čiju su argumenti vrednost promenljive koja se skalira, vr1, vr2 – minimalna i maksimalna vrednost promenljive, sk1, sk2 – minimalna i maksimalna vrednost za skaliranje (željene vrednosti). Na primer, u ovom projektu vrednost promenljive

pot kreće se od 0 do 1023, a biće linearno preskalirana na vrednost od 0 do 255. Rezultat skaliranja biće upisan u promenljivu pwmVrednost, koja je ranije deklarirana kao int. Korišćenjem funkcije analogWrite() na definisani pin, prosleđuje se vrednost koja će odrediti PWM signal.



`map(vrednost, vr1, vr2, sk1, sk2)`

Linearno preslikava (skalira) vrednost sa trenutnog na željeni opseg
vrednost: vrednost promenljive koja se skalira
vr1: minimalna trenutna vrednost
vr2: maksimalna trenutna vrednost
sk1: minimalna željena vrednost
sk2: maksimalna trenutna vrednost

```
const int potPin=A0;
const int ledPin=9;
int pwmVrednost;
void setup() {
    pinMode(ledPin,OUTPUT);
}

void loop() {
    int pot=analogRead(potPin);
    pwmVrednost=map(pot,0,1023,0,255);
    analogWrite(ledPin,pwmVrednost);
}
```

PROJEKAT 17.

Simulirati kolo Arduina sa temperaturnim senzorom TMP36, RGB diodom i tri LED-a korišćenjem programa Proteus. Napisati program za Arduino koji isčitava senzor i u zavisnosti od očitane temperature vrši indikaciju temperature. Ako je temperatura manja od 18°C RGB treba da svetli plavo, i treba da bude uključena jedna LED dioda; ako je temperatura između 18°C i 23°C RGB treba da svetli zeleno i treba da budu uključene dve LED diode; a ako je temperatura veća ili jednaka 23°C RGB treba da svetli crveno i treba da budu uključene sve tri diode. Senzor iščitavati na svakih 100 ms.

Rešenje:

Simulaciona šema Arduina sa temperaturnim senzorom TMP36, RGB diodom sa zajedničkom katodom i tri LED-a data je na slici 23.

Senzor temperature treba periodično iščitavati pa je neophodno korišćenje tajmera, zato je na početku uključena biblioteka za rad sa Timer1. Zatim su naredbom **#define** definisani korisnički nazivi pinova i njima dodeljene stvarne oznake na Arduinu. Na ovaj način štedi se memorija, jer se ne alokira memorija za promenljive koje bi čuvalе ove vrednosti. Iščitavanje senzora treba da bude na svakih 100 ms, pa je zato deklarirana konstanta tipa unsigned long koja čuva vrednost vremena očitavanja izraženo u mikrosekundama. Na početku programa deklarirana je float promenljiva temperatura čija će se vrednost koristiti u više funkcija u programu pa je zato volatile. Ostale promenljive su senzor, tipa integer i napon, tipa float.


```

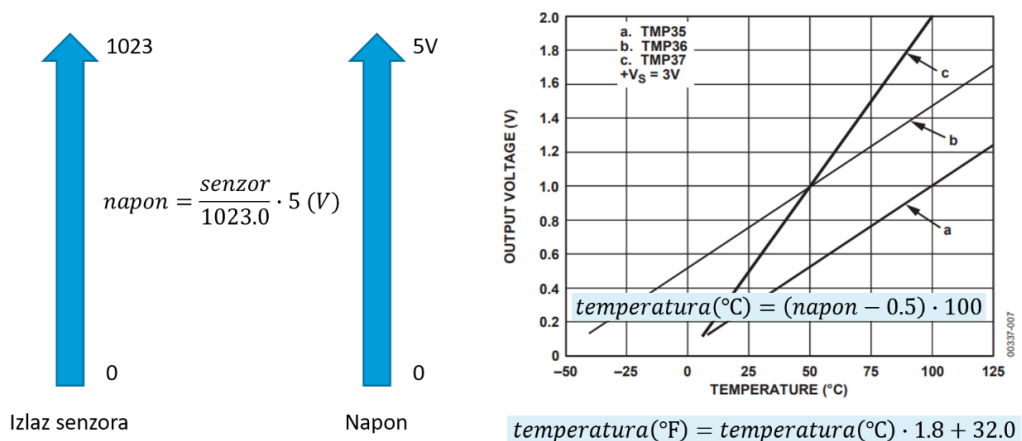
    digitalWrite(zelena, LOW);
    digitalWrite(plava, LOW);
}

digitalWrite(led1, HIGH);
if (temperatura >= 18.0) {
    digitalWrite(led2, HIGH);
    if (temperatura >= 23) {
        digitalWrite(led3, HIGH);
    }
    else {
        digitalWrite(led3, LOW);
    }
}
else {
    digitalWrite(led2, LOW);
}
}
}

```

U funkciji void setup() kroz dve for petlje podešeni su pinovi kao izlazni i inicijalno je njihova vrednost postavljena na LOW. Takođe, u okviru ove funkcije inicijalizovan je tajmer i tajmeru je dodeljena ISR pod nazivom senzorISR, koja će prekinuti izvršenje glavnog programa na svakih 100 ms radi čitanja senzora. Ovo vreme je moguće menjati, promenom vrednosti konstante vremeOcitavanja.

Postupak određivanja vrednosti temperature na osnovu očitavanja senzora prikazan je na slici 24. U ISR-u se na svakih 100 ms, u celobrojnu promenljivu senzor, korišćenjem funkcije analogRead(), upiše očitana vrednost. Očitana vrednost biće iz opsega 0-1023. Zatim je potrebno odrediti realnu vrednost napona na izlazu senzora, s obzirom da proizvođači u tehničkoj dokumentaciji daju zavisnost temperature u funkciji napona na senozru. Postavljanjem proporcije, napon se određuje kao $\text{senzor} \cdot 5.0 / 1024.0$. Ova vrednost upisuje se u promenljivu tipa float. S obzirom da je promenljiva senzor tipa int, a rezultat izračunavanja će biti realan broj (float) prilikom pisanja konstanti u izrazu za napon treba pisati realne brojeve (5.0, a ne samo 5). Na osnovu zavisnosti izlaznog napona senzora i temperature (slika 24), postavljanjem jednačine prave dobija se izraz za temperaturu: $(\text{napon} - 0.5) \cdot 100.0$. Na slici 24 dat je i izraz za konverziju temperature iz °C u °F.

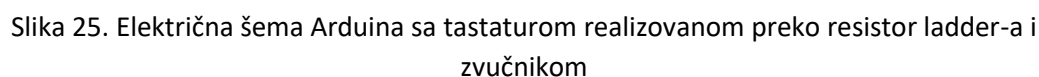


Slika 24. Postupak određivanja vrednosti temperature na osnovu očitavanja senzora

Indikacija pomoću LED-ova realizovana je na sledeći način: prvi LED u nizu je uvek uključen, ispituje se da li je temperatura veća ili jednaka 18 stepeni, ako jeste uključuje se i sledeći LED u nizu, pa se zatim ispituje i da li je temperatura veća ili jednaka od 23 stepena, ako jeste uključuje se i treći LED u nizu. Inače, ako poslednji uslov nije ispunjen, treći LED u nizu se postavlja u stanje LOW, a ako je temperatura manja od 18, isključuje se i drugi LED u nizu, te uvek ostaje prvi LED uključen.

Simulirati kolo Arduina sa tastaturom realizovanom preko *resistor ladder-a* i zvučnikom korišćenjem programa Proteus. Napisati program za Arduino koji u zavisnosti od pritisnutog tastera u tastaturi na zvučniku emituje ton različite frekvencije (klavijatura). Klavijaturu čine tri tastera, ako je pritisnut taster T1 emituje se ton frekvencije 262 Hz (do), ako je pritisnut T2 emituje se ton frekvencije 294 Hz (re) i ako je pritisnut T3 emituje se ton frekvencije 330 Hz (mi).

Simulaciona šema kola Arduino sa tastaturom realizovanom preko *resistor ladder-a* i zvučnikom data je na slici 25. U zavisnosti koji je taster pritisnut napon koji se isčitava na A0 biće različit. Ako je pritisnut taster T1, na A0 se dovodi napon napajanja (5V), što znači da će biti očitana vrednost 1023. Ako je pritisnut taster T2, otpornici R2 i R1 formiraju razdelnik napona. Kako je $R2=R1$, to će napon na A0 biti pola napona napajanja (2.5V), što znači da će biti očitana vrednost oko 512. U zavisnosti od tolerancije otpornika ovaj napon može da varira, zato će očitana vrednost biti oko 512. Ako je pritisnut taster T3, formira se razdelnik napona koga čine redna veza otpornika R3 i R2 i otpornik R1, pa će napon biti $(5/3)V$, što znači da će očitavanje biti oko 342. Otpornik R1 je *pull-down* otpornik, kada ni jedan taster nije pritisnut, A0 će preko njega biti na masi (0V). Zvučnik je povezan na PWM pin 9.



Frekvencije tonova zadate su u *nizu zvuk[]*, *indeksiranje u nizu počinje od 0*. Na početku programa definisani su pinovi za tastaturu i zvučnik. U beskonačnoj petlji iščitava se vrednost sa tastature korišćenjem funkcije `analogRead()` i smešta u promenljivu `t`. Zatim se ispituju opseg vrednosti `t`, ako je pritisnut prvi taster, tada će korišćenjem funkcije **`tone(pin,frekvencija)`**, na odgovarajući pin emituje ton zadate frekvencije. S obzirom da su frekvencije zadate u nizu, funkcija uzima prvi element niza, indeksiran sa 0. Ako je pritisnut taster T2, što se utvrđuje proverom da li je `t` u opsegu od 510 do 515, emituje se ton `re`. Zbog mogućeg variranja očitane vrednosti, uzeti su dati opsezi. Slično, ako korisnik pritisne taster T3, biće emitovan ton `mi`. Ako nijedan od tastera nije pritisnut, tj. nije ispunjen ni jedan od ispitanih uslova nema emitovanja zvuka, što se postiže funkcijom **`noTone(pin)`**. Argument ove funkcije je broj pina koji treba postaviti na LOW, kako ne bi bilo emitovanja zvuka.



`tone(pin, frekvencija, trajanje)`

Generiše povorku impulsa zadate frekvencija, pri čemu je duty cycle signala 50%

pin: broj pina

frekvencija: frekvencija tona u Hz (dozvoljeni tip podatka unsigned int)

trajanje: trajanje tona u milisekundama – opciono (dozvoljeni tip podatka unsigned long)

`noTone(pin)`

Zaustavlja generaciju povorke impulse zadatih funkcijom `tone()`

pin: broj pina

```
int zvuk[]={262,294,330};
#define taster A0
#define zvukPin 9
void setup() {
}

void loop() {
    int t=analogRead(taster);
    if(t==1023){
        tone(zvukPin,zvuk[0]);
    }
    else if(t>=510 && t<=515){
        tone(zvukPin,zvuk[1]);
    }
    else if(t>=340 && t<=345){
        tone(zvukPin,zvuk[2]);
    }
    else{
        noTone(zvukPin);
    }
}
```

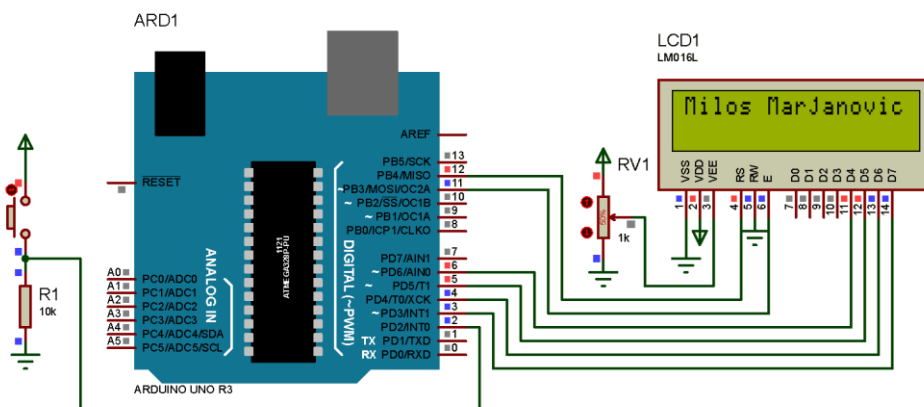
Displeji

PROJEKAT 19.

Simulirati kolo Arduina sa LCD displejem i tasterom u *pull-down* konfiguraciji u programu Proteus. Napisati program za Arduino koji na svaki pritisak tastera generiše random broj od 1-6 i prikazuje ga na LCD displeju (elektronska simulacija kockice za igru). Na početku se pokazuje pozdravna poruka i vrši se skrolovanje poruke po ekranu. Koristiti mehanizam prekida.

Rešenje:

Električna šema Arduina sa LCD displejem i tasterom u *pull-down* konfiguraciji prikazana je na slici 26. U ovom projektu biće korišćen **LCD (Liquid Crystal display) 16x2**, što znači da može prikazivati 2 reda od po 16 alfa-numeričkih karaktera. Izvodi LCD displeja su: VSS – masa; VDD – napajanje, najčešće 5V; VEE=V0 pin za podešavanje kontrasta displeja (kontrast se podešava korišćenjem potencijometra); RS (register select) pin kontroliše prenos podataka- kada je na logičkoj jedinici prenose se podaci, a kada je na logičkoj nuli prenose se komande; R/W (read/write) pin se postavlja na masu, zato što se vrši samo pisanje u displej; E (enable) pin služi za dozvolu rada; D0-D7 se koriste za prenos podataka. Najčešće se koristi 4-bitni prenos podataka. Iskorišćeni su digitalni pinovi Arduina za povezivanje sa LCD-om. LCD displeji sa pozadinskim osvetljenjem imaju i dva dodatna pina LED+ i LED- koji kontrolišu pozadinsko osvetljenje: LED+ se preko otpornika za ograničenje struje povezuje na napajanje, a LED- na masu. Svaki od segmenata LCD-a je sastavljen iz matrice, najčešće 7x5 ili 8x5, polja ispunjena tečnim kristalom.



Slika 26. Električna šema Arduina sa LCD displejem i tasterom u *pull-down* konfiguraciji

Za korišćenje LCD displeja u Arduino potrebno je uključiti biblioteku **LiquidCrystal.h**. Biblioteka omogućava Arduino da kontroliše LCD-ove bazirane na Hitachi HD44780 čipsetu. U nastavku su deklarisan konstante (oznake pinova na koje su vezani taster i LCD) i promenljiva tipa String koja će čuvati poruku koja se ispisuje na ekranu. Zatim se kreira **LCD objekat** (imenovana instanca za rad sa LCD bibliotekom), u ovom projektu **mojLCD** i deklariraju se pinovi na koje je povezan.

U okviru funkcije `void setup()` pin na koji je vezan taster podešava se kao ulazni i za njega se vezuje prekid. Svaki put kada bude pritisnut taster biće prekinuto izvršavanje glavnog programa i izvršiće se `ISR: ispisiNaLCD()`. Takođe, ovde je inicijalizovan LCD, korišćenjem funkcije **`mojLCD.begin(16,2)`** – što definiše da se koristi LCD 16x2. Korišćenjem funkcije **`mojLCD.setCursor(X,Y)`** postavlja se kursor u prvi red, prva pozicija. X predstavlja poziciju u redu, Y red, s tim da indeksiranje počinje od 0. (0,1) znači da se kursor postavlja na prvu poziciju u drugom redu, a (1,0) znači da se kursor postavlja na drugu poziciju u prvom redu. Korišćenjem funkcije **`mojLCD.print(podaci)`** šalju se podaci koje treba prikazati na LCD-u.

Za skrolovanje teksta po LCD-u koriste se funkcije **`mojLCD.scrollDisplayRight()`** – koja vrši pomeranje teksta za jednu poziciju u desno i **`mojLCD.scrollDisplayLeft()`** – koja vrši pomeranje teksta za jednu poziciju u levo. Da bi se tekst skrolovaao po ekranu neophodno je korišćenje for petlji. U funkciji `mojLCD.setCursor()`, argumenti mogu biti i promenljive, što dodatno doprinosi efektu skrolovanja teksta.

Brisanje sadržaja sa LCD-a postiže se korišćenjem funkcije **`mojLCD.clear()`**.



```
#include < LiquidCrystal .h>
```

Uključivanje biblioteke za rad sa LCD

```
LiquidCrystal mojLCD(rs, en, d4, d5, d6, d7)
```

Kreira promenljivu tipa LiquidCrystal, na primer *mojLCD*

rs: broj Arduino pina koji je povezan na RS pin LCD-a

en: broj Arduino pina koji je povezan na Enable pin LCD-a

d4, d5, d6, d7: brojevi Arduino pinova koji su povezani na odgovarajuće pinove LCD-a za prenos podataka

```
mojLCD.begin(X, Y)
```

Inicijalizuje interfejs LCD-a i specificira dimenzije LCD-a; ova funkcija mora biti pozvana pre bilo koje druge LCD funkcije

mojLCD: promenljiva tipa LiquidCrystal

X: broj kolona koje LCD ima

Y: broj redova koje LCD ima

```
mojLCD.setCursor(x, y)
```

Pozicionira kursor na LCD-u – podešava lokaciju od koje će krenuti ispisivanje podatka

mojLCD: promenljiva tipa LiquidCrystal

x: kolona na koju treba pozicionirati kursor (prva kolona označena je nulom)

y: red na koji treba pozicionirati kursor (prvi red označen je nulom)

```
mojLCD.print(podaci, BAZA)
```

Štampa tekst na LCD-u

mojLCD: promenljiva tipa LiquidCrystal

podaci: podaci koje treba odštampati (char, byte, int, long, String)

BAZA: brojni sistem u kome se štampaju brojevi (BIN, DEC, OCT, HEX) - opciono

```
mojLCD.scrollDisplayRight()
```

Skroluje sadržaj displeja (tekst i kursor) za jednu poziciju desno

```
mojLCD.scrollDisplayLeft()
```

Skroluje sadržaj displeja (tekst i kursor) za jednu poziciju levo

```
mojLCD.clear()
```

Briše LCD ekran i pozicionira kursor u gornji levi ugao (0,0)


```

#include <LiquidCrystal.h>
#define tasterPin 2
const int rs=12, en=11, d4=6, d5=5, d6=4, d7=3;
String poruka="";
LiquidCrystal mojLCD(rs,en,d4,d5,d6,d7);
void setup() {
    pinMode(tasterPin,INPUT);
    attachInterrupt(digitalPinToInterrupt(tasterPin),ispisiISR,RISING);
    mojLCD.begin(16,2);
    mojLCD.setCursor(0,0);
    mojLCD.print("Milos Marjanovic");
    delay(2000);
    for(int i=0;i<16;i++){
        mojLCD.scrollDisplayRight();
        mojLCD.setCursor(i,1);
        mojLCD.print("*");
        delay(150);
    }
    for(int i=0;i<16;i++){
        mojLCD.scrollDisplayLeft();
        delay(150);
    }
    mojLCD.clear();
    mojLCD.print("Bacite kockicu");
}

void ispisiISR(){
    mojLCD.clear();
    int broj=random(6);
    switch(broj){
        case 0:
            poruka="1";
            break;
        case 1:
            poruka="2";
            break;
        case 2:
            poruka="3";
            break;
        case 3:
            poruka="4";
            break;
        case 4:
            poruka="5";
            break;
        case 5:
            poruka="6";
            break;
    }
    mojLCD.setCursor(0,0);
    mojLCD.print("Bacili ste ");
    mojLCD.setCursor(11,0);
    mojLCD.print(poruka);
}

void loop() {
}

```

U ovom projektu koristi se mehanizam prekida, tako da je glavna beskonačna petlja prazna. Svaki puta kada se pritisne taster izvršava se prekidna rutina u okviru koje se prvo obriše

sadržaj ekrana, zatim se u promenljivu broj upiše neki random broj od 0-5, korišćenjem funkcije random(). Da bi se odredilo koja će poruka biti prikazana koristi se **switch-case struktura**. Switch uzima generisani broj i ako je vrednost 0 u string poruka upisuje se "1" i prekida se dalje ispitivanje, odnosno program izlazi iz switch naredbe. Slično, ako je broj=1, u string poruka upisuje se "2", itd. Na kraju ISR-a na LCD se šalje string "Bacili ste ", a u nastavku se štampa vrednost promenljive tipa String, poruka, korišćenjem funkcije mojLCD.print().



```
switch(promenljiva){
  case labela1:
    //naredbe
    break;
  case labela2:
    //naredbe
    break;
  default:
    //naredbe
    break;
}
```

Switch case kontroliše tok programa dozvoljavajući programerima da specificiraju različit kod koji treba da se izvrši pri različitim uslovima; switch naredba poredi vrednost promenljive sa vrednostima specificiranim u case naredbi; kada se pronađu vrednosti koje se podudaraju, izvršava se samo kod u case-u u kome postoji podudaranje; ključna reč break prekida izvršenje switch naredbe i tipično se koristi na kraju svakog case-a; ako nije pronađeno podudaranje izvršava se kod napisan iza default: (opciono)
 promenljiva: promenljiva čija vrednost se poredi sa različitim case-ovima (dozvoljeni tip podatka: int, char)
 labela1, labela2: konstante (dozvoljeni tip podatka: int, char)

PROJEKAT 20.

Simulirati kolo Arduina sa LCD displejem i potencijetrom u razdelniku napona u programu Proteus. Napisati program za Arduino koji određuje nepoznatu otpornost potencijetra i prikazuje je na displeju (ommetar). Za prikaz jedinice om kreirati poseban karakter. Akviziciju podataka sa razdelnika napona raditi periodično svake sekunde.

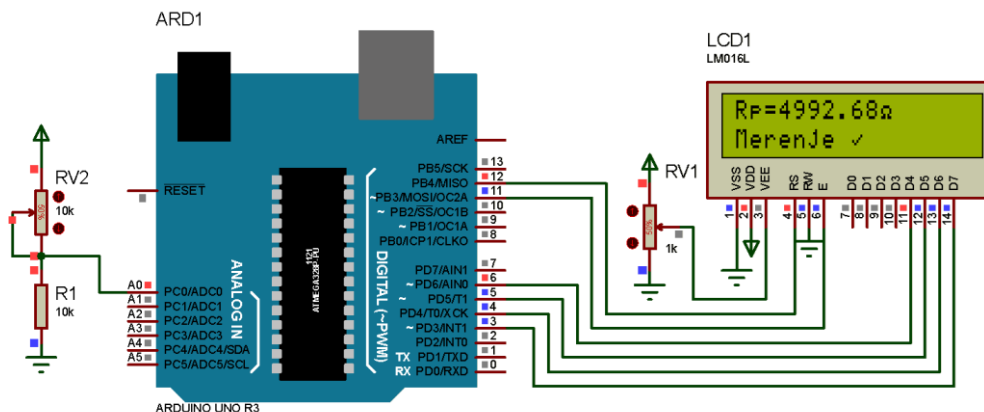
Rešenje:

Simulaciona šema Arduina sa LCD-om i potencijetrom u razdelniku napona data je na slici 27. S obzirom da će se u zavisnosti od otpornosti potencijetra menjati napon razdelnika napona, kolo je povezano na analogni ulaz A0. Napon na analogom ulazu će biti:

$$V_{A0} = \frac{R1}{R1 + RV2} 5.$$

Odavde se za nepoznatu otpornost dobija:

$$RV2 = R1 \left(\frac{5}{V_{A0}} - 1 \right).$$



Slika 27. Električna šema Arduina sa LCD-om i potencijetrom u razdelniku napona

Biblioteke za LCD i Timer1 koji će se koristiti u periodičnom isčitavanju napona sa razdelnika napona uključene su na početku programa. Definisani su pinovi i deklarirana promenljiva koja će čuvati vrednost nepoznate otpornosti, koja će se koristiti u više funkcija pa je zato volatile. Takođe, deklarirana je konstanta koja predstavlja period tajmera, kako očitavanje treba vršiti svake sekunde, potrebno je ovaj period podesiti na 1.000.000 μ s (1 s), pa je zbog velike vrednosti deklarirana kao unsigned long.

U funkciji void setup() posle inicijalizacije LCD-a kreiraju se specijalni karakteri. Za kreiranje specijalnih karaktera koristi se funkcija **mojLCD.createChar(broj,podaci)**. Argumenti ove funkcije su: broj – redni broj karaktera, moguće je kreirati ukupno 8 specijalnih karaktera i podaci – podaci o pikselima karaktera. Podaci o pikselima karaktera specificiraju se kao nizovi od 8 bajtova, po jedan za svaki red. Pet bitova najmanje težine u svakom bajtu određuju piksele u tom redu. Ovde će biti kreirana 4 specijalna karaktera, dimenzija 7x5 (slika 28).

0	0	0	0	0
0	0	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	0	1
0	1	0	1	0
1	1	0	1	1

0	0	0	0	0
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
1	0	1	0	0
0	1	0	0	0
0	0	0	0	0

0	0	0	0	0
1	0	0	0	1
0	0	0	0	0
0	0	0	0	0
1	0	0	0	1
0	1	1	1	0
0	0	0	0	0

0	0	0	0	0
1	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	1	1	1	0
1	0	0	0	1
0	0	0	0	0

Slika 28. Kreiranje specijalnih karaktera za LCD



`mojLCD.createChar(broj,podaci)`

Kreira "custom" karakter za LCD dimenzija do 5x8 piksela. Izgled svakog karaktera specificira se nizom od 8 bajtova, po jedan za svaki red; 5 bitova najmanje težine u svakom bajtu određuju piksele u tom redu

`mojLCD`: promenljiva tipa LiquidCrystal

`broj`: redni broj karaktera (0-7)

`podaci`: podaci o pikselima karaktera

Prvo se ispisuje poruka "Sacekajte" i specijalni karakter "tužić", a nakon 2 sekunde poruka "Spreman sam " i specijalni karakter "smajli". Štampanje specijalnih karaktera na LCD postiže se funkcijom **mojLCD.write(broj)**, gde je broj redni broj karaktera, pri čemu numeracija karaktera počinje od 0. Za štampanje nultog karaktera mora se koristiti **mojLCD.write(byte(0))**.



mojLCD.write(broj)

Prikazuje "custom" karakter na LCD-u

mojLCD: promenljiva tipa LiquidCrystal

broj: redni broj karaktera (0-7); za prikaz nultog karaktera (prvi definisani) mora se koristiti **byte(0)**

U okviru funkcije void setup() inicijalizovan je Timer1 i dodeljen mu interrupt. Iščitavanje sa pina A0 u ISR će biti svake sekunde. U prekidnoj rutini isčitava se vrednost sa razdelnika napona i smešta u celobrojnu promenljivu senzor, zatim se vrši preračunavanje napona, koji će biti tipa float, kao i izračunavanje nepoznate otpornosti potencijometra i poziva se funkcija za osvežavanje LCD-a.

Funkcija za osveženje displeja, briše prethodni sadržaj na ekranu i štampa novi. U gornjem redu se prikazuje "Rp=", izmerena vrednost otpornosti i u nastavku specijalni karakter (Ω). U donjem redu biće ispisana poruka "Merenje " i specijalni karakter "štiklirano".

S obzirom da se očitavanje sa razdelnika napona vrši u ISR-u koji prouzrokuje tajmer, glavna petlja je prazna.

```
#include <LiquidCrystal.h>
#include <TimerOne.h>
#define rxPin A0
volatile float rx;
const unsigned long vremeOcitavanja=1000000;
const int rs=12, en=11, d4=6, d5=5, d6=4, d7=3;
LiquidCrystal mojLCD(rs,en,d4,d5,d6,d7);

byte om[8]={
  B00000,
  B00000,
  B01110,
  B10001,
  B10001,
  B01010,
  B11011,
};

byte tacno[8]={
  B00000,
  B00000,
  B00001,
  B00010,
  B10100,
  B01000,
  B00000,
};

byte srecan[8]={
```

```

    B00000,
    B10001,
    B00000,
    B00000,
    B10001,
    B01110,
    B00000,
};

byte tuzan[8]={
    B00000,
    B10001,
    B00000,
    B00000,
    B01110,
    B10001,
    B00000,
};

void setup() {
    mojLCD.begin(16,2);
    mojLCD.createChar(0,om);
    mojLCD.createChar(1,tacno);
    mojLCD.createChar(2,srecan);
    mojLCD.createChar(3,tuzan);
    mojLCD.setCursor(0,0);
    mojLCD.print("Sacekajte ");
    mojLCD.write(3);
    delay(2000);
    mojLCD.clear();
    mojLCD.print("Spreman sam ");
    mojLCD.write(2);
    delay(2000);
    Timer1.initialize(vremeOcitavanja);
    Timer1.attachInterrupt(senzorISR);
}

void senzorISR(){
    int senzor=analogRead(rxPin);
    float napon=senzor*5.0/1024.0;
    rx=10000.0*((5.0/napon)-1.0);
    osveziLCD();
}

void osveziLCD(){
    mojLCD.clear();
    mojLCD.print("Rp=");
    mojLCD.setCursor(3,0);
    mojLCD.print(rx);
    mojLCD.write(byte(0));
    mojLCD.setCursor(0,1);
    mojLCD.print("Merenje ");
    mojLCD.write(1);
}

void loop() {
}

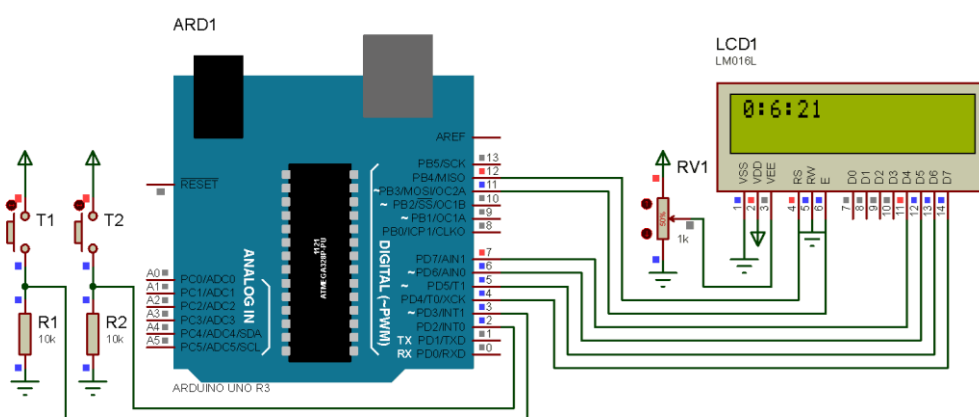
```

PROJEKAT 21.

Simulirati kolo Arduina sa LCD displejem i dva tastera u *pull-down* konfiguraciji u programu Proteus. Napisati program za Arduino koji simulira rad štoperice. Taster T1 koristi se za pokretanje/zaustavljanje štoperice, a taster T2 za resetovanje štoperice. Na LCD-u se prikazuju rezultati u formatu – minuti:sekunde:stotinke. Koristiti Timer1 i tehniku prekida.

Rešenje:

Simulaciona električna šema Arduina sa LCD-om i dva tastera data je na slici 29. Tasteri su vezani na pinove 2 i 3 jer će se koristiti interrupti.



Slika 29. Električna šema Arduina sa LCD-om i dva tastera

Uključene su biblioteke za rad sa LCD-om i Timer1, a zatim su definisani pinovi tastera, konstanta koja određuje kvant tajmera, a samim tim i štoperice – podešena je na 10 ms. Promenljive koje će se koristiti u više funkcija deklarirane su kao volatile: logička promenljiva *tajmerStop*, koja će imati vrednost *true* kada tajmer ne broji i vrednost *false* kada broji; kao i celobrojne promenljive koje će čuvati vrednosti minuta, sekundi i stotinki. Inicijalno su sve postavljene na 0. Izvršena je i deklaracija LCD-a.

U funkciji void setup() inicijalizovan je LCD, deklarirani pinovi tastera kao ulazni i pridružene su im odgovarajuće prekidne rutine, koje će se izvršavati na pritisak tastera. Takođe, inicijalizovan je tajmer i njemu dodeljena ISR će se izvršavati na svakih 10 ms.

Kada se pritisne taster T1, desi se prekid i počinje izvršavanje rutine koja ispituje stanje logičke promenljive *tajmerStop*, ako je *true* – to znači da je tajmer bio zaustavljen i treba ga pokrenuti, što se postiže funkcijom **Timer1.start()**, naravno iza toga treba promeniti status logičke promenljive *tajmerStop* u *false*. Ako je prilikom pritiska tastera T1, vrednost promenljive *tajmerStop* bilo *false*, to znači da je tajmer radio i treba ga zaustaviti, što se postiže funkcijom **Timer1.stop()**, a iza toga promeniti status promenljive *tajmerStop* u *true*.

Kada se pritisne taster T2, desi se prekid i počinje izvršavanje rutine koja resetuje vrednosti promenljivih za minute, sekunde i stotinke i poziva funkciju za osveženje LCD-a, kako bi korisnik video da je štoperica restartovana.



```
Timer1.start();
```

Startuje rad Timera 1

```
Timer1.stop();
```

Zaustavlja rad Timera 1

Kada brojač izbroji 10 ms, dešava se prekid i izvršava se ISR: `stopericaISR()` koja ispituje stanje promenljive `tajmerStop`, ako je *false*, što znači da brojač radi, tj. štoperica broji pa se uvećava vrednost promenljive koja čuva stotinke za 1 i poziva se funkcija za osveženje LCD-a.

Funkcija za osveženje LCD-a, prvo briše prethodne podatke, a zatim redom prikazuje najnovije vrednosti u promenljivama i to u formatu minuti:sekunde:stotinke.

U glavnoj beskonačnoj petlji sve vreme se poziva funkcija za formatiranje vremena. Naime, ova funkcija ispituje da li je vrednost promenljive stotinke veća od 99, ako jeste to znači da je prošla jedna sekunda, zato vrednost promenljive sekunde uvećava za jedan i resetuje vrednost promenljive stotinke na nulu; dalje ispituje da li je vrednost promenljive koja čuva sekunde veća od 59, ako jeste to znači da je prošao minut, uvećava se vrednost promenljive minuti za jedan i resetuje se vrednost promenljive sekunde na nulu; na kraju se ispituje da li je vrednost promenljive koja čuva minute veća od 59, ako jeste to znači da je prošao jedan sat, tada se resetuje vrednost ove promenljive na nulu, s obzirom da je maksimalno vreme brojanja štoperice 1 sat.

```
#include <LiquidCrystal.h>
#include <TimerOne.h>
#define startStopPin 3
#define resetPin 2
const unsigned long kvant=10000;
volatile boolean tajmerStop=true;
volatile unsigned int minuti;
volatile unsigned int sekunde;
volatile unsigned int stotinke;
const int rs=12, en=11, d4=7, d5=6, d6=5, d7=4;
LiquidCrystal mojLCD(rs,en,d4,d5,d6,d7);

void setup() {
    mojLCD.begin(16,2);
    pinMode(startStopPin,INPUT);
    pinMode(resetPin,INPUT);
    attachInterrupt(digitalPinToInterrupt(startStopPin),SSISR,RISING);
    attachInterrupt(digitalPinToInterrupt(resetPin),resetISR,RISING);
    Timer1.initialize(kvant);
    Timer1.attachInterrupt(stopericaISR);
}

void SSISR() {
    if(tajmerStop==true) {
        Timer1.start();
        tajmerStop=false;
    }
    else{
        Timer1.stop();
        tajmerStop=true;
    }
}
```

```

void resetISR() {
    stotinke=0;
    sekunde=0;
    minuti=0;
    osveziLCD();
}

void osveziLCD() {
    mojLCD.clear();
    mojLCD.print(minuti);
    mojLCD.print(':');
    mojLCD.print(sekunde);
    mojLCD.print(':');
    mojLCD.print(stotinke);
}

void stopericaISR() {
    if(tajmerStop==false) {
        stotinke+=1;
    }
    osveziLCD();
}

void loop() {
    formatirajVreme();
}

void formatirajVreme() {
    if(stotinke>99) {
        sekunde+=1;
        stotinke=0;
        if(sekunde>59) {
            minuti+=1;
            sekunde=0;
            if(minuti>59) {
                minuti=0;
            }
        }
    }
}

```

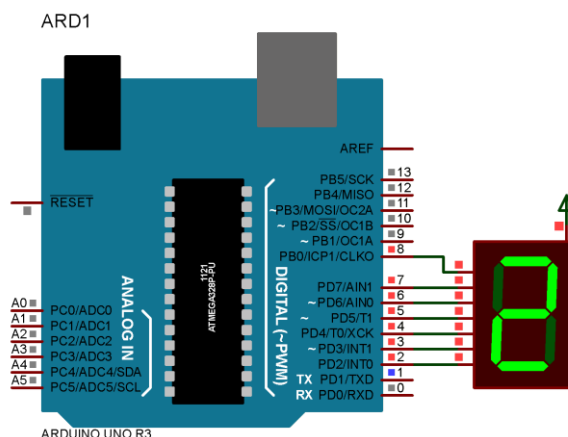
PROJEKAT 22.

Simulirati kolo Arduina sa jednim sedmosegmentnim displejem sa zajedničkom anodom u programu Proteus. Napisati program za Arduino koji prikazuje brojeve od 0 do 9 na sedmosegmentnom displeju, pri čemu se cifra smenjuje svake sekunde. Kada prikaže poslednju cifru 9, isključuje se na 2 s, a zatim ponovo broji od nule.

Rešenje:

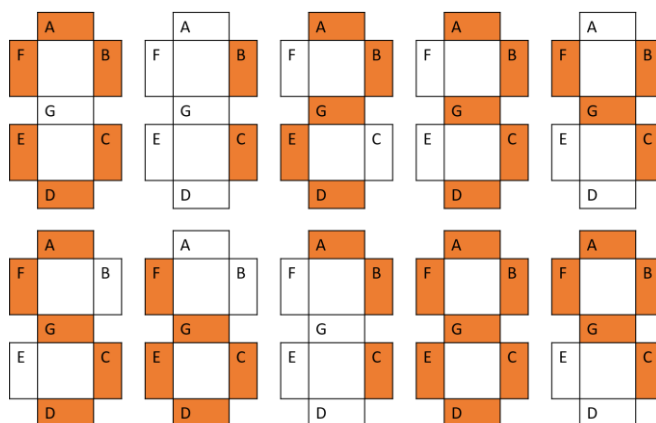
Simulaciona električna šema za kolo Arduina sa sedmosegmentnim (7seg) displejem data je na slici 30. Upotrebljen je 7seg displej sa zajedničkom anodom, što znači da će se segmenti uključivati naponom logičke nule. Segmenti su vezani na digitalne pinove Arduina i to: A-G na 8-2, respektivno. Za decimalnu tačku (DP) rezervisan je pin 1, ako je iskorišćen simulacioni model 7seg sa DP-om. U praksi se često koriste i 7seg displeji sa većim brojem (2 ili 4, najčešće) 7seg displeja u jednom integrisanom kućišu i za njihovo upravljanje koristi se

tehnika vremenskog multipleksiranja, a mogu biti potrebna i dodatna kola, kao što su pomerački registri.



Slika 30. Električna šema Arduina i 7seg displeja sa zajedničkom anodom

Na početku programa je deklarisan niz konstanti tipa bajt, koje čuvaju informaciju o stanju svakog segmenta (A-G i DP) za svaku od cifara 0-9 (slika 31). Zatim je deklarisan niz celobrojnih vrednosti koji čuva oznake pinova na koje su vezani segmenti DP, G-A.



Slika 31. Kreiranje cifara na 7seg displeju

U funkciji void setup() deklarisan su svi pinovi segmenata kao izlazni korišćenjem for petlje. U glavnom programu se poziva funkcija za prikaz broja pri svakom prolazu kroz for petlju koja ide od 0 do 10, dok je vreme između prikaza određeno funkcijom delay().

Funkcija za prikaz broja preuzima vrednost brojača iz for petlje u glavnom programu, smešta je u celobrojnu promenljivu br, koja sadrži broj koji treba ispisati. Deklarisana je logička promenljiva stanje, dakle može imati stanje "1" i "0". Kroz for petlju kreće provera segmenata, promenljiva segment kreće od 1 i broji do 7. Ako je broj koji treba ispisati manji od nule, odnosno veći od 9, promenljiva stanje postavlja se na nulu. Ako je cifra koju treba ispisati u dozvoljenom opsegu kreće ispitivanje bit-po-bit korišćenjem funkcije **bitRead(brojevi[br],segment)**. Ako dati segment treba da bude uključen, funkcija vraća "1", što se upisuje u stanje. S obzirom da se koristi 7seg sa zajedničkom anodom, korišćenjem operatora inverzije, menja se stanje u suprotno od postavljenog i korišćenjem funkcije digitalWrite() se odgovarajući segment postavlja u odgovarajuće stanje.

Na primer, treba ispisati broj 2, promenljiva br=2. Pošto je broj iz opsega kreće izvršavanje funkcije bitRead() za segment=1, biće stanje=bitRead(brojevi(2),1). Gleda se vrednost brojevi(2)=B11011010 i to segment=1, odnosno bit najveće težine s obzirom da je ovan niz definisan za segmente A-G. Pošto je taj bit=1, stanje postaje 1, pa se invertuje u 0, a onda se prosleđuje digitalWrite(segmentPin[1],0) – što će uključiti segment A. Segment se sada uvećava, segment=2. Gleda se brojevi(2)=B11011010 i to segment 2 (=B), koji je 1, pa stanje postaje 1, zatim se invertuje i sa digitalWrite(segmentPin[2],0) uključuje se segment B. Dalje, segment=3=C, gleda se brojevi(2)=B11011010 i čita bit na trećoj poziciji, koji je 0, pa stanje postaje nula, zatim se invertuje u 1 i sa digitalWrite(segmentPin[3],1) će segment C biti isključen, itd.

```
const byte brojevi[10]={
  B11111100, // 0
  B01100000, // 1
  B11011010, // 2
  B11110010, // 3
  B01100110, // 4
  B10110110, // 5
  B00111110, // 6
  B11100000, // 7
  B11111110, // 8
  B11100110, // 9
};
const int segmentPin[8]={1,2,3,4,5,6,7,8};

void setup() {
  for(int i=0;i<8;i++){
    pinMode(segmentPin[i],OUTPUT);
  }
}

void loop() {
  for(int i=0; i<=10;i++){
    prikaziBroj(i);
    delay(1000);
  }
  delay(2000);
}

void prikaziBroj(int br){
  boolean stanje;
  for(int segment=1;segment<8;segment++){
    if(br<0 || br>9){
      stanje=0;
    }
    else{
      stanje=bitRead(brojevi[br],segment);
    }
    stanje=!stanje;
    digitalWrite(segmentPin[segment],stanje);
  }
}
```

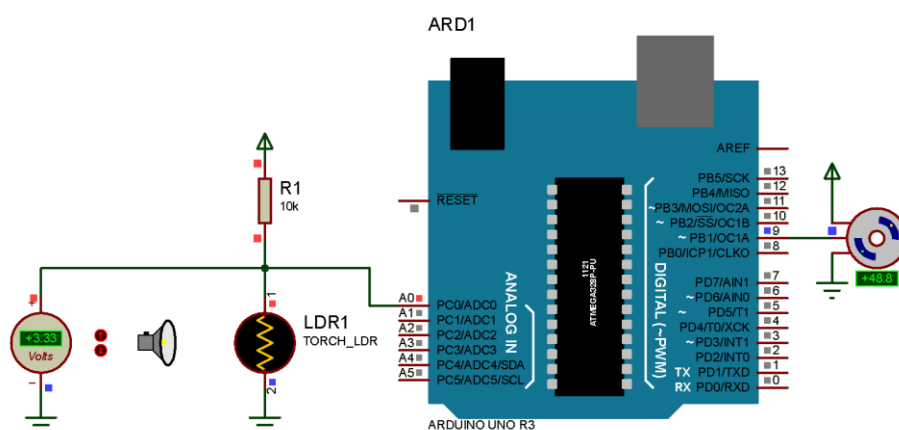
Servo Motori

PROJEKAT 23.

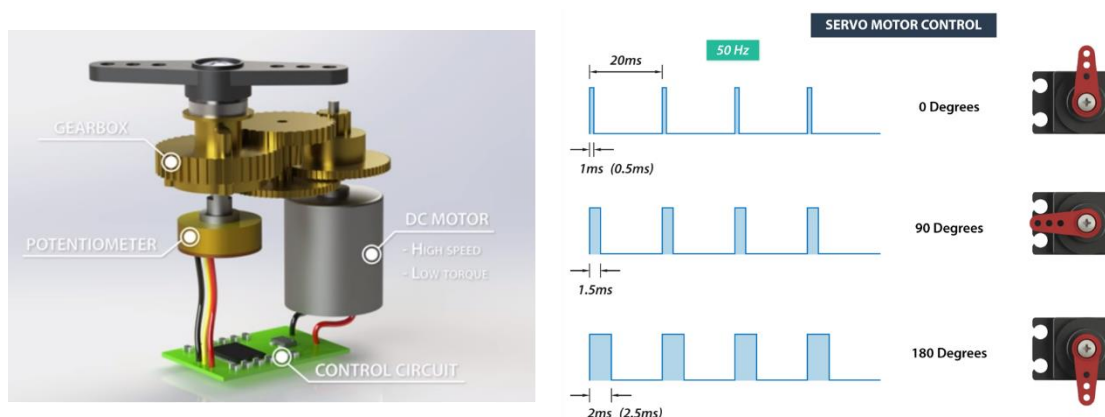
Simulirati kolo Arduina sa fotootpornikom u razdelniku napona i Servo motorom. Napisati program za Arduino koji sa promenom osvetljenosti menja položaj (ugao) servo motora (simulacija pametne roletne).

Rešenje:

Simulaciona šema kola Arduina sa fotootpornikom u razdelniku napona i servo motorom data je na slici 32. Iskorišćen je animirani model fotootpornika (torch_ldr), a paralelno sa fotootpornikom povezan je DC volmetar radi praćenja promene napona na fotootporniku sa promenom osvetljenosti. Na slici 33 ilustrovana je unutrašnjost RC (hobby) servo motora i prikazani kontrolni signali servo motora. Servo motor kontroliše se PWM tehnikom. Na primer, ako je potrebno da motor zauzme ugao 0°, na svakih 20 ms potrebno je poslati jedan naponski impuls u trajanju od 1 ms. Standardni servo motori omogućavaju pozicioniranje od 0-180°, ali postoje i servo motori sa rotacijom od 360°.



Slika 32. Električna šema Arduina sa fotootpornikom u razdelniku napona i servo motorom



Slika 33. Ilustracija unutrašnjosti i oblik kontrolnih PWM signala Servo motora

Za rad sa servo motorima potrebno je uključiti biblioteku **<Servo.h>**, koja omogućava korišćenje funkcija za upravljanje. U programu su još definisani redni brojevi pinova na koje

je vezan senzor, tj. fotootpornik u razdelniku napona i servo motor. Deklarisane su promenljive koje će čuvati očitane vrednosti sa senzora i ugao servo motora. Kreirana je promenljiva `mojServo`, tipa `Servo` preko koje će se pozivati funkcije za upravljanje servo motorom.

U okviru funkcije vodi `setup()` deklarisan je mod senzorskog pina – ulazni, dok je funkcijom `mojServo.attach(pin)` definisan pin na koji je vezan servo motor, a funkcijom `mojServo.write(ugao)`, upisana je inicijalna vrednost ugla koga servo motor treba da zauzme (0°).



```
#include <Servo.h>
```

Uključivanje biblioteke za rad sa servo motorima

```
mojServo.attach(pin)
```

Vezuje `Servo` promenljivu za zadati pin; definiše pin na koji je vezan servo motor

mojServo: promenljiva tipa `Servo`

pin: broj pina na koji je povezan servo motor

```
mojServo.write(ugao)
```

Podešava ugao servo motora za zadatu vrednost

mojServo: promenljiva tipa `Servo`

ugao: vrednost ugla koji treba postaviti (0-180)

U glavnoj petlji programa korišćenjem funkcije `analogRead()` vrši se iščitavanje senzora, a potom korišćenjem funkcije `map()` skaliranje očitane vrednosti, koja inače može da bude u opesu od 0-1023 (zbog desetobitnog ADC-a) na vrednost 0-180 (ugao servo motora). Rezultat izvršenja ove funkcije upisuje se u odgovarajuću promenljivu, a zatim se ta vrednost korišćenjem `mojServo.write()` prosleđuje motoru.

```
#include <Servo.h>
#define senzorPin A0
#define servoPin 9
int senzor = 0;
int servoUgao = 90;
Servo mojServo;

void setup() {
    pinMode(senzorPin, INPUT);
    mojServo.attach(servoPin);
    mojServo.write(servoUgao);
}

void loop() {
    senzor=analogRead(senzorPin);
    servoUgao=map(senzor,0,1023,0,180);
    mojServo.write(servoUgao);
    delay(100);
}
```

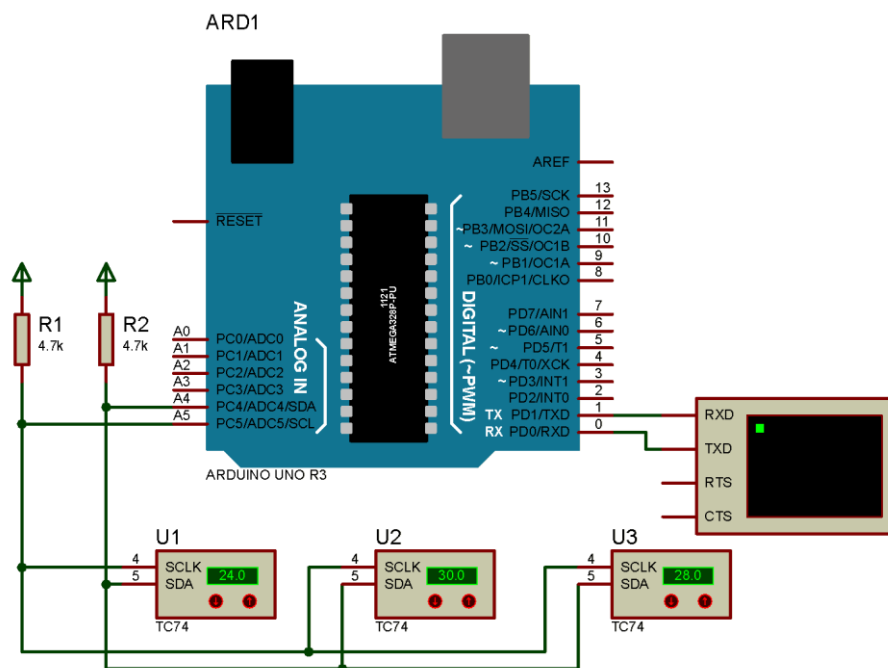
Serijska I2C komunikacija

PROJEKAT 24.

Simulirati kolo Arduino sa tri temperaturna senzora TC74 koji koriste serijsku I2C komunikaciju i virtuelnim terminalom. Napisati program za Arduino koji iščitava temperaturu sa svakog senzora i prikazuje je preko virtuelnog terminala.

Rešenje:

Električna šema Arduino sa I2C senzorima temperature TC74 prikazana je na slici 34. **I2C** (Integrated-Integrated Circuit) je **sinhroni serijski half-duplex** princip komunikacije realizovan sa dve žice. Koristi se za povezivanje sporijih periferija na mikrokontrolere i funkcioniše po principu **master-slave komunikacije**. I2C magistrala dozvoljava postojanje više mastera i više slave-ova na jednoj magistrali. Master je glavni jer diktira učestanost signala takta. U ovom projektu Arduino je master uređaj, a senzori su slave uređaji. Za prenos podataka koriste se dve žice: **SCL (Serial Clock)** koja prenosi taktni signal i **SDA (Serial Data)** koja prenosi podatke. **Obavezno je korišćenje pull-up otpornika** radi definisanosti stanja na magistrali. Vrednost otpornika zavisi od željene brzine prenosa i kapacitivnosti linije, tipično se koristi 4.7 kΩ. **Arduino ima mogućnost komunikacije preko I2C: pin A5 je SCL, dok je A4 rezervisan za SDA.**



Slika 34. Električna šema Arduino sa I2C temperaturnim senzorima TC74 i virtuelnim terminalom

Biblioteka koja omogućava I2C komunikaciju je **Wire.h**. Svako integrisano kolo koje komunicira preko I2C protokola ima jedinstvenu adresu preko koje se inicira komunikacija od mastera prema slave uređaju. **Adresa** određenog kola data je u njegovom datasheetu. Na slici 35 dat je deo iz datasheeta temperaturnog senzora TC74 gde su prikazane adrese senzora za različite podserije. Na primer, za TC74A0 adresa u binarnom formatu je 1001000,

što dekadno odgovara 72. U Proteusu je moguće podesiti ove adrese, one su u ovom projektu postavljene redom na vrednosti 72, 73, 74. Na početku programa u odgovarajućim promenljivama upisane su vrednosti izabranih adresa.

SOT-23 (V)	Address	Code	SOT-23 (V)	Address	Code
TC74A0-3.3VCT	1001 000	V0	TC74A0-5.0VCT	1001 000	U0
TC74A1-3.3VCT	1001 001	V1	TC74A1-5.0VCT	1001 001	U1
TC74A2-3.3VCT	1001 010	V2	TC74A2-5.0VCT	1001 010	U2
TC74A3-3.3VCT	1001 011	V3	TC74A3-5.0VCT	1001 011	U3
TC74A4-3.3VCT	1001 100	V4	TC74A4-5.0VCT	1001 100	U4
TC74A5-3.3VCT	1001 101*	V5	TC74A5-5.0VCT	1001 101*	U5
TC74A6-3.3VCT	1001 110	V6	TC74A6-5.0VCT	1001 110	U6
TC74A7-3.3VCT	1001 111	V7	TC74A7-5.0VCT	1001 111	U7

Note: * Default Address

Slika 35. Adrese I2C temperaturnog senzora TC74 (preuzeto iz [datasheeta](#))

U okviru funkcije `setup()` deklarirana je serijska UART komunikacija, ali i I2C korišćenjem **Wire.begin()**. Ovom funkcijom se inicira Wire biblioteka koja omogućava izlaz na I2C magistralu. U glavnom programu se poziva funkcija koja će preko I2C magistrale iščitati temperaturu sa odgovarajućeg senzora, koja se upisuje u promenljivu i štampa na virtuelni terminal korišćenjem `Serial.print()`.

Funkcija `int procitaj(adresa)`, preuzima iz glavnog programa adresu senzora sa koga treba prikupiti informacije, a deklarise se kao `int` jer vraća celobrojnu vrednost – vrednost očitane temperature. U skladu sa I2C protokolom, funkcijom **Wire.beginTransmission(adresa)** počinje komunikacija sa I2C slave uređajem prozivanjem po adresi. Naime, slave uređaj kontinuirano osluškuje SCL i SDA linije čekajući START bit: da se desi prelaz sa HIGH na LOW na SDA kada je SCL na HIGH. Slave ne odgovara odmah master-u, već master nakon START bita šalje adresu. Samo onaj slave uređaj čija je adresa poslata se odaziva (šalje tzv. ACK – acknowledge bit) i čeka narednu komandu. Master šalje izabranom slave-u komandu. U ovom projektu, Arduino će tražiti od izabranog senzora da pošalje saržaj svog registra, tj. temperaturu. Iz [datasheet-a](#) senzora nalazimo informaciju da kada senzor primi komandni bajt 0 on šalje temperaturu, zato korišćenjem funkcije **Wire.write(komanda)** pripremamo slanje nule. Iza toga, master šalje tu komandu i prekida komunikaciju ka senzoru, sa ciljem da na SDA liniji prikupi podatke koje će izabrani slave poslati. Slanje komande i prekid komunikacije realizuje se korišćenjem funkcije **Wire.endTransmission()**. Ova funkcija završiće komunikaciju sa slave-om slanjem STOP bita (prelaz sa LOW na HIGH SDA linije, dok je SCL u stanju HIGH). Nakon toga, slave počinje da šalje podatke, zato master uređaj mora da se pripremi za prihvatanje tih podataka. Korišćenjem funkcije **Wire.requestFrom(adresa,broj_bajtova)** master očekuje da primi određeni broj bajtova sa zadate adrese. U ovom projektu, senzor šalje vrednost temperature u jednom bajtu. Sa **Wire.available()** čekaju se podaci koji stižu od slave uređaja preko I2C magistrale, ova funkcija vraća broj dostupnih bajtova koje treba iščitati. U ovom projektu, dok ova funkcija vraća 0, program se vrti u praznoj while petlji, dok podaci ne stignu. Kada podaci stignu, izlazi se iz petlje i u odgovarajuću promenljivu se korišćenjem funkcije **Wire.read()** upisuju pristigli podaci. Prihvatna promenljiva je tipa `int`, tako da će u promenljivu rez biti upisana vrednost temperature u decimalnom brojnom sistemu. Sa **return rez**, u glavni program vraća se rezultat izvršenja ove funkcije, očitana temperatura.



```
#include <Wire.h>
```

Uključivanje biblioteke koja omogućava I2C komunikaciju

```
Wire.begin()
```

Inicira Wire biblioteku i omogućava izlaz na I2C magistralu; poziva se jednom; opciono se može proslediti 7-bitna adresa; ako se ne navede adresa Arduino izlazi na magistralu kao master uređaj.

```
Wire.beginTransmission(adresa)
```

Počinje komunikaciju sa I2C slave uređajem sa datom adresom.

adresa: 7-bitna adresa uređaja sa kojim se uspostavlja komunikacija

```
Wire.write(komanda)
```

Priprema podatke za prenos od mastera ka slave-u, ili upisuje podatke iz slave u odgovoru na zahtev mastera; funkcija vraća broj upisanih bajtova

komanda: vrednost koja se šalje kao jedan bajt, ili string – kao serija bajtova, ili kao niz podataka koji se šalju kao bajtovi

```
Wire.endTransmission()
```

Završava komunikaciju sa slave uređajem koja je pokrenuta funkcijom Wire.beginTransmission() i šalje bajtove koji su pripremljeni funkcijom Wire.write(); ova funkcija vraća jedan bajt koji pokazuje status transmisije, 0 – uspešan prenos, druge vrednosti ukazuju na odgovarajuće greške u komunikaciji

```
Wire.requestFrom(adresa,broj_bajtova)
```

Koristi se od strane mastera da zatraži određen broj bajtova od slave uređaja; ukoliko je vrednost ove funkcije *true* (defaultna vrednost) nakon slanja zahteva, šalje STOP biti i napušta I2C magistralu; u suprotnom šalje restart poruku posle zahteva i ne napušta magistralu, što štiti od toga da drugi master zatraži nešto od slave-a

```
Wire.available()
```

Vraća broj bajtova koji su dostupni da budu prihvaćeni sa Wire.read(); poziva se posle Wire.requestFrom()

```
Wire.read()
```

Čita bajt koji je stigao iz slave uređaja u master posle poziva funkcije Wire.requestFrom() ili je prenet iz master-a u slave uređaj

```
#include <Wire.h>
int adresa1=72;
int adresa2=73;
int adresa3=74;
void setup() {
  Serial.begin(9600);
  Wire.begin();
}

void loop() {
  int temp1=procitaj(adresa1);
  int temp2=procitaj(adresa2);
  int temp3=procitaj(adresa3);
  Serial.print("Ts1=");
  Serial.println(temp1);
  Serial.print("Ts2=");
  Serial.println(temp2);
  Serial.print("Ts3=");
  Serial.println(temp3);
  delay(500);
}
```

```

int procitaj(int adresa){
    Wire.beginTransmission(adresa);
    Wire.write(0);
    Wire.endTransmission();
    Wire.requestFrom(adresa,1);
    while(Wire.available()==0);
    int rez=Wire.read();
    return rez;
}

```

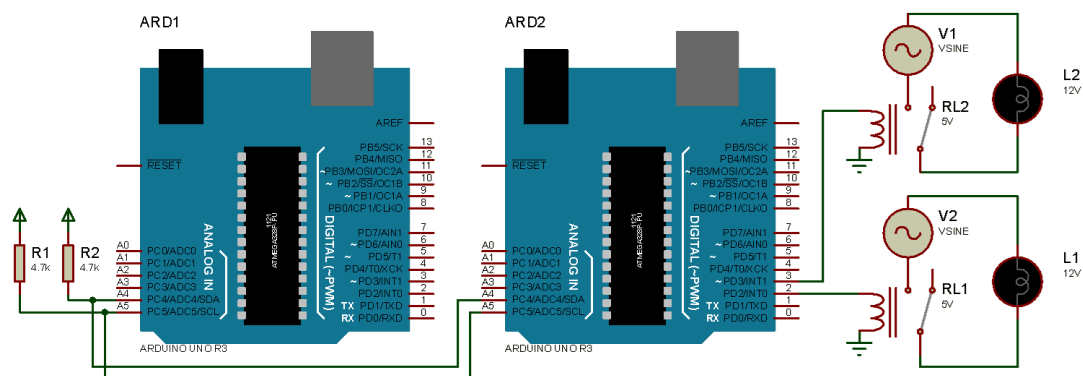
PROJEKAT 25.

Simulirati kolo sa dva Arduina u programu Proteus. Prvi Arduino ima ulogu master uređaja, a drugi Arduino ulogu slave uređaja na koji su preko dva releja povezane sijalice. Napisati programe za master i slave Arduino:

- Master Arduino šalje na adresu slave Arduina komandu za uključivanje odgovarajućeg releja, i to relej 1 treba da bude uključen 0.5 s, a relej 2 jednu sekundu, a onda se naizmenično ponavlja uključivanje/isključivanje releja.
- Slave Arduino osluškuje komande iz master Arduina, prihvata pristiglu informaciju i na osnovu nje donosi odluku koji relej treba uključiti, a koji isključiti.

Rešenje:

Električna šema sistema sa dva Arduina koji komuniciraju preko I2C magistrale prikazan je na slici 36. Iskorišćeni su *pull-up* otpornici od 4.7 kΩ na I2C magistrali kako bi u svakom trenutku obezbedili definisano stanje. ARD1 radi kao master, a ARD2 kao slave uređaj. Na pinove 2 i 3 slave Arduina vezani su releji koji se uključuju naponom od 5 V DC. U izlaznom stepenu svakog releja vezana je sijalica na generator naizmeničnog sinusnog signala.



Slika 36. Električna šema sistema dva Arduina koji komuniciraju preko I2C magistrale

Program master Arduina počinje uključivanjem biblioteke za rad sa I2C magistralom Wire.h, inicijalizuje se promenljiva čija će se vrednost slati slave uređaju. U okviru setup() funkcije inicijalizuje se I2C komunikacija korišćenjem Wire.begin(). U beskonačnoj petlji loop() se otpočinje I2C komunikacija sa slave uređajem sa adresom 9, korišćenjem Wire.beginTransmission(). Zatim se korišćenjem Wire.write(x), priprema podatak za slanje, i šalje se nakon čega se završava komunikacija, tj. master Arduino se povlači sa I2C magistrale

što se postiže sa `Wire.endTransmission()`. Iza toga vrši se inkrement podatka koji će se slati u narednoj transmisiji. Ako je taj podatak veći od 3 njegova vrednost se setuje na 1, i čeka se do narednog slanja podatka određeno vreme ($500 \cdot x$), tj. 500 ms ili 1 s.

```
#include <Wire.h>
int x=1;
void setup() {
    Wire.begin();
}

void loop() {
    Wire.beginTransmission(9);
    Wire.write(x);
    Wire.endTransmission();
    x++;
    if(x>3){
        x=1;
    }
    delay(x*500);
}
```

Program slave Arduina počinje uključivanjem biblioteke `Wire.h` koja omogućava rad na I2C magistrali, zatim se definišu pinovi na koje su povezani releji i deklariše promenljiva koja će čuvati pristigle podatke od master uređaja. U funkciji `setup()` podešavaju se pinovi za kontrolu releja kao izlazni i inicijalizuje I2C komunikacija funkcijom `Wire.begin(9)`. Slave Arduino izlazi na I2C magistralu sa adresom 9. Nakon toga se zadaje funkcija koja će biti pozvana kada slave uređaj primi komandu od mastera, što se postiže sa **`Wire.onReceive(funkcija)`**. Dakle, svaki put kada ARD2 primi podatke iz ARD1 preko I2C biće pozvana funkcija `void procitaj(int podaci)`. Ova funkcija prihvata u promenljivu `podaci` broj bajtova koje treba da pročita od mastera. U okviru funkcije `procitaj()` u promenljivu `y` smeštaju se pristigli podaci korišćenjem `Wire.read()`. U glavnoj petlji programa slave Arduina ako je stigao podatak `y=x=1`, uključuje se relej 1, a isključuje relej 2, odnosno ako je stiglo `y=x=2`, uključuje se relej 2, a isključuje relej 1. Ovde treba napomenuti da o dužini uključenosti releja ne vodi računa slave, nego master uređaj; slave samo izvršava naredbe mastera.



`Wire.onReceive(funkcija)`

Zadavanje funkcije koja će biti pozvana kada slave uređaj primi podatke iz master uređaja

funkcija: funkcija koja će biti pozvana kada slave uređaj primi podatke; ova funkcija treba da preuzme u jednu int promenljivu parametar, tj. broj bajtova koje treba pročitati od master uređaja

```
#include <Wire.h>
#define r1 2
#define r2 3
int y=1;
void setup() {
    pinMode(r1, OUTPUT);
    pinMode(r2, OUTPUT);
    Wire.begin(9);
    Wire.onReceive(procitaj);
}
```

```
}

void procitaj(int podaci){
    y=Wire.read();
}

void loop() {
    if(y==1){
        digitalWrite(r1,HIGH);
        digitalWrite(r2,LOW);
    }
    else{
        digitalWrite(r1,LOW);
        digitalWrite(r1,HIGH);
    }
}
```

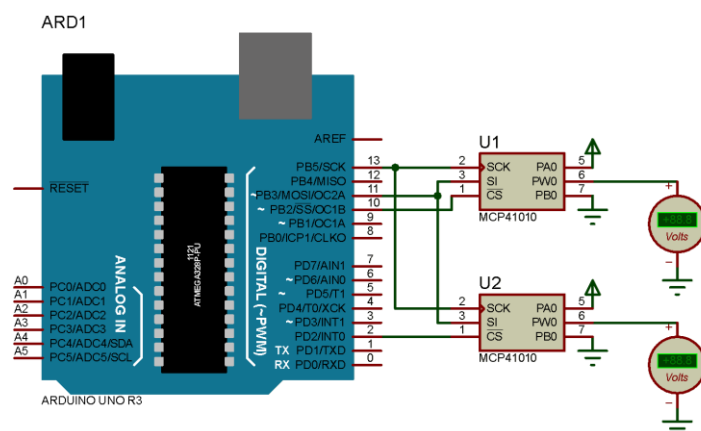
Serijska SPI komunikacija

PROJEKAT 26.

Simulirati kolo Arduina sa dva digitalna potencimetra MCP41010 koji koriste SPI protokol u programu Proteus. Napisati program za Arduino koji uspostavlja komunikaciju sa digitalnim potencimetrima i menja im otpornost tokom vremena, i to jednom od najveće ka najnižoj vrednosti, a drugom obrnuto.

Rešenje:

Simulaciona šema Arduina sa digitalnim potencimetrima MCP41010 koji koriste SPI protokol prikazana je na slici 37. **SPI (Serial Peripheral Interface)** je **serijski sinhroni full-duplex** princip komunikacije realizovan sa četiri žice. Koristi se za povezivanje uređaja koji nisu mnogo udaljeni i funkcioniše po principu **master-slave** komunikacije. Master uređaj upravlja prenosom na magistrali, generiše takt i započinje prenos. Slave uređaj prima naredbe od master uređaja i odgovara mu u skladu sa primljenim naredbama. Na SPI magistrali može biti samo jedan master, a više slave uređaja. U ovom projektu Arduino je master uređaj, a digitalnim potencimetrima su slave uređaji. Za prenos podataka koriste se četiri žice: **SCLK (Serial Clock)** koja prenosi taktni signal, **MOSI (Master Output Slave Input)** koja prenosi podatke od master-a ka slave-u, **MISO (Master Input Slave Output)** koja prenosi podatke od slave-a ka master-u i **SS (Slave Select)** koja služi za selekciju slave-a u sistemima sa više slave uređaja. **Arduino ima mogućnost komunikacije preko SPI: pin 13 je SCLK, pin 12 je MISO, 11 je MOSI, dok jedan od SS pinova može biti pin 10.** U ovom projektu koriste se dva slave uređaja, taktni signal im se dovodi sa pina 13 na SCK pinove, komande im master šalje preko pina 11 na pin SI (Slave Input). Slave uređaji ne vraćaju nikakvu informaciju masteru, tako da MISO linija ostaje nepovezana. Arduino proziva na SPI magistralu slave uređaje preko SS pina, i to prvi digitalni potencijometar preko pina 10, a drugi digitalni potencijometar preko pina 2. Povezivanjem krajnjih pinova potencimetra PA0 i PB0 na napajanje i masu, respektivno, formira se razdelnik napona. Na srednji izvod potencimetra PW0 vezan je DC voltmetar koji će meriti napon u razdelniku sa ciljem vizuelne ilustracije rada sistema.



Slika 37. Električna šema Arduina sa dva digitalna potencimetra koji koriste SPI protokol

Program počinje uključivanjem biblioteke koja omogućava SPI komunikaciju **SPI.h** i deklaracijom Slave Select pinova digitalnih potencijometara. U okviru funkcije `setup()`, ovi pinovi se podešavaju kao izlazni i funkcijom **SPI.begin()** se inicijalizuje SPI magistrala, što znači da se pinovi SCK, MOSI podešavaju kao izlazni, pinovi SCK i MOSI se postavljaju u stanje logičke nule, a SS u stanje logičke jedinice. Nakon toga se funkcijom **SPI.setBitOrder()** definiše da li će se na magistrali prvo prenositi bitovi najveće težine (MSBFIRST) ili bitovi najmanje težine (LSBFIRST). Treba napomenuti da većina integrisanih kola koja koriste SPI protokol koriste MSB redosled. Pored redosleda slanja bitova, treba voditi računa o maksimalnoj brzini na SPI magistrali, kao i o **SPI modu** koji se odnosi na to da li se podaci čitaju/šalju na rastućoj ili opadajućoj ivici taktnog signala (**clock phase**) i da li je CLK u stanju mirovanja kada je taj signal LOW ili HIGH (**clock polarity**). Postoje **4** moguća **moda**, dobijena kombinacijom clock phase i clock polarity vrednosti. Ova podešavanja moguće je postići sa **SPISettings podesavanja(2000000,MSBFIRST,SPI_MODE1)**, što bi konkretno značilo da je brzina SPI čipa 2MHz, pa bi Arduino koristio tu ili manju brzinu, sa ciljem uspešne komunikacije; prvo bi se slali bitovi najveće težine i bio bi podešen Mode 1 – koji znači da je clock polarity = 0, clock phase = 1 – podaci se primaju/šalju na opadajući ivicu taktnog signala. Nažalost, SPI standard se godinama izgubio, tako da se na različitim integrisanim kolima različito implementira, što znači da posebnu pažnju treba obratiti na informacije iz datasheet-a izabranog SPI čipa pre i tokom pisanja koda.

U glavnom programu izvršava se for petlja u okviru koje se poziva funkcija koja preko SPI magistrale šalje vrednost na koju je potrebno postaviti svaki od digitalnih potencijometara. Argumenti funkcije `posalji()` su SlaveSelect pin digitalnog potencijometra na koji se šalju podaci i vrednost koja se šalje. Izabrani digitalni potencijometar ima ukupnu otpornost 10 kΩ, a u zavisnosti od 8-bitne vrednosti (0-255) koju dobije preko SPI magistrale menja se položaj “klizača”, tj. raspodeljuje se ukupna otpornost između izlansih izvoda potencijometra.

Funkcija `posalji()` pre svega na odgovarajući SS pin, postavlja logičku nulu i time započinje komunikacija. Zatim korišćenjem funkcije **SPI.transfer()** iz mastera u slave se šalje komandni bajt, a potom i bajt podataka, a komunikacija sa slave-om se završava podizanjem SS pina na nivo logičke jedinice. Format podataka koje treba poslati slave uređaju da bi on izvršio određenu akciju dati su u datasheetu te komponente. U ovom projektu, iz [datasheeta](#) digitalnog potencijometra dobijen je format komandnog bajta (slika 38), kao i format bajta podataka. Analizom komandnog bajta može se videti da prva dva bita najveće težine nisu definisana, zatim sledeća dva određuju koji tip komande će se izvršiti (upisivanje podataka u potencijometar, isključivanje, ili ništa). Potom, naredna dva bita su takođe nedefinisana, a poslednja dva određuju koji od potencijometra u jednom čipu će biti selektovan. Naime, postoje serije ovog tipa digitalnog potencijometra koje integrišu dva potencijometra. U ovom projektu, komandni bajt je **B00010001**, što znači da će biti izabrana komanda za upisivanje podataka u potencijometar, kao i da se komanda izvršava na potencijometru 0. Čip korišćen u ovom projektu ima samo jedan potencijometar. Bajt podataka je decimalna vrednost od 0 do 255 koja će odrediti otpornost potencijometra od “klizača” do mase. Na slici 39 data je normalizovana vrednost otpornosti digitalnog potencijometra od “klizača” do mase u zavisnosti od kodirane-poslate vrednosti.



```
#include <SPI.h>
```

Uključivanje biblioteke koja omogućava rad SPI protokola

```
SPI.begin()
```

Inicijalizuje SPI magistralu podešavajući SCK, MOSI i SS pinove kao izlazne i postavlja SCK i MOSI u LOW stanje, a SS u HIGH stanje

```
SPI.setBitOrder(REDOSLED)
```

Podešava redosled slanja/primanja bitova na SPI magistrali

REDOSLED: MSBFIRST (prvi bitovi najveće težine), LSBFIRST (prvi bitovi najmanje težine)

```
SPISettings podesavanja(maksimalna_brzina, REDOSLED, mod)
```

Podešavanja SPI protokola

podesavanja: naziv skupa podešavanja

maksimalna_brzina: maksimalna brzina komunikacije, do 20MHz (20000000)

REDOSLED: MSBFIRST, LSBFIRST

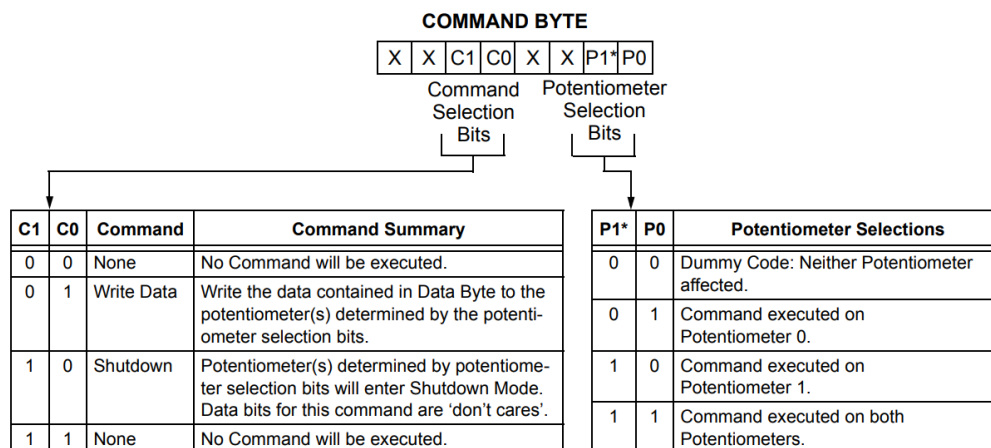
mod: SPI_MODE0, SPI_MODE1, SPI_MODE2, SPI_MODE3

```
SPI.transfer(vrednost), primljeno=SPI.transfer(vrednost)
```

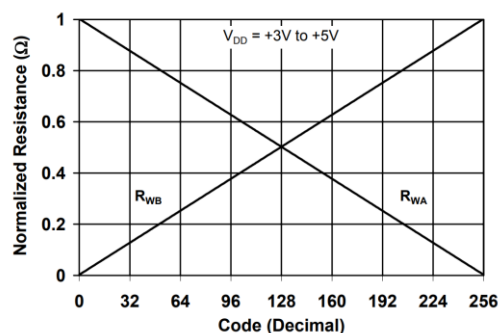
SPI prenos podataka zasnovan je na istovremenom slanju i primanju podataka; primljeni podaci prihvataju se u određenoj promenljivoj, ili u baferskom registru mikrokontrolera; moguće je poslati i dva bajta na magistralu tada se koristi SPI.transfer16()

vrednost: bajt koji se šalje na SPI magistralu

primljeno: promenljiva koja prihvata podatke pristigle preko SPI magistrale



Slika 38. Forma komandnog bajta digitalnog potencijometra (preuzeto iz [datasheeta](#))



Slika 39. Zavisnost normalizovane otpornosti digitalnog potencijometra od kodirane vrednosti (preuzeto iz [datasheeta](#))

```

#include <SPI.h>
int ss1=10;
int ss2=2;

void setup() {
  pinMode(ss1, OUTPUT);
  pinMode(ss2, OUTPUT);
  SPI.begin();
  SPI.setBitOrder(MSBFIRST);
}

void loop() {
  for(int i=0;i<256;i++){
    posalji(ss1,i);
    posalji(ss2,255-i);
    delay(50);
  }
}

void posalji(int cip, int vrednost){
  digitalWrite(cip, LOW);
  SPI.transfer(B00010001);
  SPI.transfer(vrednost);
  digitalWrite(cip, HIGH);
}

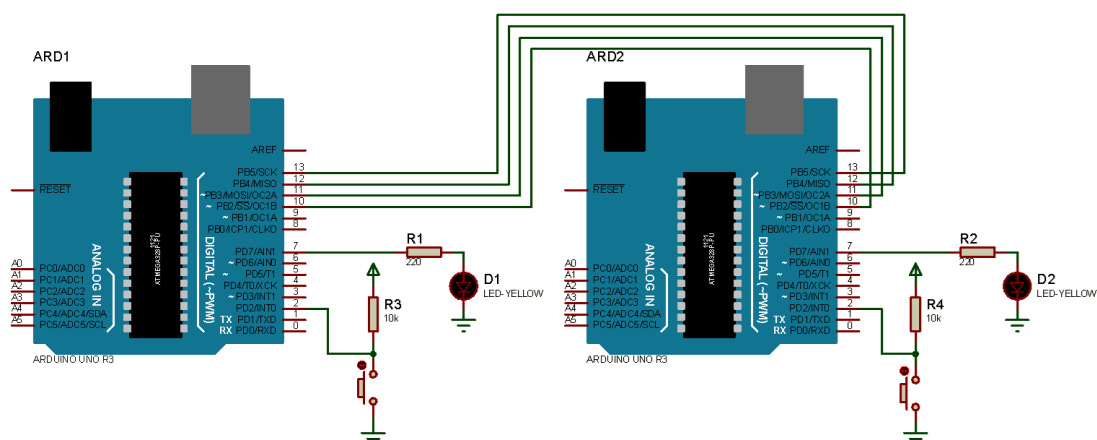
```

PROJEKAT 27.

Simulirati kolo sa dva Arduina koji komuniciraju preko SPI magistrale, a imaju povezane taster u *pull-up* konfiguraciji i LED diodu u programu Proteus. Napisati programe za master i slave Arduino, ako korisnik pritisne taster vezan za master Arduino svetli LED na slave Arduino okruženju i obrnuto.

Rešenje:

Na slici 40 data je električna šema dva Arduina koji komuniciraju preko SPI magistrale sa povezanim tasterom u *pull-up* konfiguraciji i LED-om. ARD1 biće konfigurisan da radi kao master uređaj, a ARD2 kao slave uređaj.



Slika 40. Električna šema dva Arduina (master+slave) koji komuniciraju preko SPI magistrale sa povezanim tasterima u *pull-up* konfiguraciji i LED-ovima

Program master Arduina ARD1 počinje uključivanjem biblioteke SPI.h koja omogućava rad sa SPI protokolom, definišu se pinovi gde su povezani taster i LED, ali i Slave Select pin. Deklarišu se promenljive koje će čuvati stanje tastera i podatke koji će biti poslani putem SPI magistrale, kao i promenljive tipa byte koje će čuvati podatke koji se šalju, odnosno podatke koji su primljeni preko SPI magistrale od slave Arduina. U okviru funkcije setup() definišu se pinovi i inicijalizuje SPI komunikacija. Funkcijom **SPI.setClockDivider(DIV4)** postavlja se delilac taktnog signala u odnosu na sistemski taktni signal u cilju regulisanja brzine prenosa podataka na SPI magistrali. Podrazumevana vrednost je SPI_CLOCK_DIV4, što znači da se SPI brzina podešava na ¼ frekvencije sistemskog takta (4 MHz, ako je sistemski takt 16 MHz). U ovom projektu, vrednost delioca postavljena je na SPI_CLOCK_DIV8. Funkcijom digitalWrite() Slave Select pin postavljen je na HIGH, što znači da još nije počela komunikacija sa slave-om.

U glavnom programu isčitava se stanje tastera korišćenjem digitalRead(), ako je taster pritisnut promenljiva x- koja će biti prosleđena preko SPI magistrale se postavlja na 1, a ako nije pritisnut na 0. Nakon toga kreće slanje podataka, SS pin se funkcijom digitalWrite() postavlja na LOW, u promenljivu masterSalje se upiše trenutna vrednost x, a zatim se korišćenjem masterPrima=SPI.transfer(masterSalje) izvrši full-duplex komunikacija, istovremeno u promenljivu masterPrima se upisuju podaci koji stižu iz slave uređaja i ka slave-u se iz mastera šalju podaci iz promenljive masterSalje. Ako je u promenljivu masterPrima upisana 1, LED na master Arduinu se uključuje, odnosno isključuje kada je stigla 0.

```
#include <SPI.h>
#define ledPin 7
#define tasterPin 2
#define ss 10
int taster, x;
byte masterSalje, masterPrima;
void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(tasterPin, INPUT);
    SPI.begin();
    SPI.setClockDivider(SPI_CLOCK_DIV8);
    digitalWrite(ss, HIGH);
}
void loop() {
    taster=digitalRead(tasterPin);
    if(taster==LOW){
        x=1;
    }
    else{
        x=0;
    }
    digitalWrite(ss, LOW);
    masterSalje=x;
    masterPrima=SPI.transfer(masterSalje);
    if(masterPrima==1){
        digitalWrite(ledPin, HIGH);
    }
    else{
        digitalWrite(ledPin, LOW);
    }
}
```



SPI.setClockDivider (DELILAC)

Postavlja delilac taktnog signala u odnosu na sistemski taktni signal
DELILAC: SPI_CLOCK_DIV2, SPI_CLOCK_DIV4, 8, 16, 32, 64, 128

Program slave Arduina takođe počinje uključivanjem SPI.h biblioteke za rad sa SPI protokolom, definišu se pinovi tastera, LEDa, kao i MISO pin preko koga se slave slati podatke masteru, zato mora biti deklarisan kao izlazni. Promenljive koje će čuvati primljene podatke i koje će čuvati vrednosti koje se šalju, tj. primaju iz mastera deklariraju se kao volatile jer će se njihove vrednosti menjati u nekoliko funkcija. U okviru funkcije setup() posle deklaracije pinova podešava se **SPI kontrolni registar SPCR** (slika 41), čiji se bit SPE – SPI enable setuje na 1 čime se omogućava bilo koja SPI operacija, tako da će ARD2 biti spreman kao slave da izađe na SPI magistralu. Da bi se pristupilo direktno registru mikrokontrolera koristi se: **SPCR |= 1 << SPE**, bit SPE u registru SPCR setuje se na 1. Logička promenljiva primljeno uzima vrednost *false*, dok se ne primi poruka od mastera. Funkcijom **SPI.attachInterrupt()** se dodeljuje ISR koji će se dešavati svaki put kada se završi SPI transfer između uređaja na magistrali.

Prekidna rutina ISR izvršavaće se posle svakog transfera na SPI magistrali, što je definisano **AVR Interrupt vektorom SPI_STC_vect (SPI Serial Transfer Complete)**. Naime, ovi prekidni vektori su definisani za familiju AVR mikrokontrolera, naziv vektora je identifikator koji se koristi na početku ISR-a. U ovom projektu, kada se završi SPI transfer vektor SPI_STC_vect se setuje, prekida se izvršenje glavnog programa i ulazi se u ISR gde se u promenljivu slavePrima upiše sadržaj registra podataka mikrokontrolera, **SPDR – SPI data registar**. Ovde se logička promenljiva primljeno postavlja na *true*, što je fleg za glavni program da su podaci stigli.

U glavnom programu ispituje se sadržaj promenljive primljeno, ako je *true*, ispituje se da li je slavePrima jednako 1, tada se sa digitalWrite() uključuje LED na slave strani, jer to znači da je na master strani pritisnut taster i obrnuto, ako je slavePrima jednako 0, LED se isključuje. Takođe, u glavnom programu se iščitava stanje tastera na slave strani i ako je pritisnut u promenljivu x, koja će biti poslata masteru se upisuje 1, odnosno 0 ako taster nije pritisnut. Da bi slave Arduino poslao poruku masteru potrebno je u promenljivu slaveSalje smestiti bajt podataka koji se šalje, a potom u registar podataka mikrokontrolera SPDR upisati sadržaj promenljive slaveSalje.

U ovom projektu direktno se pristupa pojedinim SPI registrima mikrokontrolera, opis svih registara i njihovih bitova dat je u datasheetu AVR ATmega328P mikrokontrolera. Na slici 41 dat je pregled SPI registara ovog mikrokontrolera: SPCR (SPI Control Register), SPSR (SPI Status Register), SPDR (SPI Data Register). Ukratko, bitovi kontrolnog registra su: SPIE – interrupt enable, SPE – enable, DORD – data order, MSTR – master/slave select, CPOL – clock polarity, CPHA – clock phase, SPRI1, SPRO – clock rate select. Kod SPSR registra koriste se bitovi SPIF – SPI interrupt flag, WCOL – collision flag, SPI2X – double SPI speed bit. SPDR je 8-bitni registar iz koga može da se čita i u koga može da se piše, koristi se za prenos podataka između registarskog fajla i SPI pomeračkog registra. Upisivanje u ovaj registar inicira prenos podataka, a čitanje registra uzrokuje da se čita pomerački registar prijemnog bafera.


```

#include <SPI.h>
#define ledPin 7
#define tasterPin 2
#define MISO 12
volatile boolean primljeno;
volatile byte slaveSalje, slavePrima;
int taster, x;
void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(tasterPin, INPUT);
    pinMode(MISO, OUTPUT);
    SPCR |= 1 << SPE;
    primljeno = false;
    SPI.attachInterrupt();
}

ISR(SPI_STC_vect) {
    slavePrima = SPDR;
    primljeno = true;
}

void loop() {
    if (primljeno) {
        if (slavePrima == 1) {
            digitalWrite(ledPin, HIGH);
        }
        else {
            digitalWrite(ledPin, LOW);
        }
        taster = digitalRead(tasterPin);
        if (taster == LOW) {
            x = 1;
        }
        else {
            x = 0;
        }
        slaveSalje = x;
        SPDR = slaveSalje;
    }
}

```

SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Slika 41. SPI registri AVR ATmega 328P mikrokontrolera (preuzeto iz [datasheeta](#))

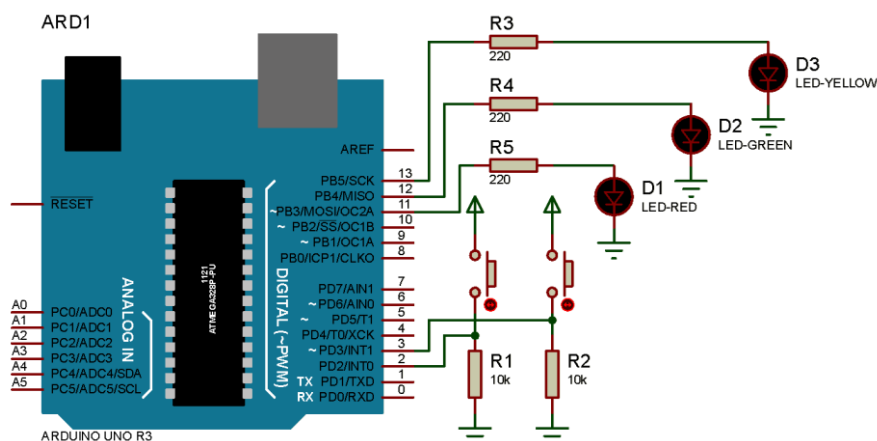
Konačni automati

PROJEKAT 28.

Simulirati kolo Arduina sa dva tastera u pull-down konfiguraciji i tri LED-a: crvenom, zelenom i žutom u programu Proteus. Napisati program za Arduino koji detektuje sekvencu pritiska tastera određenim redom: 1-2-2-1-2, ako su tasteri pritiskani zadatim redom uključuje se zeleni LED, u suprotnom svetli crveni LED. Za vreme unošenja sekvence svetli žuti LED. Zadatak rešiti korišćenjem koncepta konačnih automata.

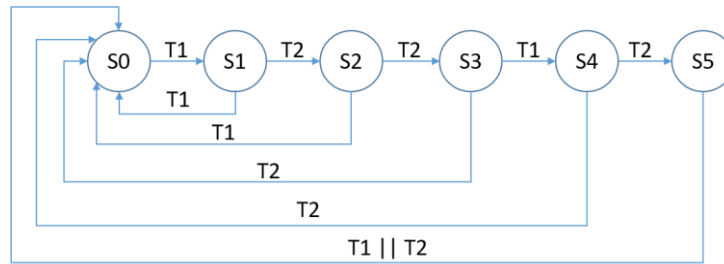
Rešenje:

Električna šema Arduina sa dva tastera u *pull-down* konfiguraciji i tri LED-a data je na slici 42. U ovom projektu biće korišćen **koncept konačnih automata (finite state machine)** – poseban pristup u sintezi digitalnih sistema. Konačni automat čine ulazi, izlazi i stanja. Postoje Milijev i Murov tip automata. **Murov konačni automat** prelazi u sledeće stanje na osnovu trenutnog stanja, dok **Milijev konačni automat** prelazi u sledeće stanje uslovljen i trenutnim stanjem i trenutnom vrednošću na ulazu u sistem, tako da se u praksi najčešće koriste Milijevi konačni automati.



Slika 42. Električna šema Arduina sa dva tastera i tri LED-a

Pre pisanja koda potrebno je formirati dijagram prelaza konačnog automata, odnosno identifikovati sva stanja automata, neohodne aktivnosti sistema u tim stanjima i uslove prelaska među stanjima. Dijagram prelaza sastoji se od stanja (upisana u kružnice) i uslova usled koga se prelazi u naredno stanje (piše se na strelici između stanja). U ovom projektu, sistem inicijalno počinje iz stanja S0 i u njemu ostaje dok se ne pritisne taster 1, tada prelazi u stanje S1. Ako korisnik pritisne taster 2 prelazi se u stanje S2 – uspešno nastavljena zadata sekvenca, a ako pritisne taster 1, vraća se u stanje S0 – prekinuta zadata sekvenca. Dalje, iz S2 prelazi u S3 ako je pritisnut opet taster 2 (nastavljena sekvenca), tj. vraća se u S0 ako je pritisnut taster 1 (prekinuta sekvecna). I tako redom, dok se iz stanja S4, ako je pritisnut taster 2 pređe u finalno stanje S5. Da bi se vratili na početak, iz S5 u S0 potrebno je da bude pritisnut bilo koji taster. Dijagram prelaza konačnog automata iz ovog projekta dat je na slici 43.



Slika 43. Dijagram prelaza konačnog automata za detekciju sekvence 1-2-2-1-2

Na početku programa deklariraju se konstante i promenljive, specijalno se definiše nabrojivi tip podatka `konacniAutomat` koji definiše moguća stanja sistema `S0`, `S1`, `S2`, `S3`, `S4`, `S5`, korišćenjem `typedef enum{S0, S1, S2, S3, S4, S5} konacni Automat`. "trenutnoStanje" predstavlja promenljivu ovog tipa i označava u kom stanju se trenutno nalazi konačni automat. Prekidne rutine se izvršavaju kada se tasteri pritisnu, u okviru ISR-a odgovarajuće promenljive (flegovi) se postavljaju na `true`. U glavnom programu, korišćenjem switch-case strukture se implementira projektovani konačni automat. Kroz switch se predaje trenutno stanje, a za svako stanje se definiše case. Na primer, ako je trenutno stanje `S0`: uključuje se crveni LED i ispituje stanje tastera 1, ako je pritisnut isključuje se crveni LED, a trenutno stanje se postavlja na `S1` i fleg taster1 se postavlja na `false`. Ako je trenutno stanje `S1`: uključuje se žuti LED, ispituje se stanje tastera 2, ako je pritisnut- fleg za taster se postavlja na `false` i trenutno stanje postaje `S2`; ispituje se i taster 1, ako je pritisnut, isključuje se žuti LED i trenutno stanje postaje `S0`... Na kraju, u slučaju stanja `S5`: uključuje se zeleni LED i ispituje se da li je jedan od tastera pritisnut, ako jeste isključuje se zeleni LED i trenutno stanje postaje `S0`, flegovi oba tastera se postavljaju na `false`. Svaki case završava se sa `break`.

```

const int taster1Pin=2;
const int taster2Pin=3;
const int crvenaPin=11;
const int zelenaPin=12;
const int zutaPin=13;

volatile boolean taster1=false;
volatile boolean taster2=false;

typedef enum{
    S0,S1,S2,S3,S4,S5
} konacniAutomat;

konacniAutomat trenutnoStanje;

void t1ISR(){
    taster1=true;
}

void t2ISR(){
    taster2=true;
}

void setup() {
    pinMode(taster1Pin, INPUT);
    pinMode(taster2Pin, INPUT);
    pinMode(crvenaPin, OUTPUT);

```

```

pinMode(zelenaPin, OUTPUT);
pinMode(zutaPin, OUTPUT);
attachInterrupt(digitalPinToInterrupt(taster1Pin), t1ISR, RISING);
attachInterrupt(digitalPinToInterrupt(taster2Pin), t2ISR, RISING);
trenutnoStanje=S0;
}

void loop() {
  switch(trenutnoStanje) {
    case S0:
      digitalWrite(crvenaPin, HIGH);
      if(taster1==true) {
        digitalWrite(crvenaPin, LOW);
        trenutnoStanje=S1;
        taster1=false;
      }
      break;
    case S1:
      digitalWrite(zutaPin, HIGH);
      if(taster2==true) {
        taster2=false;
        trenutnoStanje=S2;
      }
      if(taster1==true) {
        taster1=false;
        digitalWrite(zutaPin, LOW);
        trenutnoStanje=S0;
      }
      break;
    case S2:
      digitalWrite(zutaPin, HIGH);
      if(taster2==true) {
        taster2=false;
        trenutnoStanje=S3;
      }
      if(taster1==true) {
        taster1=false;
        digitalWrite(zutaPin, LOW);
        trenutnoStanje=S0;
      }
      break;
    case S3:
      digitalWrite(zutaPin, HIGH);
      if(taster1==true) {
        taster1=false;
        trenutnoStanje=S4;
      }
      if(taster2==true) {
        taster2=false;
        digitalWrite(zutaPin, LOW);
        trenutnoStanje=S0;
      }
      break;
    case S4:
      digitalWrite(zutaPin, HIGH);
      if(taster2==true) {
        taster2=false;
        digitalWrite(zutaPin, LOW);
        trenutnoStanje=S5;
      }
  }
}

```

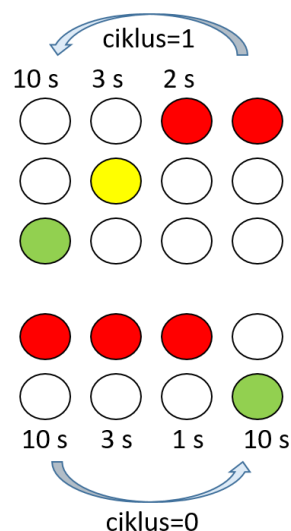
```

    if(taster1==true){
        taster1=false;
        digitalWrite(zutaPin,LOW);
        trenutnoStanje=S0;
    }
    break;
case S5:
    digitalWrite(zelenaPin,HIGH);
    if(taster1==true || taster2==true){
        taster1=false;
        taster2=false;
        trenutnoStanje=S0;
        digitalWrite(zelenaPin,LOW);
    }
    break;
}
}
}

```

PROJEKAT 29.

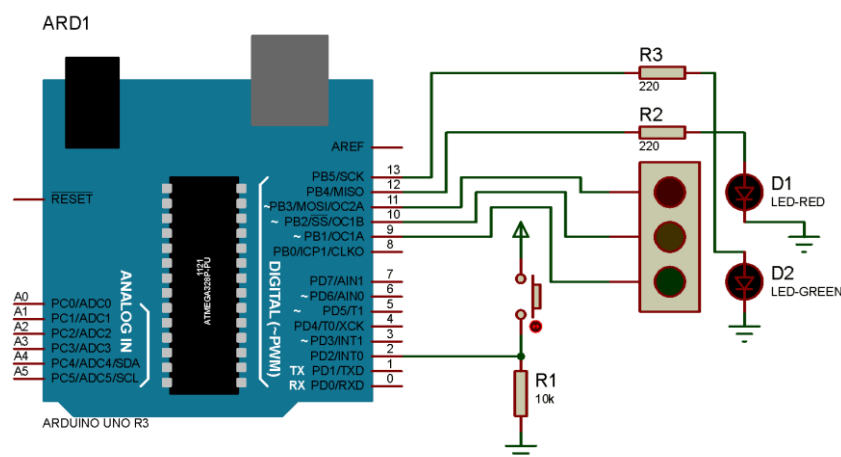
Simulirati kolo sa Arduinoom koje upravlja semaforom (vozila + pešaci) u programu Proteus. Na Arduino treba vezati 5 LED-ova i taster kojim pešak traži zeleno svetlo. Napisati program za Arduino koji realizuje cikluse rada semafora prikazane na slici 44. Na zahtev pešaka može se uključiti zeleno svetlo za pešake, ako je pre toga zeleno za vozila bilo uključeno minimalno 10 s. Zeleno svetlo za pešake ne uključuje se odmah nakon pritiska tastera, već se za vozila uključuje žuto u trajanju od 3 s, zatim se uključuje crveno za vozila, pa se tek posle 1 s uključuje zeleno za pešake. Kada se uključi zeleno za pešake, ostaje 10 s, a onda se uključuje crveno za pešake, pa posle 2 s kreće žuto u trajanju od 3s i konačno opet se uključuje zeleno za vozila. Nakon toga ciklus 0 se ponavlja. Zadatak rešiti korišćenjem koncepta konačnih automata.



Slika 44. Ilustracija stanja semafora za vozila i pešake

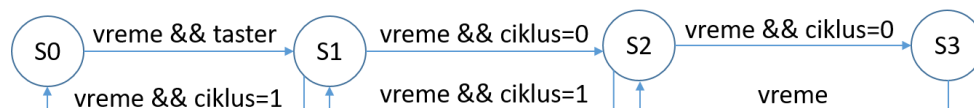
Rešenje:

Simulaciona šema Arduina sa LED-ovima i tasterom u *pull-down* konfiguraciji data je na slici 45.



Slika 45. Električna šema Arduina sa LED-ovima i tasterom u *pull-down* konfiguraciji

Na osnovu ilustracije sa slike 44 jasno je da sistem semafora ima 4 stanja: S0 – zeleno za vozila, crveno za pešake, S1 – žuto za vozila, crveno za pešake, S2 – crveno za vozila, crveno za pešake, S3 – crveno za vozila, zeleno za pešake. Uslovi prelaza među stanjima jesu: da li je taster pritisnut, da li je isteklo potrebno vreme i koji ciklus je u toku (od zelenog ka crvenom za vozila ili od crvenog ka zelenom za vozila). U skladu s tim formiran je dijagram prelaza konačnog automata semafora prikazan na slici 46. Inicijalno sistem je u stanju S0, ako je isteklo potrebno vreme i taster je pritisnut prelazi se u stanje S1. Iz stanja S1 sistem prelazi u stanje S2 ako je isteklo potrebno vreme i aktuelan je ciklus 0, u suprotnom – ako je isteklo vreme i aktuelan je ciklus 1, sistem se vraća u stanje S0. Sistem iz stanja S2 prelazi u stanje S3 ako je isteklo potrebno vreme i ciklus je 0, a ako je ciklus 1, vraća se u S1. Konačno, sistem iz stanja S3 prelazi u S2 ako je isteklo potrebno vreme.



Slika 46. Dijagram prelaza konačnog automata semafora

Na početku programa uključena je biblioteka za Timer 1 koji će se koristiti za merenje vremenskih intervala semafora, definisani su pinovi na koje su vezani LED-ov i taster. Deklarisane su promenljive taster i vreme čije će se vrednosti menjati kroz nekoliko funkcija pa su zato volatile i logička promenljiva ciklus. Zatim su definisane konstante koje čuvaju vrednosti vremena pojedinačnih stanja semafora u mikrosekunadama, a s obzirom da su ta vremena 10, 3, 2 i 1 sekunda, tip konstanti je unsigned long. Definisan je nabrojivi tip podatka konacniAutomat sa mogućim stanjima S0, S1, S2, S3. Promenljiva trenutnoStanje je tipa konacniAutomat. U funkciji setup() definisani su pinovi LED-ova kao izlazni, tastera kao ulazni, dodeljeni su ISR-ovi na promenu stanja tastera sa stanja logičke nule, u stanje logičke jedinice, kao i Tajmeru 1. Prvi period tajmera postavljene je na period zelenog svetla 10 s, trenutno inicijalno stanje je S0 i inicijalni ciklus je *false*=0. U ISR-u tastera, postavlja se fleg taster na *true*, a u ISR-u tajmera postavlja se fleg vreme na *true*.

Dijagram prelaza konačnog automata semafora sa slike 46 implementira se u glavnom programu korišćenjem switch-case strukture. Switch uzima trenutno stanje i ispituje

slučajeve. U slučaju stanja S0: postavljaju se odgovarajuća stanja LED-ova (uključeno zeleno za vozila i crveno za pešake), zatim se ispituje da li je isteklo 10s i pritisnut je taster, ako jeste na *false* se vraćaju flegovi vreme, taster i ciklus, a trenutno stanje postaje S1 i podešava se novi period tajmera korišćenjem funkcije **Timer1.setPeriod()**. Argument ove funkcije je vrednost brojača tajmera u mikrosekundama nakon koga dolazi do ponovnog prekida. U slučaju stanja S1: postavljaju se odgovarajuća stanja LED-ova (uključeno žuto za vozila i crveno za pešake), zatim se ispituje da li je isteklo postavljeno vreme, ako jeste fleg za vreme se vraća na *false* i ispituje se fleg ciklus, ako je *false*, trenutno stanje postaje S2 i podešava se novi period tajmera na 1 s; a ako je *true*, onda je trenutno stanje S0 i period tajmera se podešava na 10 s. U slučaju stanja S2: postavljaju se odgovarajuća stanja LED-ova (uključeno crveno za vozila i crveno za pešake), ispituje se da li je istekao zadati period, ako jeste fleg vreme se vraća na *false*, pa se ispituje ciklus; ako je ciklus=*false*, trenutno stanje postaje S3 i podešava se tajmer na period 10 s; a ako je ciklus=*true*, trenutno stanje će biti S1 i period tajmera se postavlja na 3 s. Konačno, u slučaju stanja S3: postavljaju se odgovarajuća stanja LED-ova (uključeno crveno za vozila i zeleno za pešake), ispituje se fleg vreme, kada je *true* – prelazi u *false*, a ciklus u *true* i postavlja se trenutno stanje u S2, te setuje period tajmera na 2 s.

```
#include <TimerOne.h>
#define tasterPin 2
#define crvenoKolaPin 11
#define zutoKolaPin 10
#define zelenoKolaPin 9
#define crvenoPesakPin 12
#define zelenoPesakPin 13

volatile boolean taster=false;
volatile boolean vreme=false;
boolean ciklus;

const unsigned long zelenoPeriod=10000000;
const unsigned long zutoPeriod=3000000;
const unsigned long crvenoKolaPeriod=1000000;
const unsigned long crvenoPesakPeriod=2000000;

typedef enum{
    S0,S1,S2,S3
} konacniAutomat;

konacniAutomat trenutnoStanje;

void setup() {
    pinMode(crvenoKolaPin,OUTPUT);
    pinMode(zutoKolaPin,OUTPUT);
    pinMode(zelenoKolaPin,OUTPUT);
    pinMode(crvenoPesakPin,OUTPUT);
    pinMode(zelenoPesakPin,OUTPUT);
    pinMode(tasterPin,INPUT);
    attachInterrupt(digitalPinToInterrupt(tasterPin),tasterISR,RISING);
    Timer1.initialize(zelenoPeriod);
    Timer1.attachInterrupt(tajmerISR);
    trenutnoStanje=S0;
    ciklus=false;
}
```

```

void tasterISR() {
    taster=true;
}

void tajmerISR() {
    vreme=true;
}

void loop() {
    switch(trenutnoStanje) {
        case S0:
            digitalWrite(crvenoKolaPin, LOW);
            digitalWrite(zutoKolaPin, LOW);
            digitalWrite(zelenoKolaPin, HIGH);
            digitalWrite(crvenoPesakPin, HIGH);
            digitalWrite(zelenoPesakPin, LOW);
            if(vreme==true && taster==true) {
                vreme=false;
                taster=false;
                ciklus=false;
                trenutnoStanje=S1;
                Timer1.setPeriod(zutoPeriod);
            }
            break;
        case S1:
            digitalWrite(crvenoKolaPin, LOW);
            digitalWrite(zutoKolaPin, HIGH);
            digitalWrite(zelenoKolaPin, LOW);
            digitalWrite(crvenoPesakPin, HIGH);
            digitalWrite(zelenoPesakPin, LOW);
            if(vreme==true) {
                vreme=false;
                if(ciklus==false) {
                    trenutnoStanje=S2;
                    Timer1.setPeriod(crvenoKolaPeriod);
                }
            }
            else{
                trenutnoStanje=S0;
                Timer1.setPeriod(zelenoPeriod);
            }
            break;
        case S2:
            digitalWrite(crvenoKolaPin, HIGH);
            digitalWrite(zutoKolaPin, LOW);
            digitalWrite(zelenoKolaPin, LOW);
            digitalWrite(crvenoPesakPin, HIGH);
            digitalWrite(zelenoPesakPin, LOW);
            if(vreme==true) {
                vreme=false;
                if(ciklus==false) {
                    trenutnoStanje=S3;
                    Timer1.setPeriod(zelenoPeriod);
                }
            }
            else{
                trenutnoStanje=S1;
                Timer1.setPeriod(zutoPeriod);
            }
            break;
    }
}

```



```
case S3:
    digitalWrite(crvenoKolaPin,HIGH);
    digitalWrite(zutoKolaPin,LOW);
    digitalWrite(zelenoKolaPin,LOW);
    digitalWrite(crvenoPesakPin,LOW);
    digitalWrite(zelenoPesakPin,HIGH);
    if(vreme==true){
        vreme=false;
        ciklus=true;
        trenutnoStanje=S2;
        Timer1.setPeriod(crvenoPesakPeriod);
    }
    break;
}
```

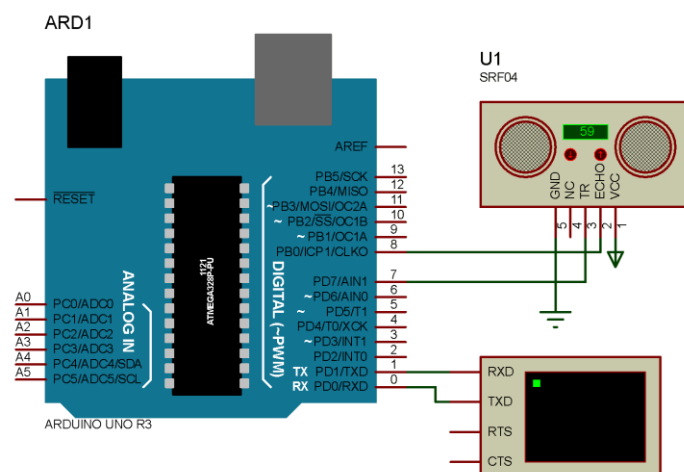
Biblioteke

PROJEKAT 30.

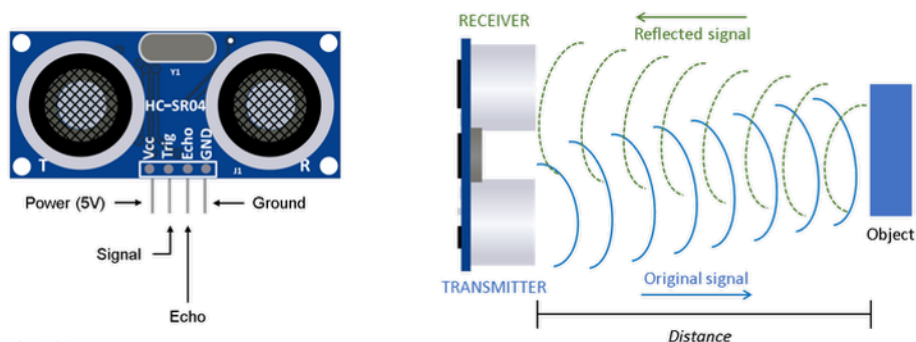
Simulirati kolo Arduina sa ultrazvučnim senzorom SRF04 u programu Proteus. Napisati korisničku biblioteku "ultraZvucni" u programskom jeziku C++ za rad sa senzorom SRF04, u okviru biblioteke treba da bude funkcija razdaljina() koja vraća rastojanje objekta od senzora u cm. Napisati program za Arduino koji uključuje napisanu biblioteku i na svake 2 sekunde šalje očitano rastojanje na virtuelni terminal.

Rešenje:

Simulaciona šema Arduina se ultrazvučnim senzorom SRF04 i virtuelnim terminalom data je na slici 47. Senzor SRF04 pored pinova za napajanje za komunikaciju ima još dva pina: TR-triger pin i ECHO – echo pin, koji su povezani na pinove 7 i 8, respektivno. Arduino je na virtuelni terminal povezan preko RX/TX pinova, 0 i 1. Ultrazvučni senzor emituje iz predajnog zvučnika ultrazvuk na 40 kHz koji putuje kroz vazduh, signal se odbija od objekta/prepreke i vraća nazad u senzorski prijemni detektor. Na osnovu vremena i brzine zvuka moguće je odrediti rastojanje. Na slici 48 prikazan je izgled senzorskog modula SRF04 i ilustrovan je princip rada senzora.

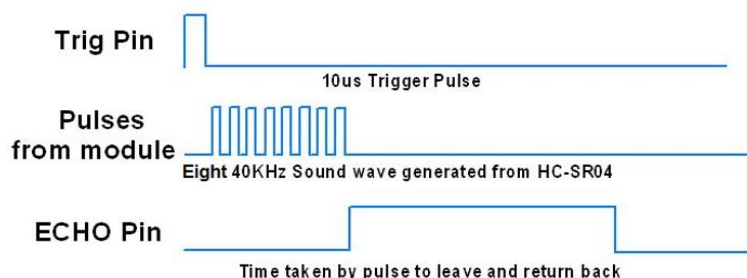


Slika 47. Električna šema Arduina i ultrazvučnog senzorskog modula SRF04



Slika 48. Izgled i ilustracija principa rada ultrazvučnog senzorskog modula SRF04

Za formiranje biblioteke, tj. funkcija za određenu komponentu ili modul potrebno je poznavati detaljno princip rade te komponente ili modula. Neophodno je proučiti datasheet i na osnovu datih specifikacija formirati funkcije. U ovom projektu, potrebno je napisati biblioteku sa funkcijom za određivanje rastojanja objekta od senzora. Da bi dobili ovu informaciju potrebno je na TRIG pin poslati okidni impuls u trajanju od 10 μ s. Nakon okidnog impulsa senzorski modul iz predajnog zvučnika emituje 8 impulsa ultrazvuka frekvencijom 40 kHz, zvuk putuju do objekta i nazad u modul. Kada se zvrši emitovanje zvuka, ECHO pin se interno postavlja u stanje logičke jedinice i ostaje u tom stanju dok emitovani zvuk ne ode do objekta i vrati se nazad u senzorski modul (slika 49). Merenjem vremena t koliko je ECHO pin bio u stanju logičke jedinice moguće je odrediti rastojanje objekta od senzora. Kako je brzina zvuka $v = 340 \text{ m/s} = 0.034 \text{ cm}/\mu\text{s}$, za rastojanje se dobija $s = vt/2$. Uzima se $t/2$ jer je vreme t proteklo od trenutka kad je zvuk od predajnog zvučnika putovao do objekta i nazad do modula.



Slika 49. Signali ultrazvučnog senzorskog modula SRF04

Biblioteku čine najmanje dva fajla: **header fajl (*.h)** i **source fajl (*.cpp)**. Header fajl sadrži definiciju biblioteke, u osnovi spisak svega što je unutar biblioteke, dok source fajl sadrži stvarni kod. U ovom projektu, kreiramo biblioteku `ultraZvucni`, pa će header fajl biti `ultraZvucni.h`. **Header fajl sadrži klasu** `ultraZvucni`. Klasa je jednostavno skup funkcija i promenljivih na jednom mestu. Ove **funkcije i promenljive** mogu biti **javne**, što znači da njima mogu pristupiti korisnici biblioteke; ili **privatne**, što znači da im se može pristupiti samo iz same klase. **Svaka klasa ima** posebnu funkciju poznatu kao **konstruktor**, koji se koristi za kreiranje instance klase. Konstruktor ima isto ime kao i klasa: `ultraZvucni(int TR, int ECHO)` i nema povratni tip. U okviru header fajla potrebno je uključiti `#include "Arduino.h"` koji omogućava pristup standardnim tipovima i konstantama Arduino jezika. Ovaj fajl se automatski dodaje sketch-u, ali u bibliotekama mora biti posebno uključen. Sadržaj header fajla, koji je opisan mora se staviti između `#ifndef`, `#define` i `#endif` što sprečava problem kompajliranja ako korisnik uključi istu biblioteku dva puta.

```
#ifndef ultraZvucni_h
#define ultraZvucni_h

#include "Arduino.h"

class ultraZvucni
{
public:
    ultraZvucni(int TR, int ECHO);
    float razdaljina();
};
```

```
private:
    int _TR;
    int _ECHO;
};

#endif
```

Source fajl (*.cpp) počinje naredbama `#include "Arduino.h"` i `#include "ultraZvucni.h"`, kojim se ostatku koda daje pristup standardnim Arduino funkcijama i definicijama u našem header fajlu. Zatim dolazi konstruktor- šta bi trebalo da se desi kada neko kreira instancu na našu klasu. U ovom projektu, korisnik specificira koje pinove želi da koristi, a oni se ovde konfigurišu- TR kao izlazni i ECHO kao ulazni i vrednosti se proslede u privatne promenljive za korišćenje u drugim funkcijama. Funkcija-konstruktor `ultraZvucni` je deo klase `ultraZvucni`, zato se piše **`ultraZvucni::ultraZvucni()`**. Po konvenciji, privatne promenljive počinju sa `_`, a imena mogu biti proizvoljna.

U nastavku *.cpp fajla nalazi se opis funkcije koja će se pozivati u Arduino programu, specificira se tip te funkcije, a pošto je ona deo klase `ultraZvucni`, piše se: **`ultraZvucni::razdaljina()`**. Između `{}` se piše funkcionalnost, pri čemu funkcija radi sa privatnim promenljivama. U ovom projektu, pin TR, kome se pristupa preko privatne `_TR` se spušta na LOW, zatim se posle 2 μ s, postavlja na visok naponski nivo koji traje 10 μ s, a potom se opet spušta na LOW stanje, čime se generiše okidni impuls da senzorski modul počne emitovanje ultrazvuka, interno senzorski modul podiže ECHO pin na HIGH, osluškuje emitovani zvuk, i vraća se u LOW kada se zvuk koji je emitovan vrati u modul. Kreira se promenljiva tipa long koja će čuvati vreme za koje je ECHO pin bio u stanju logičke jedinice. Za isčitavanje vremena koristi se Arduino funkcija **`pulseIn(pin, STANJE)`**. Argumenti ove funkcije su pin na kome se meri vreme trajanja STANJA i STANJE. U ovom slučaju, meri se vreme dok je ECHO pin, kome se pristupa preko `_ECHO` bio u stanju HIGH. Na kraju, na osnovu izmerenog vremena i poznate brzine zvuka funkcija `razdaljina()` vraća rastojanje od objekta do senzorskog modula u cm, što se postiže sa **`return`** izraz. Rezultat će biti u float formatu, jer je funkcija deklarirana kao float.

```
#include "Arduino.h"
#include "ultraZvucni.h"

ultraZvucni::ultraZvucni(int TR, int ECHO)
{
    pinMode(TR, OUTPUT);
    pinMode(ECHO, INPUT);
    _TR=TR;
    _ECHO=ECHO;
}

float ultraZvucni::razdaljina()
{
    digitalWrite(_TR, LOW);
    delayMicroseconds(2);
    digitalWrite(_TR, HIGH);
    delayMicroseconds(10);
    digitalWrite(_TR, LOW);
    long vreme = pulseIn(_ECHO, HIGH);
    return vreme*0.0175;
}
```

Kompajliranjem biće kreirani `ultraZvucni.h` i `ultraZvucni.cpp`, koji predstavljaju biblioteku koja se dalje može distribuirati. Kreirati novi sketch i u njega uključiti kreiranu biblioteku (Sketch/Import Library), na početku programa naredbom `#include "ultraZvucni.h"` ostatak koda dobija pristup funkcijama naše biblioteke. U nastavku su deklarirani pinovi na koje je povezan senzorski modul, a zatim je ***kreirana instanca na ultraZvucni klasu pod nazivom `uz(triggerPin,echoPin)`***. Kada se ova linija koda izvrši, konstruktor klase `ultraZvucni` će biti pozvan, tako da u okviru `setup()` funkcije neće biti potrebno definisati pinove sa `pinMode()` jer se to sada dešava unutar biblioteke, kada je kreirana instanca.

U beskonačnoj petlji, u okviru funkcije `Serial.print()` poziva se funkcija **`uz.razdaljina()`** sa ciljem da se dobije rezultat – rastojanje objekta od senzorskog modula. Funkcija se mora pozvati sa na početku formiranom instancom `uz`. Moguće je kreirati više instanci klase `ultraZvucni`, što omogućava da se poveže više senzorskih modula na isti Arduino uz korišćenje kreirane biblioteke.

```
#include "ultraZvucni.h"
int triggerPin=7;
int echoPin=8;

ultraZvucni uz(triggerPin,echoPin);
void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.print("Rastojanje od objekta je: ");
    Serial.print(uz.razdaljina());
    Serial.println(" cm.");
    delay(2000);
}
```

Literatura

1. Arduino Projects Book, ed. S. Fitzgerald, M. Shiloh, T. Igoe, Arduino LLC, 2012.
2. Arduino Cookbook, M. Margolis, O'Reilly, 2011.
3. Arduino Development Cookbook, C. Amariei, Packt Publishing
4. Arduino Uno, uputstvo za korišćenje sa rešenim primerima, M.Tanasković
5. Internet izvori: <https://www.arduino.cc/reference/en/>