Sustav dodjele prioriteta pametne utičnice

Križanec, Domagoj

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij

SUSTAV DODJELE PRIORITETA PAMETNE UTIČNICE

Diplomski rad

Domagoj Križanec

Osijek, 2019.

SADRŽAJ

1. UVOD

1.1. Opis praktičnog dijela rada

Zadatak diplomskog rada je napraviti sustav upravljan mikrokontrolerom ATmega16 koji će dodjeljivati prioritet korisnicima za upravljanje pojedinim utičnicama u određenom vremenskom terminu.

Korisnici će se na uređaj priključivati mobilnim uređajima preko *Bluetooth* modula i odabirati vremensko razdoblje u kojem bi koristili određenu utičnicu.

Nakon što korisnik odabere vremensko razdoblje u kojem će koristiti uređaj, niti jedan drugi korisnik neće moći koristiti odabrani uređaj u tom vremenskom razdoblju. Stoga, prvi korisnik koji se spoji na uređaj zapravo ima prioritet nad uređajem u tom vremenskom terminu. Korisnik koji pokušava dobiti taj već odabrani termin dobiva vizualno upozorenje na mobilni uređaj u kojemu se navodi ime korisnika koji je zauzeo taj termin.

Kako bi se realizirao navedeni rad, potrebno je uskladiti rad mikroupravljača sa senzorom za praćenje točnog vremena DS3231, *Bluetooth* antenom HC-06 te relejnim modulom s četiri kanala koji rade na naponu od 5 V. Uz to, potrebno je napisati programski kod u programskom jeziku C po kojemu će se izvoditi sustav. Okruženje u kojemu je pisan i kompiliran program je Atmel Studio, dok se program prebacuje na mikroupravljač pomoću Khazama programatora.

Diplomski rad strukturiran je u četiri glavna poglavlja. Prvo poglavlje odnosi se na hardverske komponente u kojem su ukratko opisani svi elementi potrebni za ostvarivanje sklopa. U idućem poglavlju opisuju se programska rješenja, odnosno sve funkcije koje služe za ispravan rad sklopa. Nadalje, rad sklopa i sve njegove mogućnosti slikovno se prikazuju u poglavlju testiranja sklopa. U zadnjem poglavlju opisuju se problemi s kojima se programer može susresti pri dizajniranju ovakvog rada.

.

2. OPIS HARDVERA

2.1. Mikrokontroler

Na prvi pogled, mikrokontroleri su računala isto kao i stolna računala, laptopi, mobilni uređaji i sl. Budući da mikrokontroleri pripadaju računalima imaju i ista obilježja kao i sve ostale vrste računala, a ona su [1]:

1. Sva računala imaju CPU (engl. *Central Processing Unit* – centralna procesna jedinica) što izvršava pojedini zadatak.
2. Svi CPU-i učitavaju programe iz neke memorije.
3. Računalo sadrži RAM (engl. *Random Access Memory* – radna memorija) u koji sprema određene varijable.
4. Računalo ima ulazne i izlazne jedinice preko kojih komunicira s korisnicima.

Za razliku od stolnih računala koja mogu obavljati tisuće zadataka istovremeno, mikrokontroleri su specijalna računala koja izvršavaju jedan zadatak jako kvalitetno. Koncept izvršenja pojedinog zadatka definiran je programom koji je korisnik učitao u ROM (engl. *Read Only Memory* – iščitavajuća memorija) te se generalno jednom učitava i ne mijenja.

Najčešća primjena mikrokontrolera je izvršenje pojedine funkcije većeg sklopovlja.

Mikrokontroleri su izvedeni unutar kučišta koje je izuzetno otporno na vanjske uvjete jer i sami mikrokontoleri moraju biti u stanju izvršavati funkcije i u najtežim uvjetima, a popularni su zbog činjenice da im je izrada poprilično jeftina.

Većini mikrokontrolera zajedničke su neke karakteristike [2]:

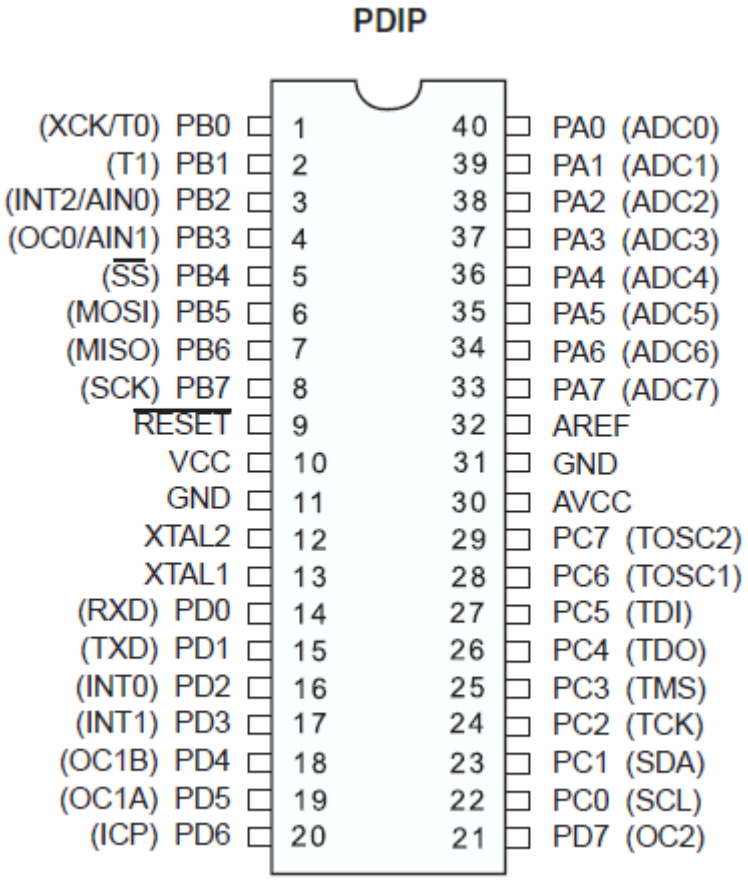
1. Imaju relativno mali radni takt reda 10 MHz.
2. Mali broj jednostavnih instrukcija, red veličine oko 100.
3. RAM reda kB.
4. Stalna memorija s programskim kodom u PROM ili EEPROM izvedbi.
5. Širok raspon napona napajanja.
6. A/D i D/A pretvornici razlučivosti prema namjeni, uobičajeno 8-bitni.

2.1.1. Atmel ATmega16

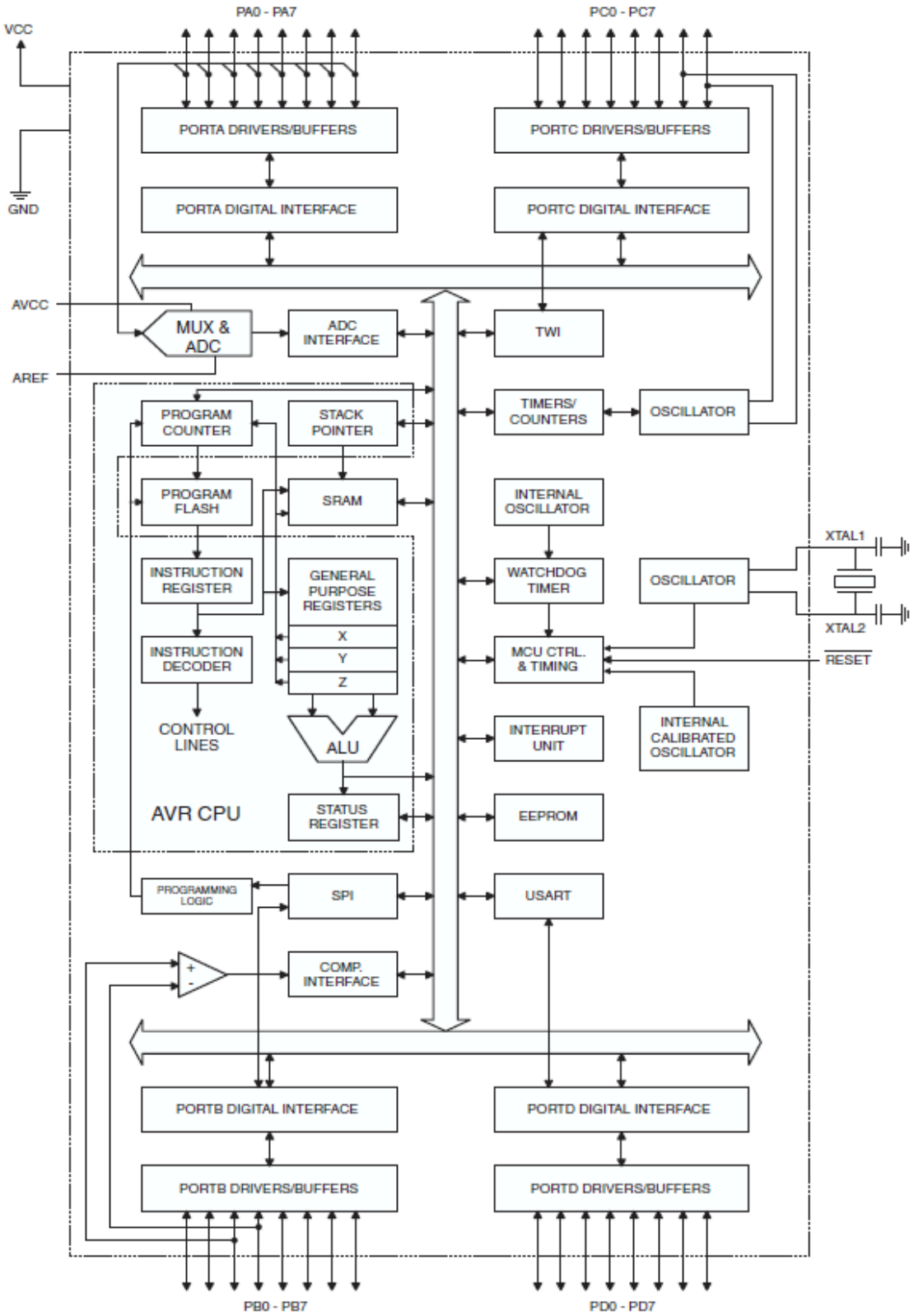
U ovom sklopu koristi se jednojezgreni Atmel ATmega16, 8-bitni AVR mikrokontroler s 16 kB *flash* memorijom, 512 B EEPROM (engl. *Electrically Erasable Programmable Read-Only Memory*) memorijom te 8-kanalnim, 10-bitnim ADC-om (engl. *Analog to Digital Converter*) [3].

ATmega 16 predstavlja niskonaponski CMOS (engl. *Complementary Metal-Oxide-Semiconductor*) mikrokontroler baziran na AVR-ovoj poboljšanoj RISC (engl. *Reduced Instruction Set Computer*) arhitekturi.

Slika 2.1. prikazuje raspored izvoda mikroupravljača Atmel ATmega16 [3], dok slika 2.2. prikazuje blok dijagram unutrašnjosti ATmega16 mikrokontrolera [3].



Slika 2.1. Raspored izvoda Atmel ATmega16 mikroupravljača [3]



Slika 2.2. Blok dijagram unutrašnjosti Atmel ATmega16 mikrokontrolera [3]

2.2. Ploča za programiranje

Postupak prebacivanja programskog koda s računala na mikroupravljač ostvaruje se pomoću ploče za programiranje te USB programatora. Uz postolje za mikroupravljač s izvedenim izvodima za pojedine nožice mikroupravljača, ploča za programiranje sadrži i izvode za napajanje. Izvodi za napajanje sastavljeni su od tri izvoda za 5 V Vcc (engl. *Voltage Common Collector*) i tri izvoda za uzemljenje GND (engl. *Ground*). Na ploču se također spaja i kvarcni (kristalni) oscilator od 11,0592 MHz, kojemu je uloga održavanje frekvencije za mikroupravljač. Odabir oscilatora ovisi o svrsi sustava, no treba zapamtiti kako frekvencija osciliranja nikad ne bi smjela premašiti maksimalnu unutarnju frekvenciju mikroupravljača koja za Atmel ATmega16 iznosi 16 MHz [3]. Slika 2.3. prikazuje izgled korištene ploče za programiranje te USB programatora [4].

Slika 2.3. Ploča za programiranje (lijevo), USB programator (sredina) [4]

2.3. Relej

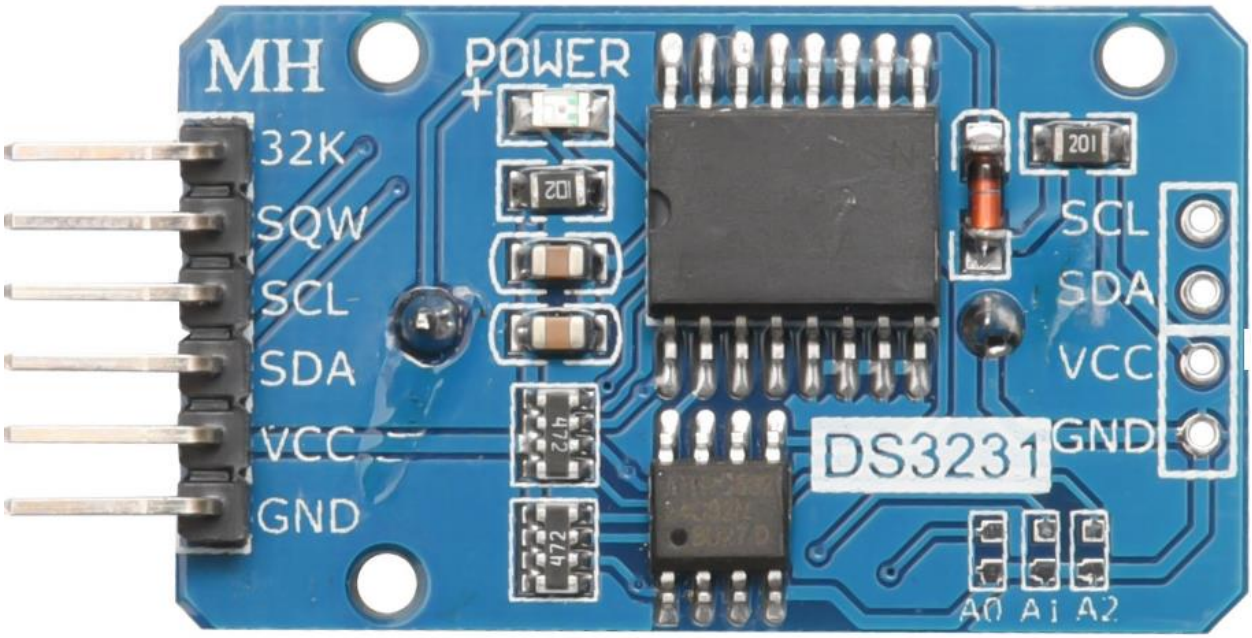
Releji su elektromagnetske sklopke upravljane strujom relativno niske razine koje mogu otvarati i zatvarati struje puno veće razine. U ovom sklopu mikrokontroler upravlja relejima u svrhu otvaranja i zatvaranja izmjeničnog strujnog kruga na koji su spojena trošila. Korišteni relejni modul je 4-kanalni relejni modul od 5 V, što znači da se na njegov izlaz mogu spojiti četiri trošila. Slika 2.4. prikazuje 4-kanalni relejni modul korišten za izradu ovog sustava. [5]

Slika 2.4. 4-kanalni relejni modul [5]

2.4. Senzor za vrijeme

Za očitavanje točnog vremena korisi se RTC (engl. *Real Time Clock*) senzor koji prati točno vrijeme i datum i daje te informacije na raspolaganje. Senzor ima ugrađenu eksternu bateriju tako da se točno vrijeme i datum neće izgubiti čak i u slučaju nestanka struje. Senzor u sebi ima ugrađen kalendar do 2100. godine i može prikazati datum i vrijeme u rezoluciji godina/mjesec/dan/sat/minuta/sekunda. Za ovaj sklop koristi se RTC senzor DS3231 koji spada u industrijsku klasu senzora, te sadrži i senzor očitavanja temperature okoline. Slika 2.5. prikazuje izgled RTC senzora DS3231 [6].

Komunikacija s mikrokontrolerom ostvaruje se pomoću I2C protokola spajajući SCL (engl. *Serial Clock Line*) te SDA (engl. *Serial Data Line*) izvode DS3231 senzora sa SCL i SDA izvodima mikrokontrolera koji se nalaze na mjestima PC0 i PC1.

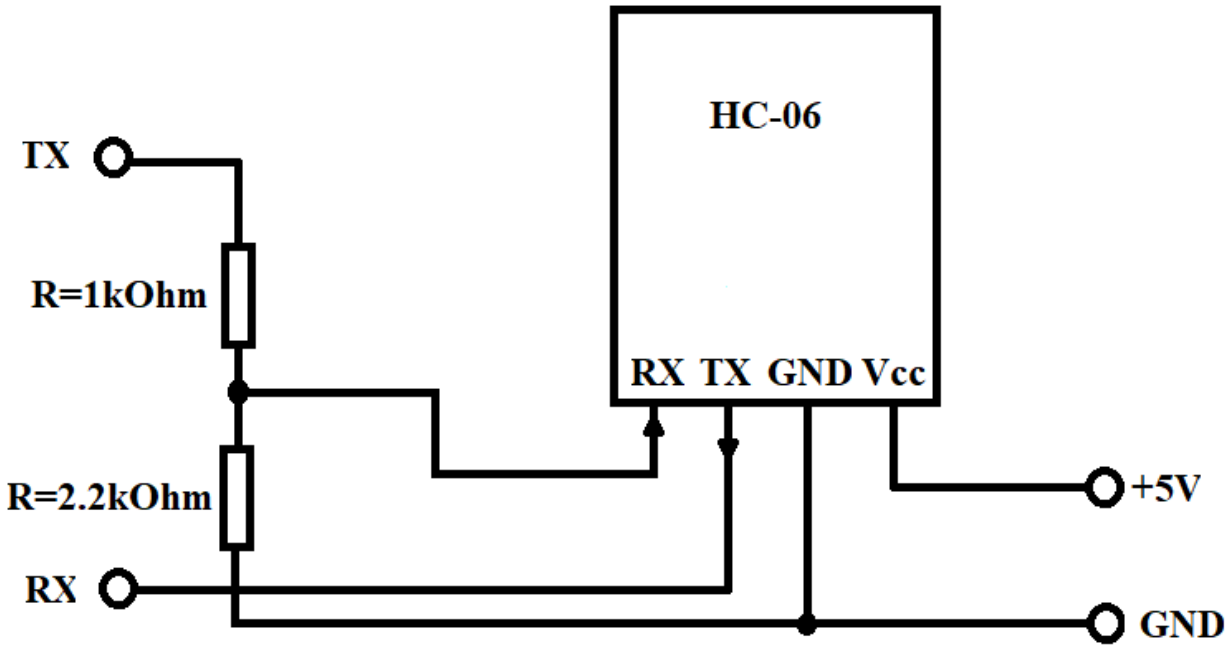


Slika 2.5. RTC senzor DS3231 [6]

2.5. Primopredajna antena

Primopredajna *Bluetooth* antena (engl. *Transreciever*) je modul koji omogućuje serijsku komunikaciju između mikrokontrolera te nekog drugog *Bluetooth* uređaja putem bežične veze. Služi za upravljanje mikrokontrolerom preko mobilnog uređaja i za primanje informacija o sustavu na mobilni uređaj. Antena komunicira s mikrokontrolerom pomoću UART (engl. *Universal Asynchronus Reciever-Transmitter*) serijske komunikacije. RXD (engl. *Recieved data*) izvod spaja se na TXD (engl. *Transmitted data*) izvod mikrokontrolera i obrnuto za ostvarivanje komunikacije između dvaju uređaja.

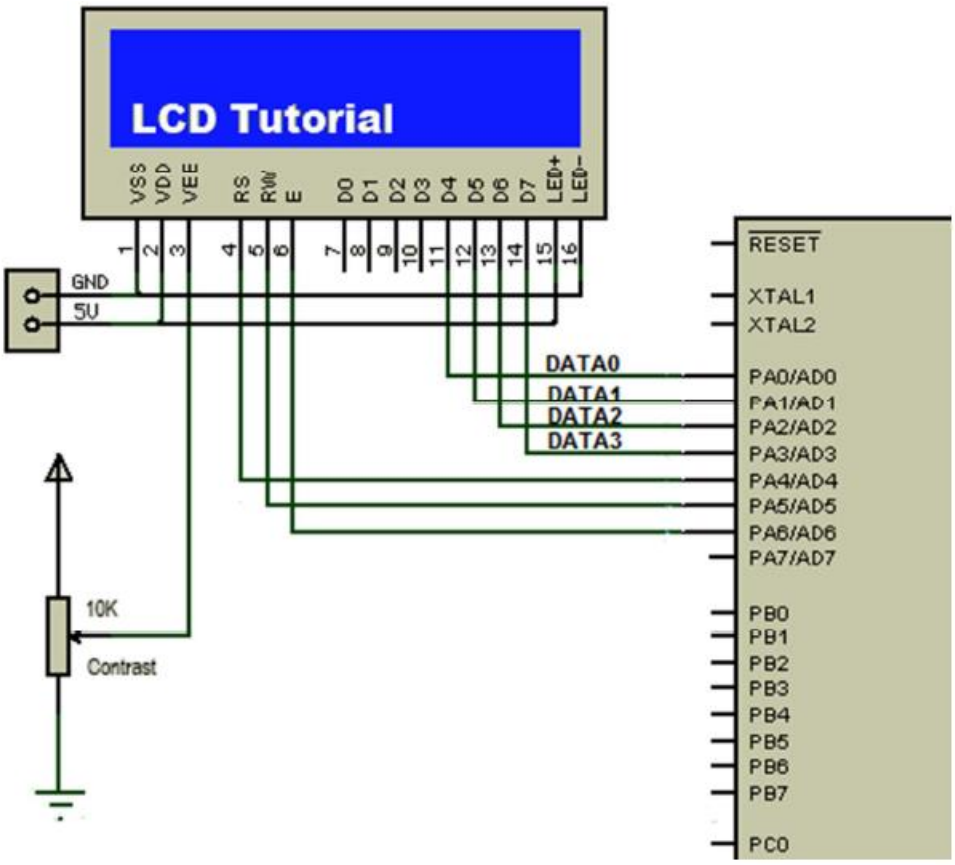
Premda se sam uređaj napaja istosmjernim naponom od 5 V, napon za RXD liniju antene treba se smanjiti na 3,3 V kao što je prikazano na slici 2.6. [7]. To se može postići korištenjem dvaju otpornika od 1 kΩ te 2,2 kΩ.



Slika 2.6. Shema spajanja HC-06 antene [7]

2.6. LCD zaslon

Za prikaz osnovnih informacija sustava poput prikaza točnog vremena i datuma, te korisničkih funkcija unosa vremena paljenja i gašenja pojedinog odabranog releja koristi se 16x2 LCD (engl. *Liquid Crystal Display*). Oznaka 16x2 znači da se na LCD-u može prikazati 16 simbola po 2 reda. Slika 2.7. prikazuje shemu spajanja LCD-a s mikroupravljačem ovisno o odabranom portu [8]. Na slici jest prikazano spajanje LCD-a na port A, ali korisnik može sam odabrati željeni port. Kako bi se prikazali simboli na zaslonu, potrebno je podesiti potenciometar na VEE izvodu tako da se gornji red zaslona vidi, a da donji red na zaslonu ostane prazan.



Slika 2.7. Shema spajanja LCD-a [8]

2.7. Izvedba diplomskog rada

Mikrokontroler ATmega16 napajan je direktno iz računala preko USB (engl. *Universal Serial Bus)* priključka. Napajanje i prebacivanje s računala direktno na mikrokontroler ostvaruje se pomoću USB AVR (engl. *Automatic Voltage Regulator*) programatora.

Za rad sklopa od senzora koristi se HC-06 bežična *Bluetooth* primateljska i odašiljačka antena za komunikaciju sklopa s mobilnim uređajima, te DS3231 senzor za vrijeme koji zbog svoje pomoćne baterije može pamtiti postavljeno vrijeme i nakon isključivanja uređaja. Mikrokontroler trenutno vrijeme i stanje uređaja ispisuje na 16x2 LCD-u. Izlaz sklopa čini relejni modul s četiri kanala, kod kojeg svaki pojedini kanal može otvarati ili zatvarati izmjenični strujni krug, ovisno o naredbama mikrokontrolera te tako upravljati uređajima spojenim na izlaze sklopova.

Pomoću *Bluetooth* terminal mobilne aplikacije, sa svakog prikladnog mobilnog uređaja mogu se slati podatci ili se mogu primati isti od strane mikrokontrolera.

3. RAZVOJ PROGRAMSKOG RJEŠENJA

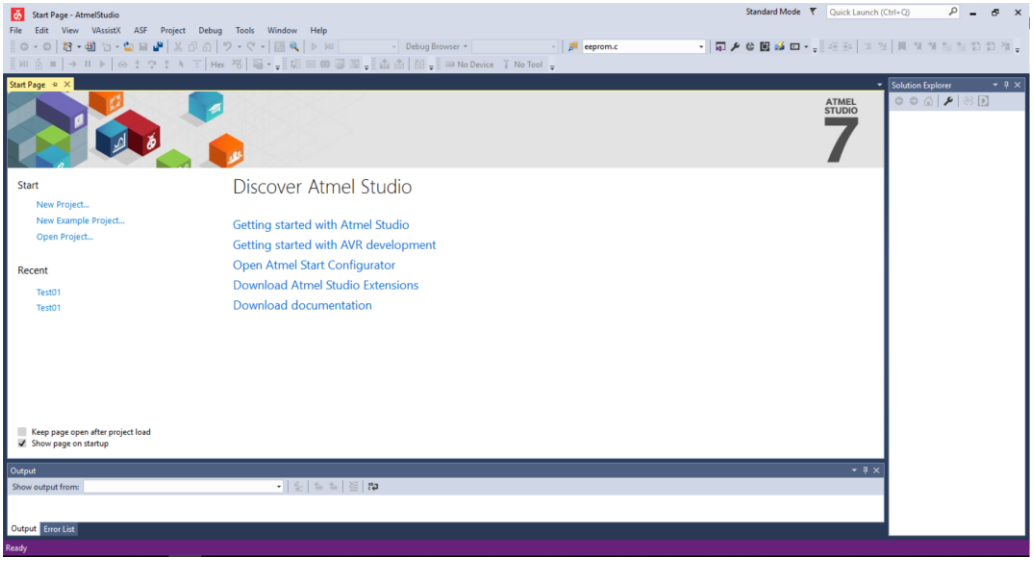
3.1. Atmel Studio 7

Programski kod za sustav pisan je na platformi Atmel Studio 7. Atmel Studio 7 programska je platforma (engl. *Integrated Development Platform*) za razvoj i otklanjanje grešaka za sve aplikacije koje uključuju AVR i SAM mikrokontrolere pisane u C/C++ programskim jezicima. Uz to, Atmel Studio 7 uključuje i *online* aplikaciju *Atmel Gallery* koja korisniku omogućuje proširivanje programske platforme raznim *plug-in* aplikacijama [9].

Neke od osnovnih karakteristika platforme su [9]:

1. Podrška za više od 500 AVR i SAM mikrokontrolera.
2. Velik broj raznih biblioteka uključujući drajvere, više od 1600 primjera projektnih zadataka s originalnim programskim kodom, grafičke usluge itd.
3. Konfiguracija i testiranje performansi bežičnih dizajna korištenjem *Wireless Composer*-a.
4. Pisanje i otklanjanje grešaka C/C++ koda.
5. Potpuna simulacija mikrokontrolera s preciznim modelom CPU-a, *interrupt*-a i periferalnih te eksternih stimulanata.

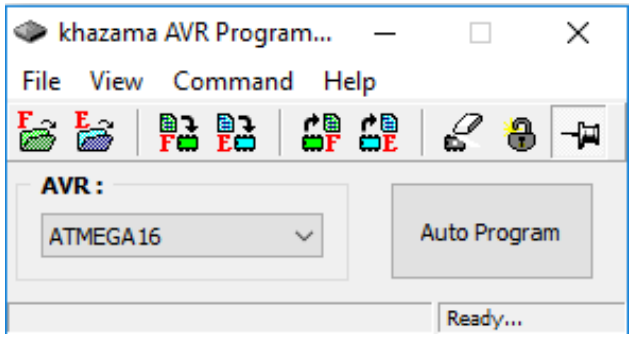
Slika 3.1. prikazuje izgled početne stranice Atmel Studia 7 [7].



Slika 3.1. Atmel Studio 7 [7]

3.2. Khazama AVR programator

Khazama programator služi za implementiranje i prilagodbu programskog koda na AVR mikroupravljače. Glavne pogodnosti su mu te što je malen, jednostavan i brz program za implementaciju. Koristi se tako da se prvo odabere željeni čip, zatim se pomoću funkcije *Load flash to buffer* prenese .hex datoteka projekta u *buffer* programatora, a nakon toga se pomoću funkcije *Auto program* projekt implementira na odabrani mikroupravljač. Khazama podržava i izmjenu *Fuse i Lock* bitova mikrokontrolera, no pri radnji s njima preporuča se izniman oprez jer se lako može zaključati mikrokontroler te postaje beskoristan. Slika 3.2. prikazuje izgled grafičkog sučelja Khazama AVR programatora [7].



Slika 3.2. Khazama AVR programator [7]

4. IZRADA PROGRAMSKOG KODA

4.1. Početne postavke

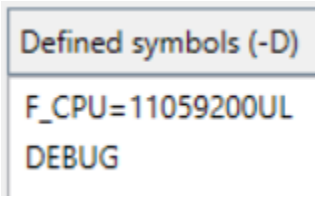
Pri stvaranju novog projekta na platformi Atmel Studio 7 najprije je potrebno odabrati željeni mikrokontroler za koji će se pisati programski kod. Za izradu ovog sustava potrebno je odabrati ATmega16 mikrokontroler. Nakon odabira mikrokontrolera pojavljuje se prva datoteka *main.c* s osnovnom bibliotekom *#include <avr/io.h>*, osnovnom *int main(void)* funkcijom te jednom beskonačnom *while* petljom *while(1).*

Korisnik najprije mora upisati željenu frekvenciju rada mikrokontrolera koja se može definirati iznad osnovnih biblioteka. Slika 4.1. prikazuje definiranje frekvencije iznad biblioteka za primjer frekvencije mikroupravljača od 8 MHz [10]. Ovaj način deklaracije je nepraktičan ukoliko korisnik treba uskladiti više datoteka u projekt jer bi onda korisnik trebao u svakoj pojedinoj datoteci definirati frekvenciju mikroupravljača.

```
#define F_CPU 8000000UL

#include <avr/io.h>
#include <util/delay.h>
```

Slika 4.1. Definiranje frekvencije iznad biblioteka za primjer 8 MHz [10]

Ukoliko korisnik ima više datoteka najlakše je definirirati frekvenciju rada pod *Properties-Toolchain-AVR/GNU C Compiler- Symbols* kao što je prikazano na slici 4.2. za primjer frekvencije 11 MHz [7].



Slika 4.2. Definiranje frekvencije za sve datoteke [7]

Vrijednost F_CPU = 11059200UL je odabrana budući da ima najmanje odstupanje za većinu vrijednosti *baud*-a (simbola po sekundi) [3].

Nakon postavljanja frekvencije za sve datoteke potrebno je postaviti i *lock* bitove unutar Khazama AVR programatora na korištenje eksternog oscilatora da frekvencija bude izjednačena za sve module.

4.2. Funkcije programskog koda

Radi bolje i efikasnije preglednosti programskog koda, razvijeni sustav podjeljen je u više funkcija što se nalaze u osam datoteka, uključujući i *main.c* datoteku. Datoteke koje sadrže kod za rad nazivaju se: *main.c, button.c, display.c, bluetooth.c, i2c.c, lcd.c, rtc3231.c, uart.c* te pripadajuće datoteke sa zaglavljima u kojima su ujedno definirane strukture koje sadrže varijable potrebne za rad sustava. Sve datoteke izuzev *main.c* imaju svoja zaglavlja.

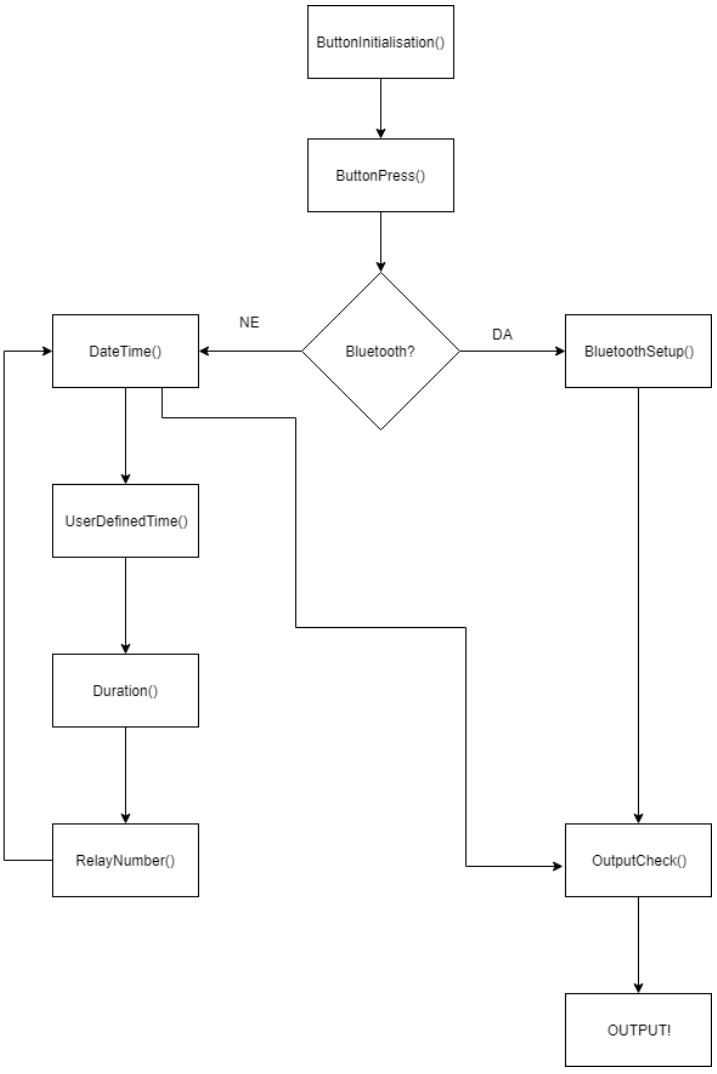
Osnovna funkcija za rad *int main(void)* nalazi se u *main.c* datoteci. Mikrokontroler pri početku rada najprije ulazi u prvu funkciju unutar *main* funkcije koja je *ButtonInitialisation()* funkcija. Ona se nalazi unutar *button.c* datoteke, a svrha joj je inicijalizacija ulazno-izlaznih jedinica. To se ostvaruje pomicanjem bitova iz stanja „0“ u stanje „1“ pojedinih portova. Slika 4.3. prikazuje programski kod koji se nalazi unutar funkcije *ButtonInitialisation()* [7].

```
void ButtonInitialisation()
{
    DDRD = 0x00; // svi pinovi u PORTD stavljeni kao input - pocetno stanje
    DDRD &=~ (1<<PD2)|(1<<PD3)|(1<<PD4)|(1<<PD5)|(1<<PD6); // input mode na pinovima PD2- PD6
    DDRA |= (1<<PA1)|(1<<PA4)|(1<<PA5)|(1<<PA6)|(1<<PA7); // output mode na pinu PA1, PA4, PA5, PA6, PA7
    PORTD = 0x00; //onemoguceni pull-up otpornici za sve PIN-ove porta D
    PORTD |= (1<<PD2)|(1<<PD3)|(1<<PD4)|(1<<PD5)|(1<<PD6); // stavljanje 1 na tipke PD2- PD6
}
```

Slika 4.3. Funkcija ButtonInitialisation() [7]

Iz slike 4.3 možemo vidjeti kako su postavljeni pojedini portovi. Port D postavljen je u *input mode* jer su na njega spojena tipkala preko kojih je korisnik u mogućnosti odabrati željeno vrijeme paljenja i gašenja sustava, te odabrati željeni broj releja. Input mode znači da su izvodi postavljeni u stanje jedinice, a ukoliko se preko tipkala stanje provede do mase, očitava se promjena stanja. Port A postavljen je na *output mode* jer je na njega spojen 4 kanalni relejni modul uz opću LED (engl. *Light Emiting Diode*) koja označava da je jedan od izlaza aktivan.

Nakon funkcije *ButtonInitialisation()*, programski kod ulazi funkciju *ButtonPress()* koja je ujedno i osnova rada cijelog sustava. Iz te funkcije slijedi grananje na jezgreni dio rada u kojemu korisnik može sam preko tipkala i prikaza na LCD-u odabrati željeno vrijeme paljenja i gašenja pojedinog releja ili na mobilni dio preko kojeg korisnik korištenjem svog mobilnog uređaja može odabrati vrijeme paljenja ili gašenja pojedinog releja. Slika 4.4. prikazuje tijek dijagram cijelog sustava [7]. Jezgreni dio sadrži funkcije *DateTtime(), UserDefinedTime(), Duration(), RelayNumber() i OutputCheck()* dok mobilni dio sadrži funkcije *BluetoothSetUp()* te *OutputCheck().*



Slika 4.4. Dijagram tijeka funkcija sustava [7]

4.3. Jezgreni dio

Prvi dio *ButtonPress()* funkcije služi za postavljanje početnih vrijednosti varijabli potrebnih za rad sustava. Postavlja se *baud rate* za *Bluetooth* komunikaciju, inicijalno vrijeme unosa koje je postavljeno na srednje vrijednosti raspona 12/12/30, početna vrijednost datuma koja predstavlja trenutni datum te raznih varijabli koje služe za primitak parametara s mobilnog uređaja. Slika 4.5. prikazuje dio koda funkcije *ButtonPress()* koji služi za inicijalizaciju početnih stanja sustava [7].

```
int state = 0;
uint8_t current = PIND;
uint8_t previous = PIND;
uint8_t uartValue = 0;
uint16_t baudPrescaler, baud = 9600;                                    //definirana vrijednost baud ratea
baudPrescaler = (F_CPU/(16UL*baud))-1;

struct rtc_date date;
struct unesenoVrijeme vrijeme;
struct uneseniDatum datum;
struct vrijemeTrajanja trajanje;
struct info varijable;
struct users korisnik;

rtc3231_read_date(&date);

vrijeme.horizontalno = 0;
vrijeme.hour = 0xC;   //postavljanje vremena unosa- samo jednom
vrijeme.min = 0x1E;   //sve vrijednosti se moraju unijeti u hex obliku
vrijeme.sec = 0x00;

datum.day = date.day;    //0xF;   //postavljanje datuma- samo jednom
datum.month = date.month;  //0x06;  //sve vrijednosti moraju se unijeti u hex obliku
datum.year =  date.year;  //0x12;

trajanje.trajanjeHorizontalno = 0; //postavljanje pocetnog stanja za horizontalni dio displaya trajanja
trajanje.stanjeZaOutputCheck = 0; //postavljanje pocetnog stanja za flag stanja
trajanje.brojReleja = 0x01; //postavljanje broja releja na pocetnu vrijednost
trajanje.hour = 0xC;   //postavljanje vremena gasenja- samo jednom
trajanje.min = 0x1E;   //sve vrijednosti se moraju unijeti u hex obliku
trajanje.sec = 0x00;

varijable.concatenatedNumber = 0;
varijable.uneseneZnamenke = 0;
varijable.unosVrijednostiKorisnika = 0;
varijable.previousUart = 0;
varijable.uartRecieve = 0;
varijable.testI = 0;
varijable.functionFlag = 0;
memset(varijable.FinalUart, 0, sizeof(varijable.FinalUart));    //postavljanje parametra Final Uart u Nulu
memset(varijable.UART, 0, sizeof(varijable.UART));

korisnik.brojKorisnika = 0;
```

Slika 4.5. Inicijalizacija početnih stanja [7]

Nakon inicijalizacije slijedi glavna beskonačna petlja *while(1)* koja je zaslužna za konstantno provjeravanje unesenih stanja ulaza i izlaza. Unutar te beskonačne petlje nalaze se četiri manje petlje koje se izvršavaju ovisno o stanju parametra *state*. Ukoliko je parametar *state* postavljen u nulu, izvršava se dio koda koji prikazuje točno vrijeme i datum na zaslonu. Unutar petlje nalazi se dio koda koji provjerava je li korisnik poslao oznaku za početak primanja podataka s mobilnog uređaja. Ukoliko korisnik pošalje znak „!“ preko mobilnog uređaja, što mikrokontroler prima kao ASCII vrijednost 33, pokreće se funkcija za primanje podataka s mobilnog uređaja *BluetoothSetUp()*. Korisnik nakon unosa parametara za funkciju mora ponovno odaslati znak „!“ da bi se mikrokontroler mogao vratiti u funkciju provjere točnog vremena i datuma. Promjena stanja parametra *state* postiže se pritiskom na tipku SELECT koja se nalazi na izvodu 3 porta D. Zbog tromosti sustava potrebno je napraviti kašnjenje od 50 ms da bi se mogao očitati pozitivni brid na tom izvodu. Slika 4.6. prikazuje dio programskog koda za promjenu stanja parametra *state*, što označava prelazak u drugu korisničku funkciju [7].

```
current = PIND;                          //debounce loop for PIND3
if((previous &(1<<PIND3)) > (current &(1<<PIND3)))  //rise edge on PIND3
{
    state = 1;
}
_delay_ms(WAIT_MS);
previous = PIND;
```

Slika 4.6. Promjena vrijednosti parametra state [7]

Slika 4.7. prikazuje kompletni računalni kod za stanje parametra *state* = 0 u kojemu se nalaze funkcije *DateTime()* i *BluetoothSetUp() [7]*.

```
while (state == 0)
{
    uartValue = uart0_getc();

    if(uartValue == 33)                                          //znak '!'
    {
        varijable.functionFlag = 1;
    }
    while(varijable.functionFlag == 1)
    {
        BluetoothSetUp(&varijable, &korisnik);
    }


    DateTime(&vrijeme, &datum, &trajanje, &korisnik);

    current = PIND;                          //debounce loop for PIND3
    if((previous &(1<<PIND3)) > (current &(1<<PIND3)))  //rise edge on PIND3
    {
        state = 1;
    }
    _delay_ms(WAIT_MS);
    previous = PIND;
};
```

Slika 4.7. Programski kod za stanje parametra state = 0 [7]

Ukoliko se sustav nalazi u funkciji za prikaz vremena *DateTime()*, na zaslonu LCD-a prikazuje se točno vrijeme te datum koje senzor DS3231 šalje koristeći I2C komunikaciju mikrokontroleru. Rad I2C komunikacije definiran je u i2c.c datoteci. Pozivanjem struktura rtc_time i rtc_date iz rtc3231.h datoteke ostvaruje se pristup varijablama u kojima su sadržani podaci o točnom vremenu i datumu pojedinog trenutka prema kojima uspoređujemo vrijednosti unesene od strane korisnika. Kako bi se ostvario uspješan rad sustava za prikaz vremena i datuma potrebno je inicijalizirati rad LCD-a pozivajući funkciju *lcd_init(LCD_DISP_ON)* te *rtc3231_init()*. Za postavljanje vremena i datuma potrebno je izjednačiti pojedine varijable strukture s odabranim vrijednostima u heksadecimalnom obliku i nakon toga pozvati funkcije *rtc3231_write_time* i *rtc3231_write_date*. Ovaj korak potrebno je napraviti samo jednom, te kasnije izostaviti iz programskog koda, da kroz svaku iteraciju mikrokontroler ne bi zadržavao odabrano vrijeme. Korišenjem funkcije *sprintf* prebacujemo *int* format u kojemu su zapisane vrijednosti u *string* format prikladan ispisivanju na zaslon. Ispisivanje vrijednosti na zaslon postiže se pomoću *lcd_gotoxy(x,y)* funkcije koja postavlja pokazivač ispisa na određeni red i stupac ispisa i pomoću *lcd_puts(char *s)* funkcije koja ispisuje željene simbole zapisane u *string* formatu.

Na kraju funkcije *DateTime()* program ulazi u funkciju *OutputCheck()* kojoj se predaju sve potrebne informacije unesene od strane korisnika. To se postiže prosljeđivanjem adresa pokazivača pojedinih struktura koje sadrže potrebne varijable. *OutputCheck()* se poziva iz funkcije *DateTime()* kao osiguranje da je korisnik ispunio sve potrebne korake za postavljanje željenih parametara

paljenja i gašenja određenog releja. Funkcija *OutputCheck()* neće se pozivati niti iz jedne druge funkcije dok korisnik odabire željene parametre. Slika 4.8. prikazuje dio programskog koda za prikaz funkcije *DateTime()* [7].

```
void DateTime(struct unesenoVrijeme *vrijeme, struct uneseniDatum *datum, struct vrijemeTrajanja *trajanje, struct users *korisnik)  //funkcija za postavljanje vremena i datuma
{
    struct rtc_time time;
    struct rtc_date date;

    char datetimebuffer[12];    //buffer za vrijeme i datum

    lcd_init(LCD_DISP_ON);
    rtc3231_init();

    /* time.hour = 0x13;   //postavljanje vremena- samo jednom
    time.min = 0x36;   //sve vrijednosti se moraju unijeti u hex obliku
    time.sec = 0x00;

    date.day = 0x1E;   //postavljanje datuma- samo jednom
    date.month = 0xA;  //sve vrijednosti moraju se unijeti u hex obliku
    date.year = 0x12;
    //date.wday = 4; */

    //rtc3231_write_time(&time);  //postavljanje vremena- samo jednom
    //rtc3231_write_date(&date); //postavljanje datuma- samo jednom

    rtc3231_read_datetime(&time, &date);  //reading time and date from RTC_3231

    sprintf(datetimebuffer, "%02u:%02u:%02u", time.hour, time.min, time.sec);  //castanje int-a u string. %02u format stavlja leading zero na jednoznamenkaste brojeve
    lcd_gotoxy(0,0); // postavlja se pokaziva? na 1. red (row0) i 1. znak
    lcd_puts("TIME: ");
    lcd_gotoxy(6,0); // postavlja se pokaziva? na 2. red (row1) i 1. znak
    lcd_puts(datetimebuffer);

    sprintf(datetimebuffer, "%02u/%02u/20%02u", date.day, date.month, date.year);
    lcd_gotoxy(0,1);
    lcd_puts("DATE: ");
    lcd_gotoxy(6,1);
    lcd_puts(datetimebuffer);

    OutputCheck(vrijeme, datum, trajanje, korisnik);   //funkcija za omogucavanje izlaza
}
```

Slika 4.8. Programski kod funkcije DateTime() [7]

Ukoliko se stanje varijable *state* pritiskom na tipku SELECT koja se nalazi na pinu 3 porta D postavi u vrijednost „1“, sustav prelazi u sučelje za unos vremena paljenja određenog releja. To izvršava funkcija *UserDefinedTime().* Na početku funkcije definiramo stanja pojedinih tipki. Tipke su zapravo *bool* varijable koje mogu zaprimiti stanje 0 ili 1. Tipke su na početku postavljene u stanje 1, a pritiskom na pojedinu tipku poprimaju stanje 0. Definirano je pet tipki. Tipke LEFT, UP, DOWN, RIGHT, SELECT koje služe za navigiranje kroz varijable, te prijelazak u iduću funkciju. Tipka UP inkrementira odabranu varijablu za vrijednost 1, tipka DOWN dekrementira vrijednost varijable za 1, tipka LEFT prebacuje odabir vrijednosti na iduću varijablu, dok tipka RIGHT vraća odabir vrijednosti na prethodnu varijablu. Funkciji su predane strukture koje sadrže varijable za korisnički unesene vrijednosti vremena i datuma. Navigiranjem tipkama mijenjamo vrijednosti varijabli koje će se kasnije proslijediti funkciji *OutputCheck()* za usporedbu s vremenom i datumom koje šalje DS3231 senzor. Sve vrijednosti se prebacuju u *string* format i kao takve ispisuju se na LCD-u.

Ponovnim pritiskom na tipku SELECT mijenjamo vrijednost varijable *state* u „2“. Tada se izvršava *while* petlja koja sadrži funkciju za unos vremena gašenja odabranog releja *Duration()*. Kao i kod funkcije za unos vremena, najprije je potrebno definirati tipke s kojima korisnik može izmjenjivati željene vrijednosti varijabli. Vrijednosti varijabli ispisuju se na LCD-u.

Ukoliko korisnik još jednom pritisne tipku SELECT postavit će vrijednost varijable state na vrijednost „3“ što ujedno predstavlja i posljednji korak odabira željenog termina. Dok je *while* petlja u toj varijabli, izvršava se funkcija *RelayNumber()* u kojoj korisnik odabire koji relej želi koristiti iz 4-kanalnog relejnog modula. Korisnik može odabrati vrijednosti releja od 1 do 4. Slika 4.9. prikazuje programski kod funkcije za odabir releja [7]. Ponovnim pritiskom na tipku SELECT vraćamo se na početni zaslon tj. funkciju *DateTime()* preko koje možemo provjeravati podudaraju li se unesene vrijednosti varijabli s vrijednostima poslanim od strane DS3231 senzora.

```
void RelayNumber(struct vrijemeTrajanja *trajanje)
{
    struct Tipka tipka;
    char relayNumber[4];

    tipka.UP =  (PIND >> PD5)&1;
    tipka.DOWN = (PIND >> PD2)&1;

    lcd_init(LCD_DISP_ON);

    if(tipka.UP == 0)
    {
        if(trajanje->brojReleja <= 0x03)
            trajanje->brojReleja++;
        else
            trajanje->brojReleja = 0x01;
    }

    if(tipka.DOWN == 0)
    {
        if(trajanje->brojReleja >= 0x02)
        trajanje->brojReleja--;
        else
        trajanje->brojReleja = 0x04;
    }

    itoa(trajanje->brojReleja, relayNumber, 10);

    lcd_gotoxy(0,0); // postavlja se pokaziva? na 1. red (row0) i 1. znak
    lcd_puts("RELAY NUMBER: ");
    lcd_gotoxy(14,0); // postavlja se pokaziva? na 2. red (row1) i 1. znak
    lcd_puts(relayNumber);
}
```

Slika 4.9. Programski kod funkcije RelayNumber() [7]

4.4. Mobilni dio

Želi li korisnik unijeti varijable preko mobilnog uređaja koristeći neku od aplikacija koje podržavaju *Bluetooth* terminal, potrebno je prvo odaslati znak "!" koji označava početak transmisije. Nakon primitka tog simbola mikrokontroler prebacuje način rada u funkciju za primitak i rad s informacijama kao što je prikazano na slici 4.7. Nakon simbola „!" korisnik treba unijeti znak „ ; " preko kojeg označava da se radi o novom unosu vremena tj. novom korisniku. Slijedi unos korisničkog imena, te vremena paljenja, gašenja i odabranog releja. Zadnji korak je ponovni unos simbola „!" što mikrokontroleru daje naredbu da se vrati u jezgreni način rada. Primjer: !; Ime, HH:MM:SS HH:MM:SS R!. Iz navedenog primjera možemo vidjeti simbole potrebne za ispravan

unos varijabli. Varijabla „Ime“ predstavlja korisničko ime koje će korisnik odabrati, prvi set znamenaka HH:MM:SS predstavlja vrijeme paljenja određenog releja u formatu sat/minuta/sekunda. Brojčane vrijednosti moraju se odvajati simbolom „ : “. Sljedeći set vrijednosti HH:MM:SS predstavlja vrijeme gašenja sustava, a varijabla „R“ prestavlja broj odabranog releja“.

Nakon što programski kod uđe u funkciju za mobilni unos podataka, mobilna primopredajna antena prenosi podatke s mobilnog uređaja prema mikrokontroleru UART serijskom komunikacijom. Mikrokontroler prima ili odašilje podatke koristeći *uart.c* biblioteku. Primopredajna antena podešena je na rad s 9600 bauda tako da je potrebno i inicijalizirati komunikacijom uart0_init() funkcijom te istu podesiti na jednak broj bauda. Nakon toga potrebno je izračunati *prescaler* vrijednost da se brzina podataka prilagodi frekvenciji mikroupravljača. To se postiže korištenjem programskog koda prikazanog na slici 4.10 gdje varijabla „baud“ označava vrijednost bauda na koju je postavljena primopredajna antena. [11]

```
uint16_t baudPrescaler;
baudPrescaler = (F_CPU/(16UL*baud))-1;
```

Slika 4.10. Računanje baud prescalera [11]

Mikrokontroler razlikuje odaslane varijable korištenjem simbola razmaka, što u ASCII kodu poistovjećuje s vrijednosti 32. Ukoliko mikrokontroler zaprimi tu vrijednost, zna da je potrebno prebaciti primljene podatke u drugi set vrijednosti. Npr. korištenjem razmaka mikrokontroler odvaja brojevne vrijednosti paljenja sustava od brojevnih vrijednosti gašenja sustava.

Mobilni uređaj odašilje podatke prema mikrokontroleru u ASCII zapisu, a da bi se moglo raditi sa znamenkama potrebno je svaku pojedinu znamenku oduzeti s brojem 48 što u ASCII zapisu predstavlja broj „0“. Budući da je većina znamenki dvoznamenkasta, a mobilnim uređajem možemo slati samo jedan po jedan simbol, potrebno je spojiti dva jednoznamenkasta broja u cjelinu. To se postiže funkcijom *concatenate* koja prvi primljeni simbol množi s brojem 10 i na taj broj dodaje drugi primljeni simbol. Slika 4.11. prikazuje programski kod *concatenate* funkcije [7] .

```
unsigned concatenate(uint8_t x, uint8_t y)
 {
    uint8_t pow = 10;
    while(y >= pow)
    pow *= 10;
    return x*pow + y;
}
```

Slika 4.11. Concatenate funkcija [7]

Iznosi li primljena ASCII vrijednost 58 što predstavlja simbol „:“, programski kod prebacuje unos u nove varijable i pomoću *goto* funkcije odlazi na početak izvršavanja petlje. Pomoću funkcije *isalpha* provjeravamo predstavlja li unesena vrijednost simbol ili broj. Ukoliko je u pitanju simbol, pomoću funkcije *sprintf* prebacujemo vrijednost u char format, te pomoću funkcije *strcat* dodjeljujemo nove simbole prethodim kako bi stvorili riječ. Oformirana riječ predstavlja korisničko ime uneseno od strane korisnika.

4.5. Sustav dodjele prioriteta

Mobilni sustav dizajniran je tako da može zaprimiti unose do 20 korisnika. Svakim novim unosom koji se ostvaruje slanjem simbola „ ; “, povećava se brojač koji broji trenutni broj korisnika u memoriji. Sustav dodjele prioriteta nalazi se u datoteci *bluetooth.c.* On se sastoji od jedne *for* petlje koja inkrementira brojač sve dok ne dođe do vrijednosti broja korisnika umanjene za jednu jedinicu. Tada uspoređuje korisničke unose broja brojača *for* petlje s posljednjim unesenim vrijednostima. Ukoliko se vremena paljenja i broj releja podudaraju s nekim od prethodnih unosa, tada sustav pošalje poruku na korisnikov mobilni uređaj da je došlo do poklapanja s već prethodnim unesenim vremenom paljenja i brojem releja te ispisuje ime korisnika koji je zauzeo taj termin. Važno je naglasiti da će se sustav aktivirati samo ako se podudaraju i vrijeme paljenja i broj releja korisnika. Ukoliko je broj releja drugačiji, sustav dodjele prioriteta neće proraditi nego će unijeti novog korisnika. Dogodi li se poklapanje, sustav automatski briše korisničko ime i sve unesene varijable te na kraju dekrementira broj korisnika za jedno mjesto. Slika 4.12. prikazuje programski kod koji radi dodjelu prioriteta [7].

```
for(uint8_t i = 0; i< korisnik->brojKorisnika; i++)
{
    if(korisnik->brojKorisnika != 0)
    {
        if((korisnik->openingHour[i] == korisnik->openingHour[korisnik->brojKorisnika])
         && (korisnik->openingMin[i] == korisnik->openingMin[korisnik->brojKorisnika])
         && (korisnik->openingSec[i] == korisnik->openingSec[korisnik->brojKorisnika])
         && (korisnik->relayNumber[i] == korisnik->relayNumber[korisnik->brojKorisnika]))
        {
            memset(korisnik->userName[korisnik->brojKorisnika],0, sizeof(korisnik->userName[korisnik->brojKorisnika]));
            korisnik->openingHour[korisnik->brojKorisnika] = 0;
            korisnik->openingMin[korisnik->brojKorisnika] = 0;
            korisnik->openingSec[korisnik->brojKorisnika] = 0;
            korisnik->relayNumber[korisnik->brojKorisnika] = 0;
            korisnik->brojKorisnika--;
            uart0_puts("WARNING! Input time and relay are unavailable! User that has that period is ");
            uart0_puts(korisnik->userName[i]);
        }
    }
}
```

Slika 4.12. Dodjela prioriteta [7]

4.6. Usporedba vrijednosti s stvarnim vremenom

Završetak sustava predstavlja funkcija za uspoređivanje vrijednosti unesenih od strane korisnika s vrijednostima varijabli primljenih od DS3231 senzora. To u sustavu rješava funkcija *OutputCheck().* Ona radi odvojenu usporedbu jezgrenog dijela s vrijednostima senzora i mobilnog dijela s istim. Kako jezgrenim dijelom upravlja samo jedan korisnik, provjera se odvija samo jednom *if-else* funkcijom. Funkcija provjerava jednakost trenutnih sati, minuta i sekunda s unesenim vrijednostima. Poklapaju li se vrijednosti, provjerava se koji relej je odabran i sa *switch-case* funkcijom daje se naredba za paljenje određenog releja. Istim načinom provjerava se jednakost vrijednosti varijabli za gašenje i se daje naredba gašenja određenog releja.

Provjera mobilnog dijela slična je provjeri jezgrenog dijela, osim što mobilni dio zahtjeva i jednu *for* petlju koja će provjeravati sve unesene korisničke vrijednosti varijabli. Pojavi li se podudaranje na bilo kojoj iteraciji petlje, funkcija će dati naredbu za paljenjem određenog releja. Dok se pojavi podudaranje gašenja određenog releja, funkcija će uz gašenje releja također obrisati sve varijable tog korisnika iz memorije. Slika 4.13 prikazuje usporedbu i naredbe za paljenje jezgrenog dijela, dok slika 4.14. prikazuje usporedbu i paljenje sustava preko mobilnog dijela [7].

```
if(date.year == datum->year && date.month == datum->month
&& date.day == datum->day && time.hour == vrijeme->hour
&& time.min == vrijeme->min &&  time.sec == vrijeme->sec) //provjera postavljenog vremena
{
    trajanje->stanjeZaOutputCheck = 1;
    switch(trajanje->brojReleja)
    {
        case 1:
        trajanje->relay1 = 1;
        break;
        case 2:
        trajanje->relay2 = 1;
        break;
        case 3:
        trajanje->relay3 = 1;
        break;
        case 4:
        trajanje->relay4 = 1;
        break;
    }
}

if(time.hour == trajanje->hour && time.min == trajanje->min
&&  time.sec == trajanje->sec) // provjera vremena zavrsetka
{
    trajanje->stanjeZaOutputCheck = 0;
    switch(trajanje->brojReleja)
    {
        case 1:
        trajanje->relay1 = 0;
        break;
        case 2:
        trajanje->relay2 = 0;
        break;
        case 3:
        trajanje->relay3 = 0;
        break;
        case 4:
        trajanje->relay4 = 0;
        break;
    }
}
```

Slika 4.13. Provjera jezgrenog dijela [7]

```
for(uint8_t i = 0; i <= korisnik->brojKorisnika; i++)
{
    if(time.hour == korisnik->openingHour[i] && time.min == korisnik->openingMin[i] &&  time.sec == korisnik->openingSec[i])
    {
        switch(korisnik->relayNumber[i])
        {
            case 1:
                    trajanje->relay1 = 1;
            break;
            case 2:
                    trajanje->relay2 = 1;
            break;
            case 3:
                    trajanje->relay3 = 1;
            break;
            case 4:
                    trajanje->relay4 = 1;
            break;
        }
    }
    if(time.hour == korisnik->closingHour[i] && time.min == korisnik->closingMin[i] &&  time.sec == korisnik->closingSec[i])
    {
        switch(korisnik->relayNumber[i])
        {
            case 1:
            trajanje->relay1 = 0;
            break;
            case 2:
            trajanje->relay2 = 0;
            break;
            case 3:
            trajanje->relay3 = 0;
            break;
            case 4:
            trajanje->relay4 = 0;
            break;
        }
        memset(korisnik->userName[i],0, sizeof(korisnik->userName[i]));
        korisnik->openingHour[i] = 0;
        korisnik->openingMin[i] = 0;
        korisnik->openingSec[i]= 0;
        korisnik->closingHour[i]=0;
        korisnik->closingMin[i] = 0;
        korisnik->closingSec[i]= 0;
        korisnik->relayNumber[i]= 0;
    }
}
```

Slika 4.14. Provjera mobilnog dijela[7]

Slika 4.15. prikazuje naredbe paljenja i gašenja određenih relejnih jedinica [7].

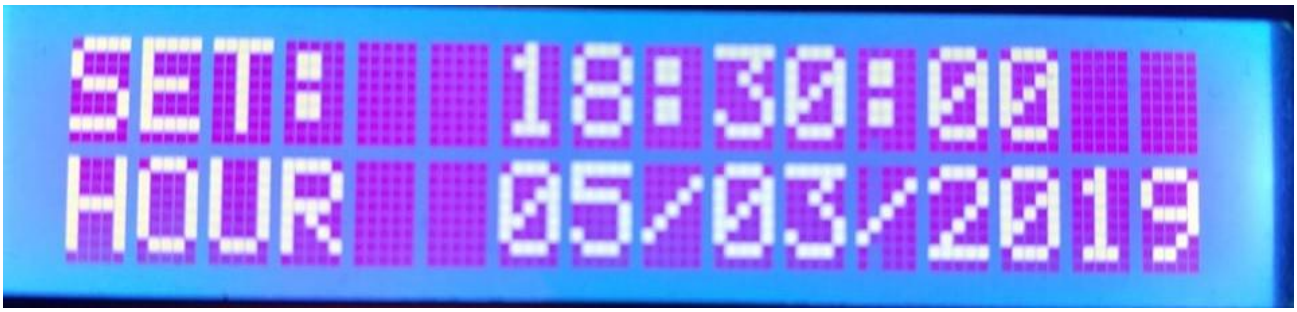
```
if(trajanje->relay1 == 1)
{
    PORTA &=~ (1<<PA4);                 //RELAY 1
}
else
{
    PORTA |= (1<<PA4);
}

if(trajanje->relay2 == 1)
{
    PORTA &=~ (1<<PA5);                 //RELAY 2
}
else
{
    PORTA |= (1<<PA5);
}

if(trajanje->relay3 == 1)
{
    PORTA &=~ (1<<PA6);                 //RELAY 3
}
else
{
    PORTA |= (1<<PA6);
}

if(trajanje->relay4 == 1)
{
    PORTA &=~ (1<<PA7);                 //RELAY 4
}
else
{
    PORTA |= (1<<PA7);
}
```

Slika 4.15. Naredbe za paljenje i gašenje releja [7]

5. TESTIRANJE RADA PAMETNE UTIČNICE

5.1. Upravljanje mikrokontrolerom

Pritiskom na tipku SELECT ulazimo u funkciju odabira vremena paljenja sustava. Pritiskom na tipke UP ili DOWN mijenjamo vrijednost varijabli. Mijenjamo li vrijednost varijable „HOUR“, pritiskom na tipku UP povećavamo vrijednost sve do vrijednosti „23“. Daljnjim povećavanjem vrijednosti dolazimo do vremena „00“ te ponovno započinjemo novi ciklus vrijednosti. Tipkama LEFT i RIGHT mijenjamo parametre sustava. Pokazivač se prebacuje na mijenjanje varijabli MIN pritisne li korisnik tipku RIGHT. U tom slučaju na ekranu se ispisuje ime varijable koja se mijenja. Slika 5.1. prikazuje izgled sučelja za odabir vremena otvaranja sustava korištenjem tipkala [7].



Slika 5.1. Sučelje za odabir vremena paljenja [7]

Daljnjim pritiskom tipke SELECT ulazimo u funkciju odabira vremena paljenja. Koncept rada je isti kao i u prethodnoj funkciji. Slika 5.2. prikazuje izgled sučelja za odabir vremena gašenja određenog releja korištenjem tipkala [7].

Slika 5.2. Sučelje za odabir vremena gašenja [7]

Još jednim pritiskom tipke SELECT korisnik ulazi u funkciju odabira releja. Korisnik može odabrati između četiri vrijednosti odabira. Na slici 5.3. prikazan je izgled sučelja odabira broja releja korištenjem tipkala [7].

Slika 5.3. Sučelje za odabir releja [7]

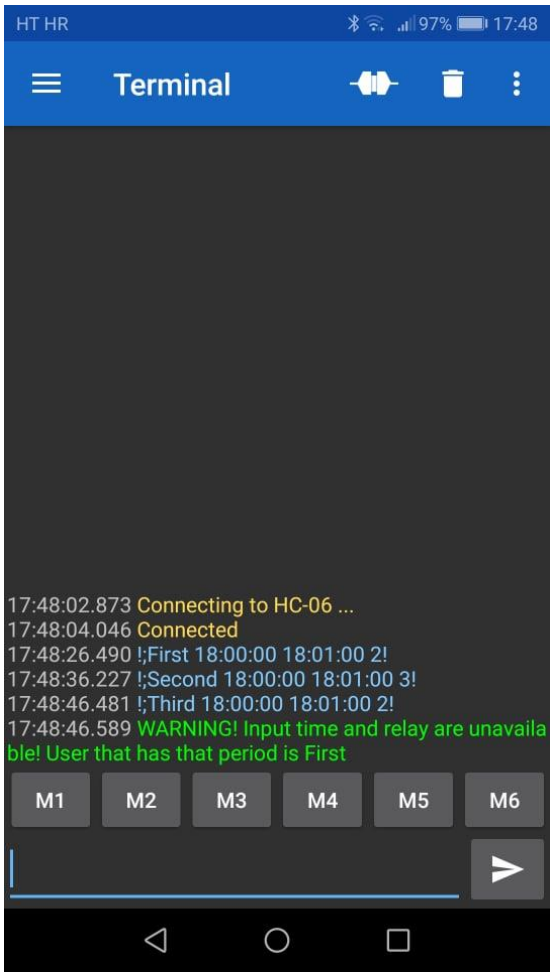
Da bi se proces odabira vremena korištenja releja završio, potrebno je još jednom pritisnuti tipku SELECT da se sustav nađe u početnoj funkciji prikaza vremena i datuma, koja poziva funkciju provjere stanja *OutputCheck()*. Slika 5.4. prikazuje izgled sučelja početne funkcije za prikaz vremena i datuma. [7]

Slika 5.4. Početno sučelje za prikaz vremena i datuma [7]

Ukoliko su svi koraci zadovoljeni, u zadano vrijeme upalit će se određeni relej i strujni krug držat će se otvorenim sve dok se vrijeme gašenja ne izjednači s trenutnim vremenom očitanim sa senzora DS3231.

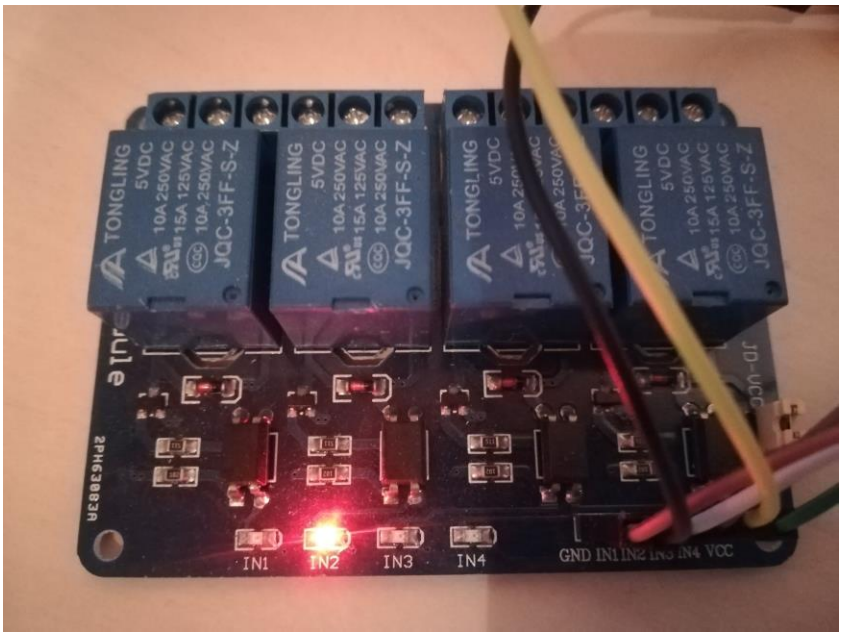
5.2. Upravljanje mobilnim uređajem

Za upravljanje mobilnim uređajem korisnik mora skinuti jednu od aplikacija koje podržavaju rad s *Bluetooth* terminalom. Korisnik treba imati upaljen *Bluetooth* modul na svome mobilnom uređaju i spojiti se s HC-06 uređajem. Lozinka za spajanje s primopredajnikom HC-06 je „1234“. Nakon spajanja s uređajem korisnik treba varijable za rad sustava. Odabere li korisnik vrijednosti koje su već zauzete, na svoj mobilni uređaj primit će upozorenje i ime korisnika koji je zauzeo određeni termin. Slika 5.5. prikazuje primjer unosa varijabli za tri korisnika, kod kojeg je treći korisnik odabrao isti termin kao i prvi korisnik [7].



Slika 5.5. Upravljanje sustava mobilnim uređajem [7]

Odabrani relej primit će signal za otvaranje strujnog kruga kao što je prikazano na slici 5.6. ako su zadovoljeni svi koraci.



Slika 5.6. Signal otvaranja releja broj 2 [7]

5.3. Uočeni problemi sustava

Pri radu na sustavu uočeni su određeni problemi. Glavni problem je u tome što mikrokontroler zbog ograničenja jednojezgrene arhitekture ne može provoditi jezgreni i mobilni dio istovremeno. U ovom sustavu to je riješeno korištenjem jednog simbola koji će označiti početak i kraj transmisije, tako da se mikrokontroler može prebaciti iz jedne funkcije u drugu, premda ih nikada neće istovremeno vrtiti. Korištenjem arhitekture s više procesorskih jezgri navedeni problem mogao bi se riješiti korištenjem više *threadova* koji će paralelno vrtiti funkcije.

Idući problem pojavljuje se pri dolasku naredbe otvaranja releja na više jedinica. U tom slučaju LED će zasvijetliti na odabranim relejnim jedinicama, ali se one neće otvoriti zbog niskog strujnog ograničenja struje pogonjene iz USB porta. Rješenje tog problema jest odvajanje napajanja za mikrokontroler i relejni modul. Preporuča se korištenje ispravljača s 220 V izmjeničnog napona na 5 V istosmjernog napona direktno napajanog iz gradske mreže.

6. ZAKLJUČAK

Diplomskim radom ostvaren je sustav dodjele prioriteta sustava korisnika koji se može koristiti u kućanstvu ili bilo kojim drugim javnim ili poslovnim prostorima.

Za relativno nisku cijenu izrade dobije se kvalitetan sustav za korištenje uređaja napajanih izmjeničnom strujom gradske mreže. Korisnicima se daje na izbor korištenje jezgrenog modula, gdje na lokaciji sustava korištenjem tipkala mogu unijeti termin korištenja određenog uređaja napajanog preko jednog od relejnih jedinica uređaja. Isto tako, korisnicima se daje mogućnost određivanja termina preko mobilnog uređaja korištenjem *Bluetooth* tehnologije. Korištenjem te mogućnosti korisnik ne mora biti blizu jedinice sustava već mora biti u dometu *Bluetooth* primopredajne antene. Pri kraju termina određenog korisnika, sustav će obrisati vrijednosti iz memorije zauzete za varijable tog korisnika. Pokuša li više korisnika zauzeti isti termin s istom relejnom jedinicom, korisnik će na mobilni uređaj dobiti upozorenje da je sustav prethodno zauzet te će mu se prikazati ime korisnika koji je prvi zauzeo taj termin, tj. ima prioritet nad tim terminom.

Pogodnost ovog rada je što se za relativno nisku financijsku investiciju može korisnicima omogućiti daljinsko upravljanje uređajima te im tako olakšati svakodnevne aktivnosti.

Ograničanje ovog rada je u tomu što korisnici moraju biti u dometu primopredajne *Bluetooth* antene kako bi mogli komunicirati s mikrokontrolerom koristeći mobilne uređaje. Rad bi se mogao unaprijediti korištenjem WLAN (*engl. Wireless Local Area Network*) tehnologije, koja bi omogućila povezivanje korisnika s mikrokontrolerom što bi korisnik mogao napraviti neovisno o svojoj lokaciji.

LITERATURA

[1]. https://electronics.howstuffworks.com/microcontroller1.htm, 20.6.2018.

[2]. https://informatika.buzdo.com/s882-mikrokontroler.htm, 20.6.2018.

[3]. 8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash ATmega16, ATmega16L Preliminary Datasheet, © Atmel Corporation, 2002.

[4]. https://www.amazon.com/ATMEGA16-Minimum-System-ATmega32-Programmer/dp/B07L8L5M23, 1.3.2019.

[5]. https://potentiallabs.com/cart/relay-board-4-channel, 1.3.2019.

[6]. https://e-radionica.com/hr/rtc-modul-ds3231.html, 1.3.2019.

[7]. Vlastita izrada.

[8]. Vježba 5. Kristalni oscilator i Biblioteke: LCD i WDT, Primjena Mikroupravljačkih Sustava – laboratorijske vježbe, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Zavod za komunikacije.

[9]. https://www.microchip.com/mplab/avr-support/atmel-studio-7, 4.3.2019.

[10]. Vježba 1. Uvod u mikroupravljače i ulazno-izlazne priključke, Primjena Mikroupravljačkih Sustava – laboratorijske vježbe, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Zavod za komunikacije.

[11]. Vježba 2. USART, Primjena Mikroupravljačkih Sustava – laboratorijske vježbe, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Zavod za komunikacije.

SAŽETAK

Sustav dodjele prioriteta pametne utičnice je sustav u kojemu mikroupravljač Atmel ATmega16 upravlja relejnim modulom za otvaranje ili zatvaranje izmjeničnog strujnog kruga ovisno o varijablama unešenim od strane korisnika. Mikrokontroler može raditi u jezgrenom načinu rada pri kojem korisnik upravlja sustavom pomoći tipki direktno spojenim na mikrokontroler i u mobilnom načinu rada gdje korisnik može upravljati sustavom svojim mobilnim uređajem koristeći *Bluetooth* tehnologiju.

Dodjela prioriteta zamišljena je na način da korisnik koji prvi odabere željeni termin i relejnu jedinicu ima prioritet nad tom jedinicom u tom terminu. Želi li se drugi korisnik prijaviti u tom terminu, sustav odašilje poruku na mobilni uređaj korisnika da je termin prethodno zauzet i ispisuje ime korisnika koji ga je zauzeo.

Ključne riječi: dodjela, prioriteta, pametna, utičnica, mikrokontroler, mobitel, Bluetooth, senzor, Atmega, Atmel.

SMARTPLUG PRIORITY ASSIGNMENT SYSTEM

ABSTRACT

Smartplug priority assignment system is designed as a device in which microcontroller Atmel ATmega16 controls a relay module for opening and closing alternating current circuits depending on the commands entered by multiple users. The microcontroller has 2 modes of operation. The first mode, called „Core Mode“, allows the user to enter the commands using pushbuttons near the microcontroller. The second mode of operation, called „Mobile mode“, allows the user to enter the commands by using a personal mobile device via *Bluetooth* technology.

Priority allocation means that the first user who enters his opening period and chooses a relay number gets priority during that period for that relay. If any other user tries to connect during that period, the system will send a message to his mobile device via *Bluetooth* that says that that period is already occupied and it will display the name of the user that has priority during that period.

Keywords: allocation, priority, system, plug, microcontroller, mobile, Bluetooth, sensor, ATmega, Atmel.

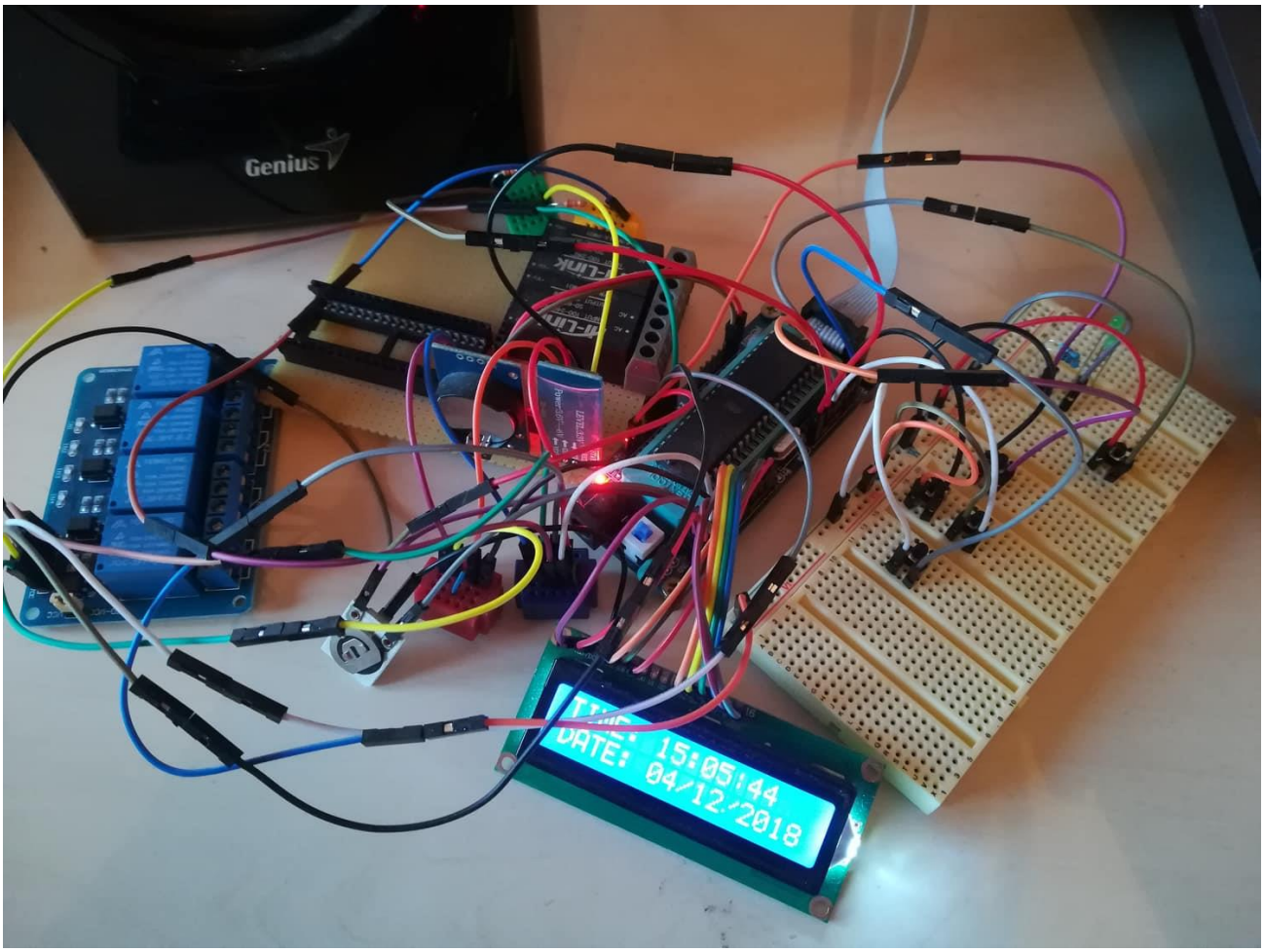
ŽIVOTOPIS

Domagoj Križanec rođen je 10.12.1993. u Osijeku. Trenutačno prebiva u Ladimirevcima gdje je i završio osnovnu školu. 2008. godine obrazovanje nastavlja u „Srednjoj Školi Valpovo“, elektrotehnički smjer. Fakultet elektrotehnike, računarstva i informacijskih tehnologija (tadašnji ETFOS) sveučilišta Josipa Jurja Strossmayera u Osijeku upisuje 2012. godine. Preddiplomski studij elektrotehnike završava 2016. godine sa završnim radom „Pregled baza komprimiranih video signala sa subjektivnim rezultatima ocjene kvalitete“ i iste godine upisuje diplomski studij elektrotehnike, smjera „Komunikacije i informatika“ i odjela „DKA – Komunikacijske tehnologije“. Stručnu praksu odrađuje u tvrtki „TEO Belišće d.o.o.“ na poziciji „Programer za puštanje strojeva u rad“.

PRILOZI

SLIKE

Slika 1. prikazuje izgled kompletnog sklopa sa svim hardverskim komponentama.



Slika 1. Izgled kompletnog sklopa [7]

PROGRAMSKI KOD

main.c

```
*
 * Test01.c
 *
 * Created: 08-Oct-18 21:23:12
 * Author : Domagoj Krizanec
 */

#include <avr/io.h>
#include <stdlib.h>
#include "button.h"


int main (void)
{
	ButtonInitialisation(); //pozivanje funkcije tipka s postavljenim vrijednostima
	ButtonPress();     //rad centralnog sustava
}
```

button.c

```
/*
 * button.c
 *
 * Created: 11-Oct-18 23:29:46
 *  Author: Domagoj Krizanec
 */

#include <avr/io.h>
#include <util/delay.h>
#include <string.h>
#include "button.h"
#include "display.h"
#include "rtc3231.h"
#include "lcd.h"
#include "bluetooth.h"
#include "uart.h"
#define WAIT_MS 50 //vrijeme cekanja u ms

void ButtonInitialisation()
{
	DDRD = 0x00; // svi pinovi u PORTD stavljeni kao input - pocetno stanje
	DDRD &=~ (1<<PD2)|(1<<PD3)|(1<<PD4)|(1<<PD5)|(1<<PD6); // input mode na pinovima PD2-
PD6
	DDRA |= (1<<PA1)|(1<<PA4)|(1<<PA5)|(1<<PA6)|(1<<PA7); // output mode na pinu PA1,
PA4, PA5, PA6, PA7
	PORTD = 0x00; //onemoguceni pull-up otpornici za sve PIN-ove porta D
	PORTD |= (1<<PD2)|(1<<PD3)|(1<<PD4)|(1<<PD5)|(1<<PD6); // stavljanje 1 na tipke PD2-
PD6
}
```

```
void ButtonPress()
{
		int state = 0;
		uint8_t current = PIND;
		uint8_t previous = PIND;
		uint8_t uartValue = 0;
		uint16_t baudPrescaler, baud = 9600;
		//definirana vrijednost baud ratea
		baudPrescaler = (F_CPU/(16UL*baud))-1;

		struct rtc_date date;
		struct unesenoVrijeme vrijeme;
		struct uneseniDatum datum;
		struct vrijemeTrajanja trajanje;
		struct info varijable;
		struct users korisnik;

		rtc3231_read_date(&date);

		vrijeme.horizontalno = 0;
		vrijeme.hour = 0xC;   //postavljanje vremena unosa- samo jednom
		vrijeme.min = 0x1E;   //sve vrijednosti se moraju unijeti u hex obliku
		vrijeme.sec = 0x00;

		datum.day = date.day;    //0xF;   //postavljanje datuma- samo jednom
		datum.month = date.month;  //0x06;  //sve vrijednosti moraju se unijeti u hex
obliku
		datum.year =  date.year;  //0x12;

		trajanje.trajanjeHorizontalno = 0; //postavljanje pocetnog stanja za
horizontalni dio displaya trajanja
		trajanje.stanjeZaOutputCheck = 0; //postavljanje pocetnog stanja za flag
stanja
		trajanje.brojReleja = 0x01; //postavljanje broja releja na pocetnu vrijednost
		trajanje.hour = 0xC;   //postavljanje vremena gasenja- samo jednom
		trajanje.min = 0x1E;   //sve vrijednosti se moraju unijeti u hex obliku
		trajanje.sec = 0x00;

		varijable.concatenatedNumber = 0;
		varijable.uneseneZnamenke = 0;
		varijable.unosVrijednostiKorisnika = 0;
		varijable.previousUart = 0;
		varijable.uartRecieve = 0;
		varijable.testI = 0;
		varijable.functionFlag = 0;
		memset(varijable.FinalUart, 0, sizeof(varijable.FinalUart));  //postavljanje
parametra Final Uart u Nulu
		memset(varijable.UART, 0, sizeof(varijable.UART));

		korisnik.brojKorisnika = 0;
		uart0_init(baudPrescaler);

	while(1)
	{
		while (state == 0)
		{
			uartValue = uart0_getc();
```

```
                        if(uartValue == 33)
                           //znak '!'
                        {
                               varijable.functionFlag = 1;
                        }
                        while(varijable.functionFlag == 1)
                        {
                               BluetoothSetUp(&varijable, &korisnik);
                        }


                        DateTime(&vrijeme, &datum, &trajanje, &korisnik);

                        current = PIND;                                                 //debounce
loop for PIND3
                        if((previous &(1<<PIND3)) > (current &(1<<PIND3)))  //rise edge on PIND3
                        {
                               state = 1;
                        }
                        _delay_ms(WAIT_MS);
                        previous = PIND;

                };

                while(state == 1)
                {
                        uartValue = uart0_getc();
                        if(uartValue == 33)
                           //znak '!'
                        {
                               varijable.functionFlag = 1;
                        }
                        while(varijable.functionFlag == 1)
                        {
                               BluetoothSetUp(&varijable, &korisnik);
                        }

                        UserDefinedTime(&vrijeme, &datum);
                        current = PIND;
                        if((previous &(1<<PIND3)) > (current &(1<<PIND3)))  //rise edge on PIND3
                        {
                               state = 2;
                        }
                        _delay_ms(WAIT_MS);
                        previous = PIND;

                };

                while(state == 2)
                {
                        uartValue = uart0_getc();
                        if(uartValue == 33)
                           //znak '!'
                        {
                               varijable.functionFlag = 1;
                        }
                        while(varijable.functionFlag == 1)
                        {
```

```
                        BluetoothSetUp(&varijable, &korisnik);
                }

                Duration(&trajanje);
                current = PIND;
                if((previous &(1<<PIND3)) > (current &(1<<PIND3)))  //rise edge on PIND3
                {
                        state = 3;
                }
                _delay_ms(WAIT_MS);
                previous = PIND;
        };

        while(state == 3)
        {
                uartValue = uart0_getc();
                if(uartValue == 33)
                   //znak '!'
                {
                        varijable.functionFlag = 1;
                }
                while(varijable.functionFlag == 1)
                {
                        BluetoothSetUp(&varijable, &korisnik);
                }

                RelayNumber(&trajanje);
                current = PIND;
                if((previous &(1<<PIND3)) > (current &(1<<PIND3)))  //rise edge on PIND3
                {
                        state = 0;
                }
                _delay_ms(WAIT_MS);
                previous = PIND;
        };


    }
}

void OutputCheck(struct unesenoVrijeme *vrijeme, struct uneseniDatum *datum, struct
vrijemeTrajanja *trajanje , struct users *korisnik)  //funkcija za upravljanjem izlazom
{
    struct rtc_time time;
    struct rtc_date date;

    rtc3231_init();
    rtc3231_read_datetime(&time, &date);


    if(date.year == datum->year && date.month == datum->month
    && date.day == datum->day && time.hour == vrijeme->hour
    && time.min == vrijeme->min &&  time.sec == vrijeme->sec) //provjera postavljenog
vremena
    {
        trajanje->stanjeZaOutputCheck = 1;
        switch(trajanje->brojReleja)
        {
```

```
                        case 1:
                        trajanje->relay1 = 1;
                        break;
                        case 2:
                        trajanje->relay2 = 1;
                        break;
                        case 3:
                        trajanje->relay3 = 1;
                        break;
                        case 4:
                        trajanje->relay4 = 1;
                        break;
                }
        }

        if(time.hour == trajanje->hour && time.min == trajanje->min
        &&  time.sec == trajanje->sec) // provjera vremena zavrsetka
        {
                trajanje->stanjeZaOutputCheck = 0;
                switch(trajanje->brojReleja)
                {
                        case 1:
                        trajanje->relay1 = 0;
                        break;
                        case 2:
                        trajanje->relay2 = 0;
                        break;
                        case 3:
                        trajanje->relay3 = 0;
                        break;
                        case 4:
                        trajanje->relay4 = 0;
                        break;
                }
        }

        for(uint8_t i = 0; i <= korisnik->brojKorisnika; i++)
        {
                if(time.hour == korisnik->openingHour[i] && time.min == korisnik-
>openingMin[i] &&  time.sec == korisnik->openingSec[i]) //postavljenog vremena
                {
                        switch(korisnik->relayNumber[i])
                        {
                                case 1:
                                                trajanje->relay1 = 1;
                                break;
                                case 2:
                                                trajanje->relay2 = 1;
                                break;
                                case 3:
                                                trajanje->relay3 = 1;
                                break;
                                case 4:
                                                trajanje->relay4 = 1;
                                break;
                        }
                }
```

```
			if(time.hour == korisnik->closingHour[i] && time.min == korisnik-
>closingMin[i] &&  time.sec == korisnik->closingSec[i])
			{
				switch(korisnik->relayNumber[i])
				{
					case 1:
					trajanje->relay1 = 0;
					break;
					case 2:
					trajanje->relay2 = 0;
					break;
					case 3:
					trajanje->relay3 = 0;
					break;
					case 4:
					trajanje->relay4 = 0;
					break;
				}
				memset(korisnik->userName[i],0, sizeof(korisnik->userName[i]));
				korisnik->openingHour[i] = 0;
				korisnik->openingMin[i] = 0;
				korisnik->openingSec[i]= 0;
				korisnik->closingHour[i]=0;
				korisnik->closingMin[i] = 0;
				korisnik->closingSec[i]= 0;
				korisnik->relayNumber[i]= 0;
			}
		}

		if(trajanje->relay1 == 1)
		{
			PORTA &=~ (1<<PA4);                              //RELAY 1
		}
		else
		{
			PORTA |= (1<<PA4);
		}

		if(trajanje->relay2 == 1)
		{
			PORTA &=~ (1<<PA5);                              //RELAY 2
		}
		else
		{
			PORTA |= (1<<PA5);
		}

		if(trajanje->relay3 == 1)
		{
			PORTA &=~ (1<<PA6);                              //RELAY 3
		}
		else
		{
			PORTA |= (1<<PA6);
		}

		if(trajanje->relay4 == 1)
		{
```

```
		PORTA &=~ (1<<PA7);				//RELAY 4
	}
	else
	{
		PORTA |= (1<<PA7);
	}

	if(trajanje->stanjeZaOutputCheck == 1) PORTA |= (1<<PA1);
	else PORTA &=~ (1<<PA1);
}
```

button.h

```
*
 * button.h
 *
 * Created: 11-Oct-18 23:32:51
 *  Author: Domagoj Krizanec
 */
#ifndef TIPKE
#define TIPKE

  struct Tipka
{
	_Bool LEFT;
	_Bool UP;
	_Bool DOWN;
	_Bool RIGHT;
	_Bool SELECT;

};

struct unesenoVrijeme;
struct uneseniDatum;
struct vrijemeTrajanja;
struct users;


void ButtonInitialisation();
void ButtonPress();
void OutputCheck(struct unesenoVrijeme *vrijeme, struct uneseniDatum *datum, struct
vrijemeTrajanja *trajanje , struct users *korisnik);

#endif
```

display.c

```
/*
 * display.c
 *
 * Created: 10-Oct-18 13:24:58
 *  Author: Domagoj Krizanec
 */

#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
```

```
#include "lcd.h"
#include "rtc3231.h"
#include "button.h"
#include "display.h"

void DateTime(struct unesenoVrijeme *vrijeme, struct uneseniDatum *datum, struct
vrijemeTrajanja *trajanje, struct users *korisnik)  //funkcija za postavljanje vremena i
datuma
{
	struct rtc_time time;
	struct rtc_date date;

	char datetimebuffer[12];    //buffer za vrijeme i datum

	lcd_init(LCD_DISP_ON);
	rtc3231_init();

	/* time.hour = 0x13;   //postavljanje vremena- samo jednom
	time.min = 0x36;   //sve vrijednosti se moraju unijeti u hex obliku
	time.sec = 0x00;

	date.day = 0x1E;   //postavljanje datuma- samo jednom
	date.month = 0xA;  //sve vrijednosti moraju se unijeti u hex obliku
	date.year = 0x12;
	//date.wday = 4; */

	//rtc3231_write_time(&time);  //postavljanje vremena- samo jednom
	//rtc3231_write_date(&date); //postavljanje datuma- samo jednom

	rtc3231_read_datetime(&time, &date);  //reading time and date from RTC_3231

	sprintf(datetimebuffer, "%02u:%02u:%02u", time.hour, time.min, time.sec);  //castanje
int-a u string. %02u format stavlja leading zero na jednoznamenkaste brojeve
	lcd_gotoxy(0,0); // postavlja se pokaziva? na 1. red (row0) i 1. znak
	lcd_puts("TIME: ");
	lcd_gotoxy(6,0); // postavlja se pokaziva? na 2. red (row1) i 1. znak
	lcd_puts(datetimebuffer);

	sprintf(datetimebuffer, "%02u/%02u/20%02u", date.day, date.month, date.year);
	lcd_gotoxy(0,1);
	lcd_puts("DATE: ");
	lcd_gotoxy(6,1);
	lcd_puts(datetimebuffer);

	OutputCheck(vrijeme, datum, trajanje, korisnik);   //funkcija za omogucavanje izlaza

}

void UserDefinedTime(struct unesenoVrijeme *vrijeme, struct uneseniDatum *datum)
{
	struct Tipka tipka;            //svim tipkama pocetno stanje je "1"

	char datetimebuffer[12];

	tipka.LEFT = (PIND >> PD6)&1;
	tipka.UP =  (PIND >> PD5)&1;
	tipka.DOWN = (PIND >> PD2)&1;
	tipka.RIGHT = (PIND >> PD4)&1;
```

```
tipka.SELECT = (PIND >> PD3)&1;

lcd_init(LCD_DISP_ON);

if(tipka.RIGHT== 0)
	vrijeme->horizontalno++;
if(tipka.LEFT == 0)
	vrijeme->horizontalno--;

switch(vrijeme->horizontalno)
	{
		case 0:
			{
				lcd_gotoxy(0,1);
				lcd_puts("HOUR");

				if(tipka.UP == 0)
				{
					if(vrijeme->hour <= 0x16)

						vrijeme->hour++;
					else
						vrijeme->hour = 0x00;

				}

				if(tipka.DOWN == 0)
				{
					if(vrijeme->hour >= 0x01)
						vrijeme->hour--;
					else
						vrijeme->hour = 0x17;
				}
				break;
			}

		case 1:
		{			lcd_gotoxy(0,1);
				lcd_puts("MIN");
				if(tipka.UP == 0)
				{
					if(vrijeme->min <= 0x3A)
						vrijeme->min++;
					else
						vrijeme->min = 0x00;
				}

				if(tipka.DOWN == 0)
				{
					if(vrijeme->min >= 0x01)
						vrijeme->min--;
					else
						vrijeme->min = 0x3B;
				}
				break;
		}

		case 2:
```

```
{
	lcd_gotoxy(0,1);
	lcd_puts("SEC");
	if(tipka.UP == 0)
	{
		if(vrijeme->sec <= 0x3A)
		vrijeme->sec++;
		else
		vrijeme->sec = 0x00;
	}

	if(tipka.DOWN == 0)
	{
		if(vrijeme->sec >= 0x01)
		vrijeme->sec--;
		else
		vrijeme->sec = 0x3B;
	}
	break;
}
case 3:
{
			lcd_gotoxy(0,1);
			lcd_puts("DAY");
			if(tipka.UP == 0)
			{
				if(datum->day <= 0x1E)
				datum->day++;
				else
				datum->day = 0x00;
			}

			if(tipka.DOWN == 0)
			{
				if(datum->day >= 0x01)
				datum->day--;
				else
				datum->day = 0x1F;

			}
		break;
}
case 4:
{
				lcd_gotoxy(0,1);
				lcd_puts("MNTH");
				if(tipka.UP == 0)
				{
					if(datum->month <= 0xB)
					datum->month++;
					else
					datum->month = 0x00;
				}

				if(tipka.DOWN == 0)
				{
					if(datum->month >= 0x01)
					datum->month--;
					else
```

```
                                                datum->month = 0xC;

                                        }
                                break;
                        }

                        case 5:
                        {                               lcd_gotoxy(0,1);
                                                        lcd_puts("YEAR");
                                                        if(tipka.UP == 0)
                                                        {
                                                                if(datum->year <= 0x62)
                                                                datum->year++;
                                                                else
                                                                datum->year = 0x00;
                                                        }

                                                        if(tipka.DOWN == 0)
                                                        {
                                                                if(datum->year >= 0x01)
                                                                datum->year--;
                                                                else
                                                                datum->year = 0x63;

                                                        }
                                        break;
                        }
                        default: vrijeme->horizontalno = 0;
                        break;
                }

        sprintf(datetimebuffer, "%02u:%02u:%02u", vrijeme->hour, vrijeme->min, vrijeme->sec);

        lcd_gotoxy(0,0); // postavlja se pokaziva? na 1. red (row0) i 1. znak
        lcd_puts("SET: ");
        lcd_gotoxy(6,0); // postavlja se pokaziva? na 2. red (row1) i 1. znak
        lcd_puts(datetimebuffer);

        sprintf(datetimebuffer, "%02u/%02u/20%02u", datum->day, datum->month, datum->year);

        lcd_gotoxy(6,1);
        lcd_puts(datetimebuffer);
}

void Duration(struct vrijemeTrajanja *trajanje)
{
        struct Tipka tipka;   //svim tipkama pocetno stanje je "1"

        char timeBuffer[12];

        tipka.LEFT = (PIND >> PD6)&1;
        tipka.UP =  (PIND >> PD5)&1;
        tipka.DOWN = (PIND >> PD2)&1;
        tipka.RIGHT = (PIND >> PD4)&1;
        tipka.SELECT = (PIND >> PD3)&1;

        lcd_init(LCD_DISP_ON);
```

```
if(tipka.RIGHT== 0)
trajanje->trajanjeHorizontalno++;
if(tipka.LEFT == 0)
trajanje->trajanjeHorizontalno--;

switch(trajanje->trajanjeHorizontalno)
{
	case 0:
	{
		lcd_gotoxy(0,1);
		lcd_puts("HOUR");
		if(tipka.UP == 0)
		{
			if(trajanje->hour <= 0x16)
			trajanje->hour++;
			else
			trajanje->hour = 0x00;
		}

		if(tipka.DOWN == 0)
		{
			if(trajanje->hour >= 0x01)
			trajanje->hour--;
			else
			trajanje->hour = 0x17;
		}
		break;
	}

	case 1:
	{
		lcd_gotoxy(0,1);
		lcd_puts("MIN");
		if(tipka.UP == 0)
		{
			if(trajanje->min <= 0x3A)
			trajanje->min++;
			else
			trajanje->min = 0x00;
		}

		if(tipka.DOWN == 0)
		{
			if(trajanje->min >= 0x01)
			trajanje->min--;
			else
			trajanje->min = 0x3B;
		}
		break;
	}

	case 2:
	{
		lcd_gotoxy(0,1);
		lcd_puts("SEC");
		if(tipka.UP == 0)
		{
			if(trajanje->sec <= 0x3A)
```

```
                        trajanje->sec++;
                        else
                        trajanje->sec = 0x00;
                }

                if(tipka.DOWN == 0)
                {
                        if(trajanje->sec >= 0x01)
                        trajanje->sec--;
                        else
                        trajanje->sec = 0x3B;
                }
                break;
        }

        default: trajanje->trajanjeHorizontalno = 0;
        break;
    }

    sprintf(timeBuffer, "%02u:%02u:%02u", trajanje->hour, trajanje->min, trajanje->sec);

    lcd_gotoxy(0,0); // postavlja se pokaziva? na 1. red (row0) i 1. znak
    lcd_puts("ENDING: ");
    lcd_gotoxy(8,1); // postavlja se pokaziva? na 2. red (row1) i 1. znak
    lcd_puts(timeBuffer);
}

void RelayNumber(struct vrijemeTrajanja *trajanje)
{
    struct Tipka tipka;
    char relayNumber[4];

    tipka.UP =  (PIND >> PD5)&1;
    tipka.DOWN = (PIND >> PD2)&1;

    lcd_init(LCD_DISP_ON);

    if(tipka.UP == 0)
    {
        if(trajanje->brojReleja <= 0x03)
            trajanje->brojReleja++;
        else
            trajanje->brojReleja = 0x01;
    }

    if(tipka.DOWN == 0)
    {
        if(trajanje->brojReleja >= 0x02)
        trajanje->brojReleja--;
        else
        trajanje->brojReleja = 0x04;
    }

    itoa(trajanje->brojReleja, relayNumber, 10);

    lcd_gotoxy(0,0); // postavlja se pokaziva? na 1. red (row0) i 1. znak
    lcd_puts("RELAY NUMBER: ");
    lcd_gotoxy(14,0); // postavlja se pokaziva? na 2. red (row1) i 1. znak
```

```
	lcd_puts(relayNumber);
}
```

display.h

```
/*
 * display.h
 *
 * Created: 10-Oct-18 13:24:41
 *  Author: Domagoj Krizanec
 */


#ifndef DISPLAY
#define DISPLAY

void DateTime(struct unesenoVrijeme *vrijeme, struct uneseniDatum *datum, struct
vrijemeTrajanja *trajanje,  struct users *korisnik); // prototip funkcije DateTime
void UserDefinedTime (struct unesenoVrijeme *vrijeme, struct uneseniDatum *datum);//prototip
funkcije UserDefinedTime
void Duration(struct vrijemeTrajanja *trajanje); //prototip funkcije Duration
void RelayNumber(struct vrijemeTrajanja *trajanje); //broj izlaza koji se koristi

#endif
```

bluetooth.c

```
/*
 * bluetooth.c
 *
 * Created: 19-Nov-18 20:17:28
 *  Author: Domagoj Krizanec
 */

#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include "bluetooth.h"
#include "lcd.h"
#include "uart.h"


unsigned concatenate(uint8_t x, uint8_t y)
 {
	uint8_t pow = 10;
	while(y >= pow)
	pow *= 10;
	return x*pow + y;
}

void BluetoothSetUp(struct info *varijable, struct users *korisnik)
{

	PETLJA: while (1)
```

```
	{
		varijable->uartRecieve = 0;

		varijable->uartRecieve = uart0_getc();
			//primanje bluetooth informacije i spremanje u buffer-- informacija
se prima u ASCII vrijednosti

		if(varijable->uartRecieve == 33)
				//znak '!' - prebacivanje u drugi režim rada -- IZLAZ IZ
BLUETOOTH FUNKCIJE
		{
			varijable->functionFlag = 0;
			varijable->uneseneZnamenke = 0;
			varijable->unosVrijednostiKorisnika = 0;
			varijable->uartRecieve = 0;
			break;
		}

		if(varijable->uartRecieve == 10)
// bluetooth veza izgubljena
		{
			lcd_gotoxy(0,1);
			uart0_puts("Device lost");
			varijable->functionFlag = 0;
			break;
		}

		if(varijable->uartRecieve == 59)
				//novi korisnik- znak ";"
		{
			if(korisnik->brojKorisnika < 10)    korisnik->brojKorisnika++;
			else korisnik->brojKorisnika = 0;

			varijable->uneseneZnamenke = 0;
			varijable->unosVrijednostiKorisnika = 0;
			goto PETLJA;
		}

		if(varijable->uartRecieve == 32)
//odvajanje vrijednosti razmakom
		{
			memset(varijable->FinalUart, 0, sizeof(varijable->FinalUart));
					//postavljanje parametra Final Uart u
Nulu
			memset(varijable->UART, 0, sizeof(varijable->UART));
			varijable->unosVrijednostiKorisnika++;
			varijable->uneseneZnamenke = 0;
			goto PETLJA;
		}

		if(varijable->uartRecieve >= 48 && varijable->uartRecieve <= 58)
//RAD S BROJEVIMA
		{

			if(varijable->uartRecieve == 58)
					// znak ":"   nova vrijednost -HH/MM/SS
			{
				varijable->uneseneZnamenke++;
```

```
                        varijable->previousUart = 0;
                        goto PETLJA;
                }

                varijable->uartRecieve = varijable->uartRecieve - 48;
                                                //oduzimanjem dobijemo stvarnu
vrijednost broja

                varijable->concatenatedNumber = concatenate(varijable->previousUart,
varijable->uartRecieve);

                if(varijable->concatenatedNumber <=0 && varijable->concatenatedNumber
>=60)                           //ukoliko je unesni broj izvan brojevnog podru?ja
                {
                        uart0_puts("Invalid number input! Please use format HH/MM/SS");
                        goto PETLJA;
                }

                if (varijable->unosVrijednostiKorisnika == 1)
                {
                        if(varijable->uneseneZnamenke == 0) korisnik-
>openingHour[korisnik->brojKorisnika] = varijable->concatenatedNumber;
                        else if(varijable->uneseneZnamenke == 1) korisnik-
>openingMin[korisnik->brojKorisnika] = varijable->concatenatedNumber;
                        else if(varijable->uneseneZnamenke == 2) korisnik-
>openingSec[korisnik->brojKorisnika] = varijable->concatenatedNumber;
                        else varijable->uneseneZnamenke = 0;
                }
                if (varijable->unosVrijednostiKorisnika == 2)
                {
                        if(varijable->uneseneZnamenke == 0) korisnik-
>closingHour[korisnik->brojKorisnika] = varijable->concatenatedNumber;
                        else if(varijable->uneseneZnamenke == 1) korisnik->closingMin
[korisnik->brojKorisnika] = varijable->concatenatedNumber;
                        else if(varijable->uneseneZnamenke == 2) korisnik-
>closingSec[korisnik->brojKorisnika] = varijable->concatenatedNumber;
                        else varijable->uneseneZnamenke = 0;
                }
                if(varijable->unosVrijednostiKorisnika == 3)
                {
                        if(varijable->uneseneZnamenke == 0)
                        {
                                korisnik->relayNumber[korisnik->brojKorisnika] =
varijable->uartRecieve;
                        }
                }

                        varijable->previousUart = varijable->uartRecieve;
        }

        sprintf(varijable->UART, "%c", varijable->uartRecieve);
                        //prebacuje iz decimalnog u ASCII
        strcat(varijable->FinalUart, varijable->UART);
                        // funkcija dodaje drugi dio prvome, te vraca prvi dio
natrag
```

```
		if(isalpha(varijable->UART[0]))
                                                // provjerava ukoliko je uneseni char simbol
slovo
		{
			strcpy(korisnik->userName[korisnik->brojKorisnika], varijable-
>FinalUart);


		}

		for(uint8_t i = 0; i< korisnik->brojKorisnika; i++)
		{
			if(korisnik->brojKorisnika != 0)
			{
				if((korisnik->openingHour[i] == korisnik->openingHour[korisnik-
>brojKorisnika])
				 && (korisnik->openingMin[i] == korisnik->openingMin[korisnik-
>brojKorisnika])
				 && (korisnik->openingSec[i] == korisnik->openingSec[korisnik-
>brojKorisnika])
				 && (korisnik->relayNumber[i] == korisnik->relayNumber[korisnik-
>brojKorisnika]))
				{
					memset(korisnik->userName[korisnik->brojKorisnika],0,
sizeof(korisnik->userName[korisnik->brojKorisnika]));  //termin vec zauzet
					korisnik->openingHour[korisnik->brojKorisnika] = 0;
					korisnik->openingMin[korisnik->brojKorisnika] = 0;
					korisnik->openingSec[korisnik->brojKorisnika] = 0;
					korisnik->relayNumber[korisnik->brojKorisnika] = 0;

					korisnik->brojKorisnika--;
					uart0_puts("WARNING! Input time and relay are unavailable!
User that has that period is ");
					uart0_puts(korisnik->userName[i]);

				}
			}
		}
		break;

	}
}
```

bluetooth.h

```
*
 * bluetooth.h
 *
 * Created: 19-Nov-18 20:17:47
 *  Author: Domagoj Krizanec
 */
#ifndef BLUETOOTH
#define BLUETOOTH

struct users
{
	char userName[20][8];
```

```
        uint8_t openingHour[20];
        uint8_t closingHour[20];
        uint8_t openingMin[20];
        uint8_t closingMin[20];
        uint8_t openingSec[20];
        uint8_t closingSec[20];
        uint8_t relayNumber[20];
        uint8_t brojKorisnika;

};

struct info
{
        char UART[8];
        char FinalUart[8];
        uint8_t uartRecieve;
        uint8_t previousUart;
        uint8_t concatenatedNumber;
        uint8_t uneseneZnamenke;                //HH/MM/SS
        uint8_t unosVrijednostiKorisnika;       //Otvaranje-Zatvaranje-BrojReleja
        uint8_t testI;
        uint8_t functionFlag;
};

void BluetoothSetUp(struct info *varijable, struct users *korisnik);

#endif
```