Tehničko Veleučilište u Zagrebu

Stručni studij Mehatronike

Akademska godina 2020/2021

SEMINARSKI RAD

Stolni digitalni sat/budilica s prognozom vremena

Student:

Marko Josipović

Profesori:

PURS: dr.sc. Toni Bjažić, prof.v.š.

KTM: dr.sc. Tomislav Pavlović

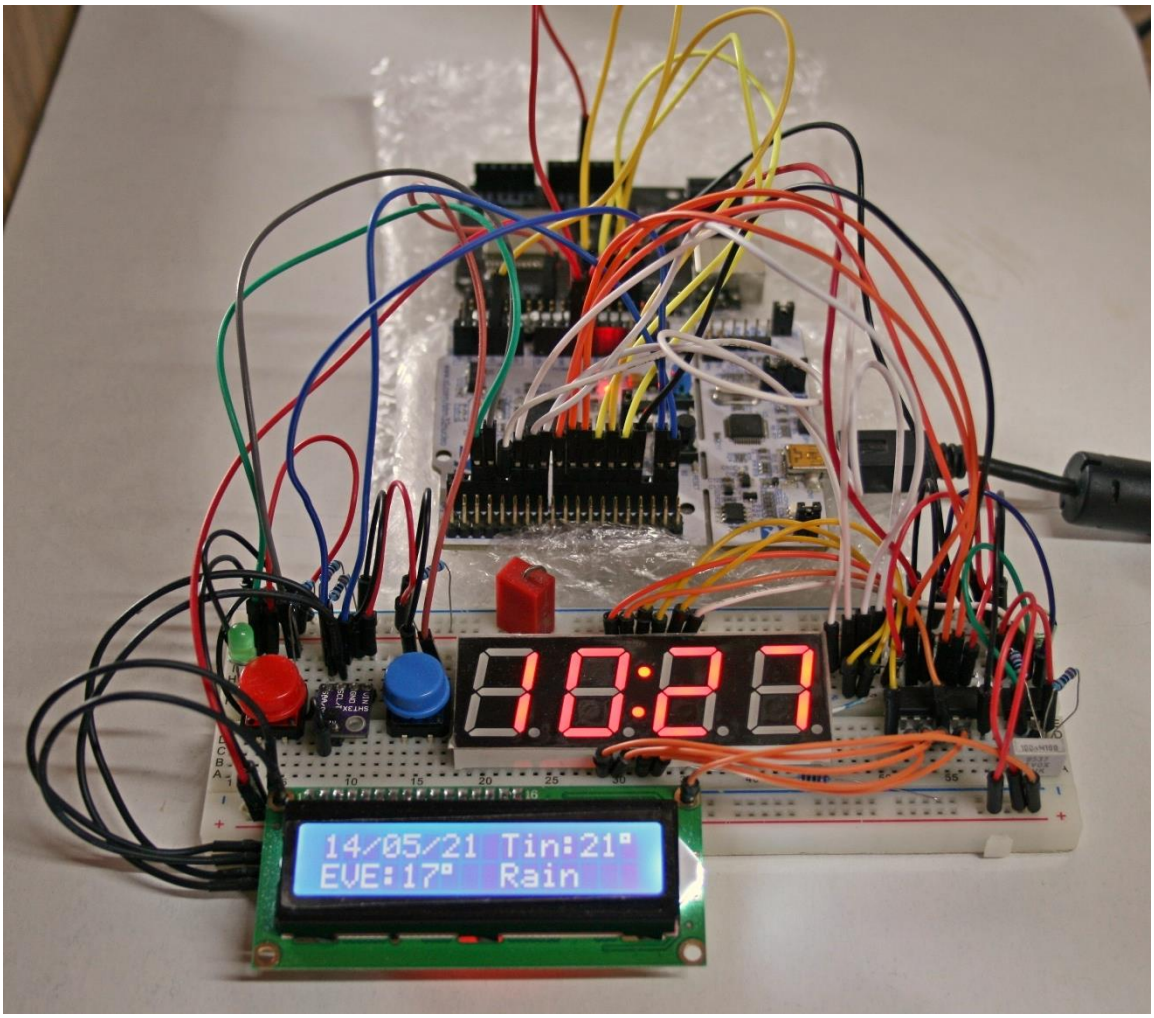
Sadržaj

1. Opis sklopa

Tema ovog seminarskog rada je stolni digitalni sat/budilica (u nastavku „sat“). Primarna svrha sata je prikazivanje vremena na četveroznamenkastom 8-segmentnom numeričkom LED display-u, uz adiciju 16x2 LC (Liquid Crystal) monokromatskog display-a, na kojem se prikazuju datum, unutarnja temperatura i vlaga te prognoza temperature po dobu dana i vremenske prilike.

Sat također ima funkciju budilice koja se namješta putem internetskog sučelja, te se izvršava preko ugrađenog piezo zvučnika. Uz navedene funkcije sat ima mogućnost mjerenja trenutačne temperature i vlage zraka u prostoriji u kojoj se nalazi putem senzora SHT30.

Budući da pravila kolegija zahtijevaju dvije različite vrste komunikacije, odlučio sam se za I2C protokol kojim bi mbed (NUCLEO FB-072RB) komunicirao sa senzorom SHT30 i 16x2 LCD-om (zajednička sabirnica), te za SPI protokol kojim bi se vršila komunikacija s drugim mikrokontrolerom. Na prethodno sam se odlučili jer sam već posjedovao razvojnu pločicu pogonjenu mikroprocesorom ESP-WROOM-32 koji dolazi s ugrađenim WiFi i Bluetooth modulima te mogućnošću serijske komunikacije preko prije spomenutog protokola. Ovime je satu osigurano povezivanje na Internet, te samim time široka paleta mogućnosti koju smo iskoristili za dohvaćanje: stvarnog (trenutačnog) vremena, namještene budilice i prognoze vanjske temperature i vremenskih prilika za nadolazeće doba dana (jutro, podne, večer).

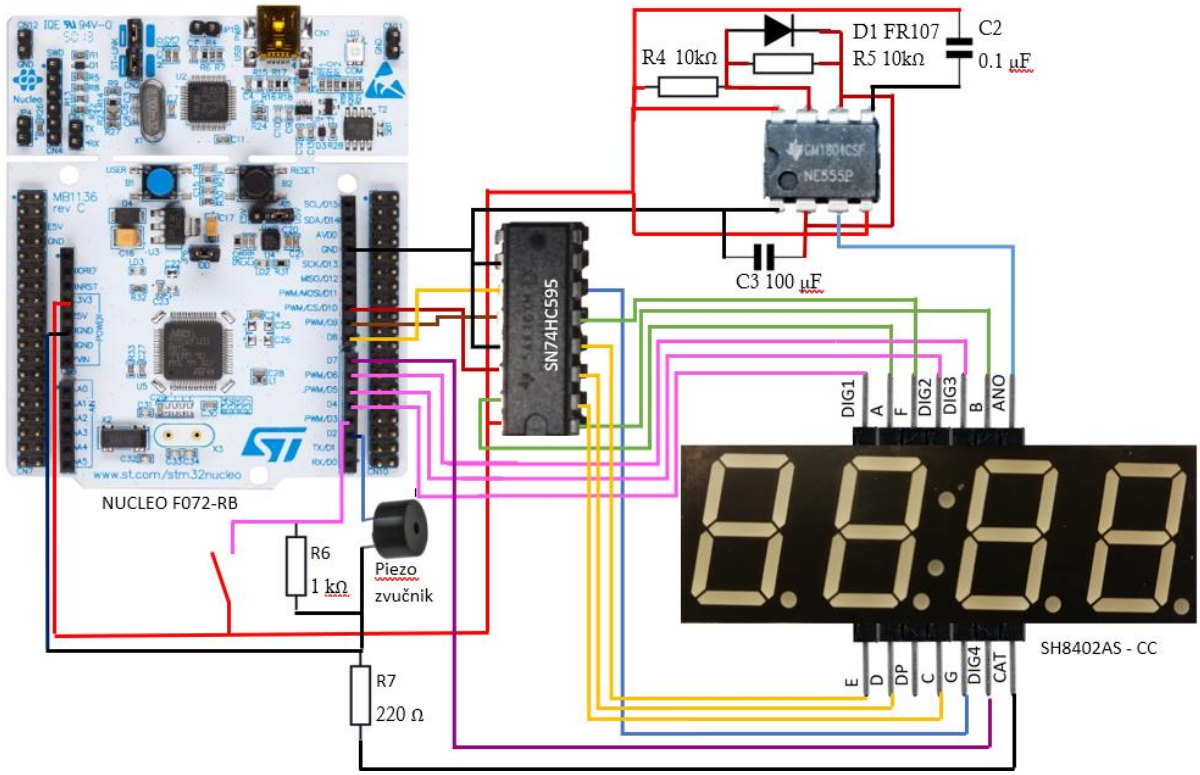


Slika 1-1: Izgled prototipa

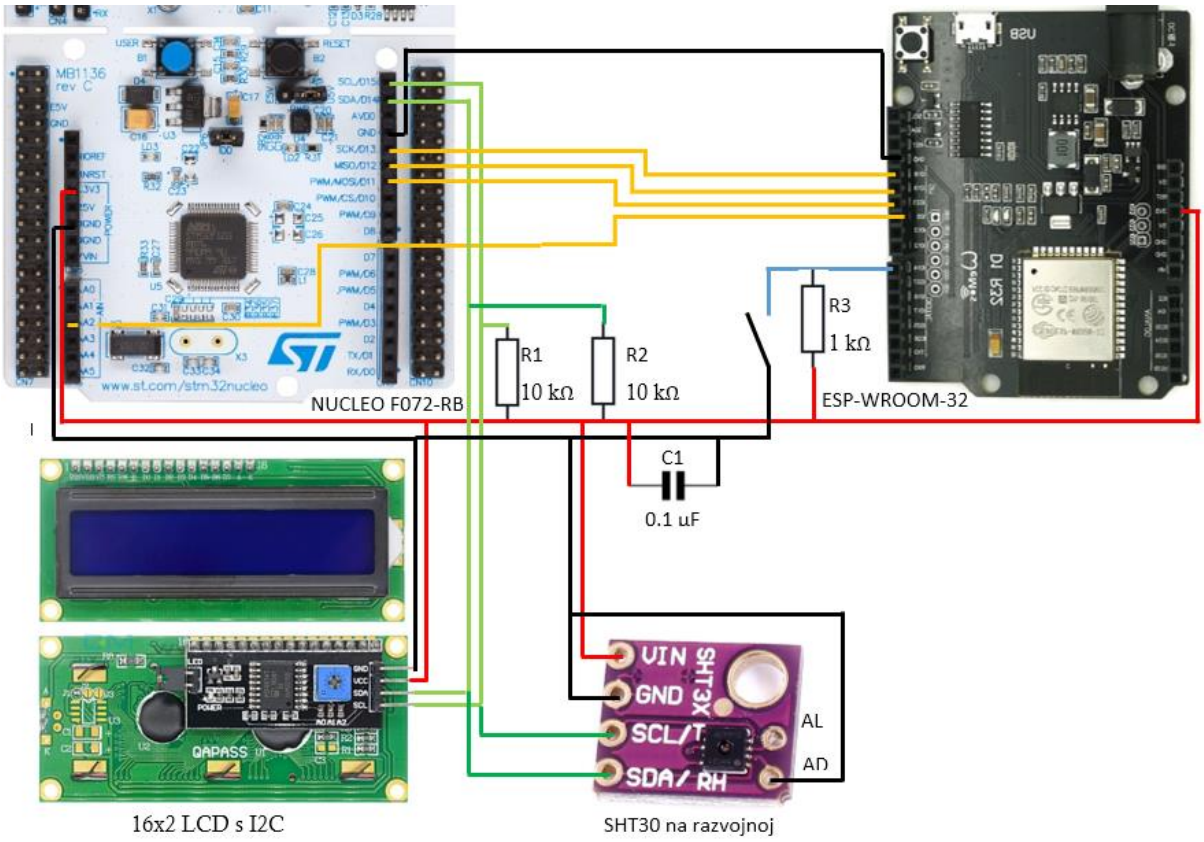
Komponenta	Količina
NUCLEO FB-072RB	1
ESP-WROOM-32 na razvojnoj pločici	1
SH8402AS (numerički ekran, CC)	1
SN74HC595 (posmični registar)	1
16x2 LCD s I2C modulom	1
SHT30 na razvojnoj pločici	1
NE555P	1
Dioda FR107	1
Piezo zvučnik (eng. buzzer)	1
Kondenzator 0.1 µF	2
Kondenzator 100 µF	1
Otpornik 1kΩ	2
Otpornik 10kΩ	4
Tipkalo NO	2

Tablica 1-1: Popis komponenti

1.1 Shema sklopa



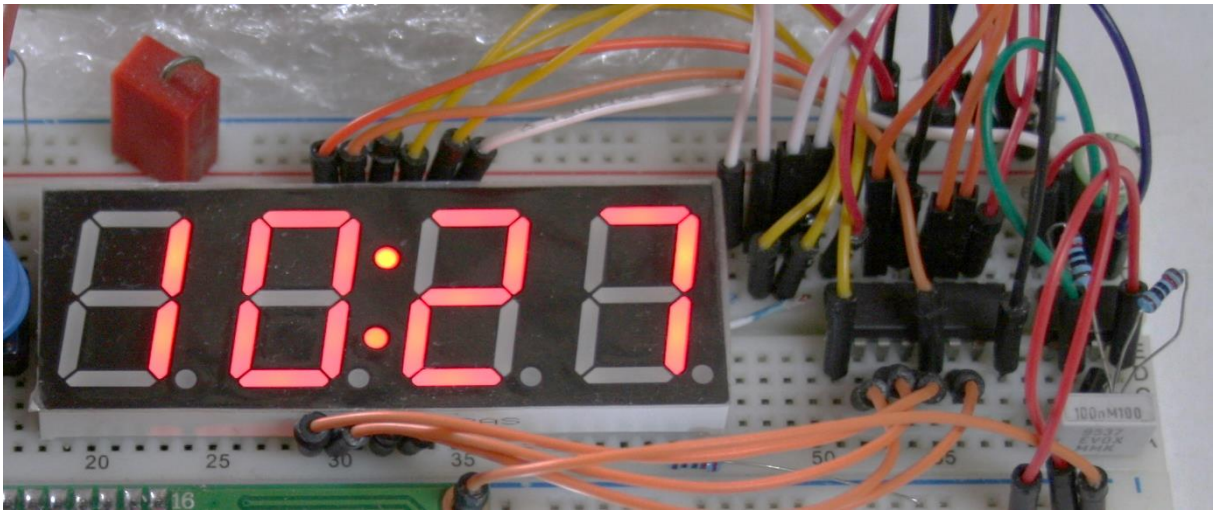
Slika 1.1-2 Shema dijela sklopa s NUCLEO-m, SN74HC595, NE55P, SH8402AS i pasivnim komponentama



Slika 1.1-2 Shema dijela sklopa s NUCLEO-m, LCD-om, SHT30, ESP32 i pasivnim komponentama

2. Numerički sat

Temelj projekta iz perspektive samoga sata je postići ispis trenutačnog vremena na numeričkom ekranu (SH8402AS). Ekran koji je korišten u ovome seminarskom radu funkcionira na principu zajedničke katode. Naime, kako bi se na ekranu mogli prikazivati podaci potrebno je na njegove pinove A do G (+ pin DP koji se ne koristi) dovesti odgovarajuće naponske razine (ovisno o kojoj se brojci radi, logička jedinica aktivira potrebni segment, dok logička nula deaktivira isti) te je zatim potrebno dovesti logičku nulu na jedan od pinova DIG1 do DIG4 koji želimo upaliti. Zbog ovakvog načina upravljanja sklop je ograničen na samo jednu brojku koja se može nalaziti na bilo kojem od segmenata istovremeno. Solucija ovoga problema je tromost ljudskoga oka koje ne primjećuje relativno brze promjene već ih spaja u jednu sliku, te se na takav način ovim zaslonom upravlja po principu dovođenja visokih naponskih razina na linije A do G, a potom se na korespondirajuću liniju za znamenku (DIG1 do DIG4) postavi logička nula u trajanju od 5 milisekundi (ili se pokreće funkcija osvježavanja LCD-a, više u nastavku). Nakon isteka odgode ili završetka funkcije isključuje se prethodno aktivna znamenka (postavljanjem logičke jedinice na korištenu DIG liniju), te se funkcija prikaz ovime završava. Ovaj proces je iterativan za sve znamenke (njih četiri), te se tako stvara prikaz sata na numeričkom ekranu koji se čini statičan i bez titranja.



Slika 2-1: Numerički ekran s posmičnim registrom

Funkcija urica je osnovna funkcija za pokretanje svih ostalih funkcija, izvršava se u while petlji koja se nalazi unutar main petlje. Pokretanje iste uvjetovano je prvotnom provjerom stanja SPI prijenosa podataka, te u slučaju kada nije aktivan omogućeno je izvršavanje funkcije urica.

```
286         while(1){
287               if(spi_port.receive()){
288                     SPI_receive();
289               }else{
290                     urica();
```

2.1 Pretvorba oblika vremena (EPOCH)

Kako bi funkcija urica, odnosno funkcija za prikaz brojki na numeričkom ekranu mogla ispravno funkcionirati, prvotno je potrebno pretvoriti zapis vremena iz EPOCH formata u ljudski čitljivi, odnosno iz broja sekundi od ponoći 01.01.1970. u sate, minute i sekunde. Iz tog razloga se na samome početku funkcije urica pokreće funkcija imena EPOCH_and_ALARM (tipa void) koja za svoj rad ne prihvaća ulazne parametre već ima inicijalizirane globalne varijable.

```
140   void urica(void){
141         EPOCH_and_ALARM();
```

Globalne varijable iz reda broj 24 su zapravo liste varijabli slične matrici koje se koriste za pristupanje većoj količini podataka kroz isti pokazivač. Konkretne liste koriste se za pohranu pretvorenih podataka (ljudski čitljivi format) o vremenu dobivenog od strane funkcije koja je zadužena za RTC (eng. Real Time Clock).

```
24    struct tm  ts;
```

Potom slijede varijable u kojima će se direktno pohraniti podaci o vremenu u ljudski čitljivom formatu.

```
25    int second, minute, hour, day, month, year;
```

Naposljetku slijedi varijabla koja će u sebi sadržavati zapis vremena u EPOCH formatu.

```
26    time_t seconds;
```

Funkcija EPOCH_and_ALARM pokreće se prethodno svakom ciklusu osvježavanja numeričkog ekrana unutar funkcije urica kako bi za svako osvježavanje ekrana bili u potpunosti sigurni da su informacije točne.

```
140   void urica(void){
141         EPOCH_and_ALARM();
142         prikaz(array_data[hour/10], 0b0111);
143         prikaz(array_data[hour%10], 0b1011);
144         prikaz(array_data[minute/10], 0b1101);
145         prikaz(array_data[minute%10], 0b1110);
146   }
```

Funkcija, kada pozvana, najprije aktivira funkciju time kojoj za ulazni parametar daje tzv. praznu varijablu, odnosno varijablu bez sadržaja na koju će funkcija time odgovoriti listom varijabli koje će u sebi sadržavati zapise vremena u svim potrebnim formatima. Iste će se pohraniti u listu varijabli imena seconds.

```
92          time_t seconds = time(NULL);
```

Nakon prethodne radnje stvaraju se znakovne varijable (eng. char) koje će biti objašnjene u nastavku, odnosno u kontekstu koda. Varijable su ograničene na jedno mjesto više od maksimalno potrebnog broja mjesta (npr. za sekunde su potrebne dvije znamenke pa je zato stavljeno ograničenje na maksimalno tri znamenke).

```
94          char second_S[3]; //ss
95          char minute_S[3]; //mm
96          char hour_S[3]; //hh
97          char day_S[3]; //DD
98          char month_S[3]; //MM
99          char year_S[3]; //YY
```

Nakon inicijalizacije varijabli radi se separacija iz liste varijabli u zasebne varijable kako bi manipulacija nad istima bila olakšana. Ovaj proces se obavlja uz pomoć funkcije strftime koja za svoje ulazne parametre prihvaća redom: isključivo karakternu varijablu (eng. char), broj mjesta za znakove unutar prethodne varijable, oznaku varijable iz liste varijabli i listu varijabli. Funkcija imena localtime brine o pravilnom podešavanju zapisa i vremenske zone u danim varijablama.

```
101         strftime(second_S, 3, "%S\n\r", localtime(&seconds));
102         strftime(minute_S, 3, "%M\n\r", localtime(&seconds));
103         strftime(hour_S, 3, "%H\n\r", localtime(&seconds));
104         strftime(day_S, 3, "%d\n\r", localtime(&seconds));
105         strftime(month_S, 3, "%m\n\r", localtime(&seconds));
106         strftime(year_S, 4, "%y\n\r", localtime(&seconds));
```

Tijekom prethodnih procesa stvorene su karakterne varijable (eng. char) od kojih svaka zasebno sadrži podatke o trenutačnom vremenu. Za lakšu manipulaciju podacima potrebo ih je pretvoriti u cjelobrojni tip varijable (eng. int). Za potrebe izvršenja ove zadaće iskorištena je funkcija imena atoi čija je svrha upravo pretvorba karakterne varijable (eng. char) u cjelobrojnu varijablu (eng. int).

```
109         second = atoi(second_S);
110         minute = atoi(minute_S);
111         hour = atoi(hour_S);
112         day = atoi(day_S);
113         month = atoi(month_S);
114         year = atoi(year_S);
```

Ovime su pripremljeni podaci za ostale funkcije koje imaju potrebe za istima, te je funkcija pretvorbe EPOCH vremena u ljudski čitljivi format završena. Ostatak funkcije namijenjen je izvršavanju postavljenog alarma koji je objašnjen u poglavlju 7.

2.2 Ispis vremena na numeričkome ekranu

Kao što je rečeno u uvodu ovoga poglavlja, za prikaz sata koristi se četveroznamenkasti numerički ekran sa osam segmenata u kombinaciji s posmičnim registrom koji smanjuje broj iskorištenih digitalnih izlaza na NUCELO mikrokontroleru (ulaz podataka je serijski, a izlaz paralelni). S obzirom da numerički ekran radi na principu zajedničke katode, na segmente se dovode logičke jedinice, a krugovi se zatvaraju dovođenjem logičke nule na jednu od četiri znamenke (koja je u tome trenutku potrebna).

Prije analize funkcija za prikaz brojki na numeričkom ekranu potrebno je inicijalizirati nekolicinu digitalnih izlaza putem kojih će se prenositi podaci. Za potrebe posmičnog registra korištene su ukupno tri linije od kojih je jedna za prijenos podataka (ulaz SER), a druge dvije za takt rada serijske komunikacije i registra (ulaz SRCLK i RCLK, respektivno). Ova komponenta također ima još dvije linije od kojih je jedna za onemogućavanje paralelnog izlaza registra, a druga za brisanje sadržaja istog (obje linije su inverznog tipa, te su iz toga razloga spojene na visoku logičku razinu kako bi se deaktivirale).

```
5     //SHIFT registar
6     DigitalOut data(D10);
7     //OE se ne koristi
8     DigitalOut RCLK(D9);
9     DigitalOut SRCLK(D8);
10    //SRCLRn se ne koristi
```

Nakon inicijalizacije linija za komunikaciju s posmičnim registrom (koji upravlja segmentima), potrebno je ostvariti kontrolu za drugi kraj numeričkog ekrana koji će zatvarati strujni krug na željenoj znamenki. Kontrola ovoga dijela se izvodi direktno putem digitalnih izlaza koji su organizirani unutar BusOut funkcije koja je odabrana za kasnije lakše programiranje jer funkcija omogućava istovremenu kontrolu nad više digitalnih izlaza.

```
12    BusOut segme(D7, D6, D5, D4);
```

S obzirom da je numerički ekran striktno pasivna komponenta (nema logičkih krugova, samo svjetleće diode), a posmični registar relativno primitivna komponenta koja privremeno pohranjuje podatke i pretvara serijsku u paralelnu komunikaciju, potrebno je željenu brojku pretvoriti u za prikaz prigodan oblik. Naime, posmični registar će podatke dobivene na svoj serijski ulaz postaviti na paralelni izlaz tako da će najmanje značajan bit biti na mjestu A, bit lijevo od njega na mjestu B i tako dalje do najznačajnijeg bita koji označava decimalnu točku (uvijek 0 jer nije potrebna). Kako bi se izbjegli kompleksniji algoritmi koji obavljaju pretvorbu iz dekadske brojke u zapis pogodan za numerički ekran, obavljena je ručna konverzija podataka. Od tih podataka napravljena je konstantna jednodimenzionalna matrica cjelobrojnog tipa (eng. int) koja na nultom mjestu sadržava zapis nule pogodne za prikaz na numeričkom ekranu potom na sljedećem, prvom mjestu, zapis jedinice pogodne za prikaz i tako dalje sve do broja 9 (uključujući isti).

```
31    const int array_data[] = {0b00111111, 0b00000110, 0b01011011,
0b01001111, 0b01100110, 0b01101101, 0b01111101, 0b00000111, 0b01111111,
0b01101111};
```

Za potrebe LCD-a, opisan u 3. poglavlju, inicijalizira se varijabla imena izmjena koja će služiti kao brojač prolazaka kroz funkciju prikaz (detaljnija analiza prilikom upotrebe varijable kasnije u kodu).

```
55    int izmjena=0;
```

S obzirom da postoje ukupno četiri znamenke unutar funkcije urica, napravljena su četiri individualna poziva funkcije za prikaz podataka na numeričkome ekranu prigodnog imena prikaz. Funkcija prikaz je

tipa void, no prihvaća dva ulazna parametra. Prvi ulazni parametar je brojka koja će se prikazati, dok je drugi parametar binarni broj za BusOut funkciju koja će zatvoriti strujni krug na jednoj od ukupno četiri znamenke, te će time prikazati prethodnu brojku (logička nula označava potrebnu znamenku). Funkcija prikaz može prihvatiti samo jednu brojku istovremeno (ekran može prikazati samo jednu znamenku istovremeno), potrebno je sate i minute rastaviti na jedinice i desetice (obavljeno naredbom za cjelobrojno dijeljenje, sa i bez ostatka). Naime, u slučaju potrebe jedinice varijabla sati/minuta cjelobrojno se dijeli brojem 10, te se uzima ostatak toga dijeljenja(oznaka „%“, dok se za deseticu uzima rezultat dijeljenja (oznaka „/“). Dobiveni jednoznamenkasti rezultat se uzima kao redni broj podatka unutar matrice array_data na čijem mjestu se nalazi ekvivalent u zapisu povoljnom za numerički ekran, nakon čega se predaje funkciji prikaz.

```
140   void urica(void){
141         EPOCH_and_ALARM();
142         prikaz(array_data[hour/10], 0b0111);
143         prikaz(array_data[hour%10], 0b1011);
144         prikaz(array_data[minute/10], 0b1101);
145         prikaz(array_data[minute%10], 0b1110);
146   }
```

Funkcija prikaz ulazne parametre prihvaća kao varijable naziva podaci (brojka koja će se prikazati) i segx (znamenka na kojoj će se ista prikazati). Obje varijable su karakternog tipa (eng. char) radi uštede memorije (cjelobrojni tip zauzima dva ili četiri bajta, dok karakterni zauzima samo jedan bajt).

```
60    void prikaz(char podaci, char segx){
```

Lokalna varijabla imena out služiti će kao privremeni spremnik podatka koji se u tome trenutku priprema, a potom se šalje serijskom komunikacijom prema posmičnom registru.

```
61          int out=0;
```

Za slanje podataka na posmični registar potrebno je napraviti funkciju koja će podatak varijable podaci separirati u zasebne dijelove, te ih jednoga po jednoga postavljati na liniju data i označavati linijom SRCLK da su isti spremni za čitanje od strane registra. S obzirom da ukupno ima 8 bitova, funkcija for podešena je na ponavljanje jednako toliko puta. Za pravilno slanje podataka potrebno ih je redom slati od najmanje značajnog bita prema više značajnim bitovima, te iz tog razloga varijabla imena i kreće od brojke sedam. Varijabla s podacima se binarno pomiče za i mjesta udesno, te se potom spaja logičkom operacijom „I“ (eng. AND) s dekadskom jedinicom kako bi bili sigurni da samo krajnje desni bit (nakon obrade) bude postavljen na izlaznu liniju (out).

```
64          for(int i=7;i>=0;i--){
65                out=podaci>>i & 0b00000001;
66                data=out;
67                wait_us(1);
```

Nakon isteka odgode od jedne mikrosekunde posmičnom registru se signalizira mogućnost čitanja linije s podacima te ponovno nakon jedne mikrosekunde se signal spušta na logičku nulu i varijabla out se izjednačava s nulom za sljedeći podatak u nizu.

```
68                SRCLK=1;
69                wait_us(1);
70                SRCLK=0;
71                out=0;
72          }
```

Prilikom završetka prijenosa podataka na posmični registar označava se ulaz na registru koji omogućuje izlaz podataka na paralelne izlaze, te se potom na izlaze u grupi segme dovodi binarni niz sadržan unutar varijable segx koji će aktivirati korespondentnu znamenku.

```
76          RCLK=1;
77          segme=segx;
```

Svaka znamenka treba biti aktivna nekoliko milisekundi te se to vrijeme svake desete sekunde iskorištava za promjenu podataka na LCD-u kako bi se bolje iskoristilo procesorsko vrijeme i smanjio treptaj numeričkog ekrana.

Prethodno naveden princip rada ostvaren je na način da vrijednost varijable izmjena sa svakim prolaskom kroz petlju prikaz povećava za jedan. U slučaju kada varijabla izmjena ima vrijednost manju od 2000 izvršava se odgoda izvršavanja koda koja traje 4950 mikrosekundi (uz ostale odgode i vrijeme potrebno za izvršavanje prijašnjih linija koda dolazi se do otprilike 5 milisekundi). Za LCD je predviđena promjena prikazanih podataka svakih deset sekundi što je točno svake 2000. iteracije. Iz toga razloga stavljena je if funkcija koja za svoj uvjet ima provjeru stanja varijable izmjena, te kada ista dosegne vrijednost od 2000 pokreće se funkcija za promjenu podataka na LCD-u i vrijednost brojača se vraća na nulu.

```
79          izmjena++;
80          if(izmjena == 2000){
81                display_main();
82                izmjena = 0;
83          }else{
84                wait_us(4950);
85          }
```

Nakon završetka prvog ili drugog slučaja se posmičnom registru onemogućavaju paralelni izlazi, te se funkcija ovime završava.

```
87          RCLK=0;
88    }
```

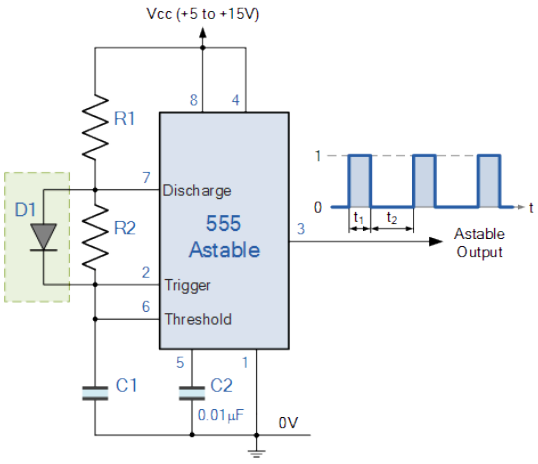
2.3 Astabilni sklop za dvotočku numeričkog ekrana

S misijom izrade ovoga rada u stilu klasičnih stolnih satova izabran je numerički ekran koji na sebi ima ugrađenu dvotočku između druge i treće znamenke. Za poboljšanje estetskog dojma ekrana napravljen je sklop koji će generirati treptajući signal za dvotočku. Kako bi dvotočka imala dodira sa samim satom uzeta je frekvencija od 0.5 Hz i 50% radnoga ciklusa (eng. duty cycle), odnosno način rada pri kojem će oba stanja (upaljeno i gašeno) imati period od jedne sekunde i međusobno se izmjenjivati. Obzirom da s ovako trivijalnom funkcijom nije potrebno opterećivati mikrokontroler izabrali smo precizni tajmer (eng. Single Precision Timer) u obliku integriranog kruga pod oznakom NE555T od proizvođača Texas Instruments.

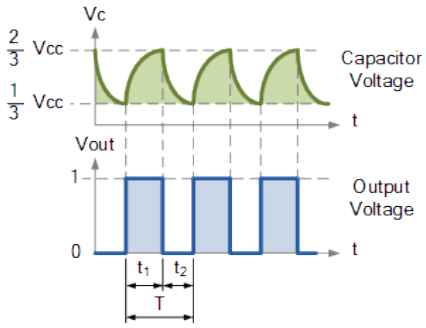
Integrirani krug omogućuje razne načine rada od kojih je primarni astabilni način koji omogućava prije spomenutu funkciju. Informacije o izradi ovoga sklopa preuzete su s foruma Stack Exchange od korisnika pod imenom hkBattousai na sljedećoj poveznici (https://electronics.stackexchange.com/questions/112875/astable-555-timing-circuit-0-5-hz-and-50-duty-cycle). Prema uputama s foruma, idealna shema samoga sklopa trebala bi biti izvedena prema slici 2.3-1. Shema sklopa prikazuje NE555 u spoju s naponskim djelilom, diodom za ograničavanje smjera toka struje i kondenzatorom. Ovaj sklop radi na principu klasičnog RC kruga u kojem se kondenzator puni i prazni ovisno o naponskom djelilu. RC krug povezan je na tranzistor (pin imena trigger) unutar NE555 koji će dati visoko stanje kada je napon na kondenzatoru 2/3 V_{cc}, a nisko stanje pri 1/3 V_{cc} (visoko ili nisko stanje izlazi na pin imena astable output). Formule prema kojima se računaju periodi t_1 i t_2 dani su na slici 2.3-2, te sa jednostavnom analizom se može doći do zaključka da je za 0.5 Hz i 50% radnog ciklusa potrebno t_1 i t_2 izjednačiti, što se postiže izjednačavanjem otpora R_1 s R_2. Pad napona na diodi D1 se zanemaruje. Signal s pina astable output se dovodi na V_{cc} pin numeričkog ekrana.

$$t_1 = \ln(2) R_1 C_1, \quad t_2 = \ln(2) R_2 C_1.$$

Slika 2.3-2 Računanje perioda oscilacija, izvor: https://electronics.stackexchange.com/questions/112875/astable-555-timing-circuit-0-5-hz-and-50-duty-cycle



Slika 2.3-1 NE555 astabilni sklop, izvor: https://electronics.stackexchange.com/questions/112875/astable-555-timing-circuit-0-5-hz-and-50-duty-cycle



Slika 2.3-3 Napon na kondenzatoru u usporedbi s izlaznim signalom, izvor: https://electronics.stackexchange.com/questions/112875/astable-555-timing-circuit-0-5-hz-and-50-duty-cycle

3. 16x2 LCD (Liquid crystal display) sa sučeljem za I2C komunikaciju

Ekran je baziran na tehnologiji tekućih kristala s ugrađenim pozadinskim osvjetljenjem. Tekući kristali posjeduju specifično svojstvo koje se manifestira prilikom prolaska struje kroz iste na način zatamnjenja količine svijetla (od pozadinskog osvjetljenja) koje može izaći prema korisniku. U ovome seminarskom radu korišten je ekran (u daljnjem tekstu LCD) koji ukupno ima 32 mjesta za znakove s time da je podijeljen u 16 stupaca i 2 retka. U svakom od 32 segmenata moguće je iskoristiti 8x5 (stupci x redci) individualnih točaka (eng. pixels), što za ovu primjenu omogućava jasan tekst s relativno male udaljenosti gledanja. Radi pojednostavljenja komunikacija sa samim LCD-om upotrjebljeno je sučelje za konverziju paralelne komunikacije u serijsku (konkretno I2C). S obzirom da je LCD univerzalan, postoji veliki izbor biblioteka koje podržavaju i olakšavaju upravljanje istim. Za ovaj seminarski rad izabrana je biblioteka imena TextLCD autora cristian vega, dostupna na poveznici: https://os.mbed.com/users/cristianve/code/TextLCD/. Prethodno korištenju biblioteke potrebno ju je dodati u projekt putem web sučelja Mbed Compiler i potom naznačiti u samome kodu da se ista koristi.

```
3       #include "TextLCD.h"
```

Po izvršenju prethodnog, potrebno je postaviti metodu komunikacije između LCD-a i NUCELO-a na način da se definira ime kanala komunikacije i označe fizički izlazi koji će se koristiti (oznake I2C_SDA i I2C_SCL upućuju na sučelje broj 1 koje se nalazi na izlazima D14 i D15).

```
21      I2C i2c_lcd(I2C_SDA, I2C_SCL);
```

Potom je za prije spomenutu biblioteku potrebno napraviti klasu imena lcd koja kao ulazne parametre uzima: I2C sučelje za komunikaciju s LCD-om, I2C adresu na kojoj se nalazi LCD i tip ekrana.

```
40      TextLCD_I2C lcd(&i2c_lcd, 0x4E, TextLCD::LCD16x2); //PCF8574
```

Unutar glavne (eng. main) funkcije obavlja se inicijalizacija I2C protokola i LCD biblioteke prema uputama autora. U I2C komunikaciji koristi se tzv. fast-mode brzina (frekvencija) komunikacije kako bi se smanjio treptaj numeričkog ekrana prilikom izvođenja funkcije koja je zadužena za osvježavanje LCD-a.

```
274         i2c_lcd.frequency(400000);
```

Inicijalna konverzija podataka obavlja se u slučaju da je RTC (eng. Real Time Clock) spojen na bateriju, te je nastavio brojati vrijeme čak i nakon gašenja samog NUCLEO-a.

```
277         EPOCH_and_ALARM();
278
```

Za unaprjeđenje estetskoga dojma i funkcionalnosti prikaza podataka na LCD-u isključen je prikaz kursora i uključeno je pozadinsko osvjetljenje.

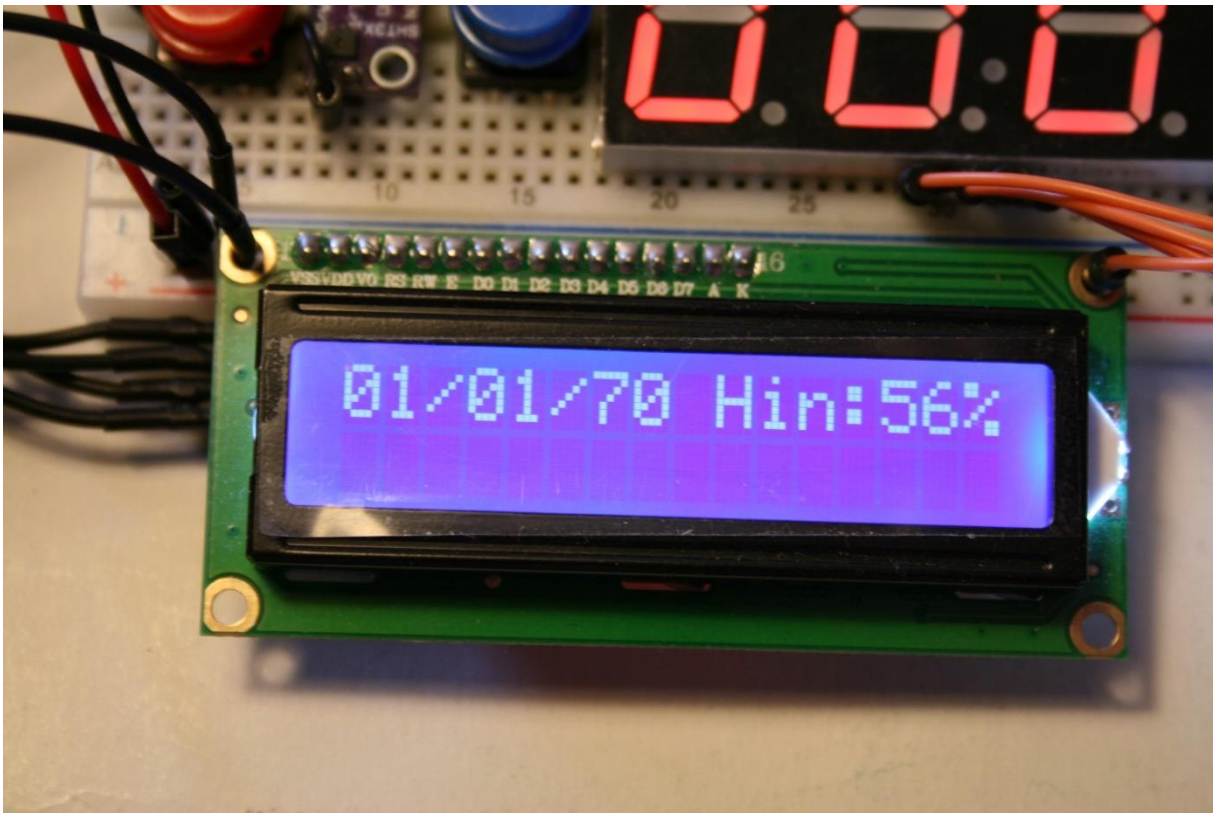
```
280         lcd.setCursor(TextLCD::CurOff_BlkOff);
281         lcd.setBacklight(TextLCD::LightOn);
```

Nakon inicijalizacije LCD-a poželjno je dostaviti podatke putem I2C protokola kako bi se po samome uključenju ovoga sustava na ekranu prikazale informacije. Za prikaz informacija na LCD-u napravljene su dvije funkcije koje su detaljno obrađene u nastavku ovoga seminarskog rada, no potrebno je napomenuti kako je datum (podatak koji se mijenja samo jednom dnevno) izoliran u zasebnu funkciju za potrebe smanjenja količine podataka, a samim time i vremena koje je potrebno I2C komunikaciji za

prijenos podataka na LCD (ponovno zbog smanjenja treptaja numeričkog ekrana prilikom izvođenja funkcije).

```
283         display_main();
284         update_date();
```

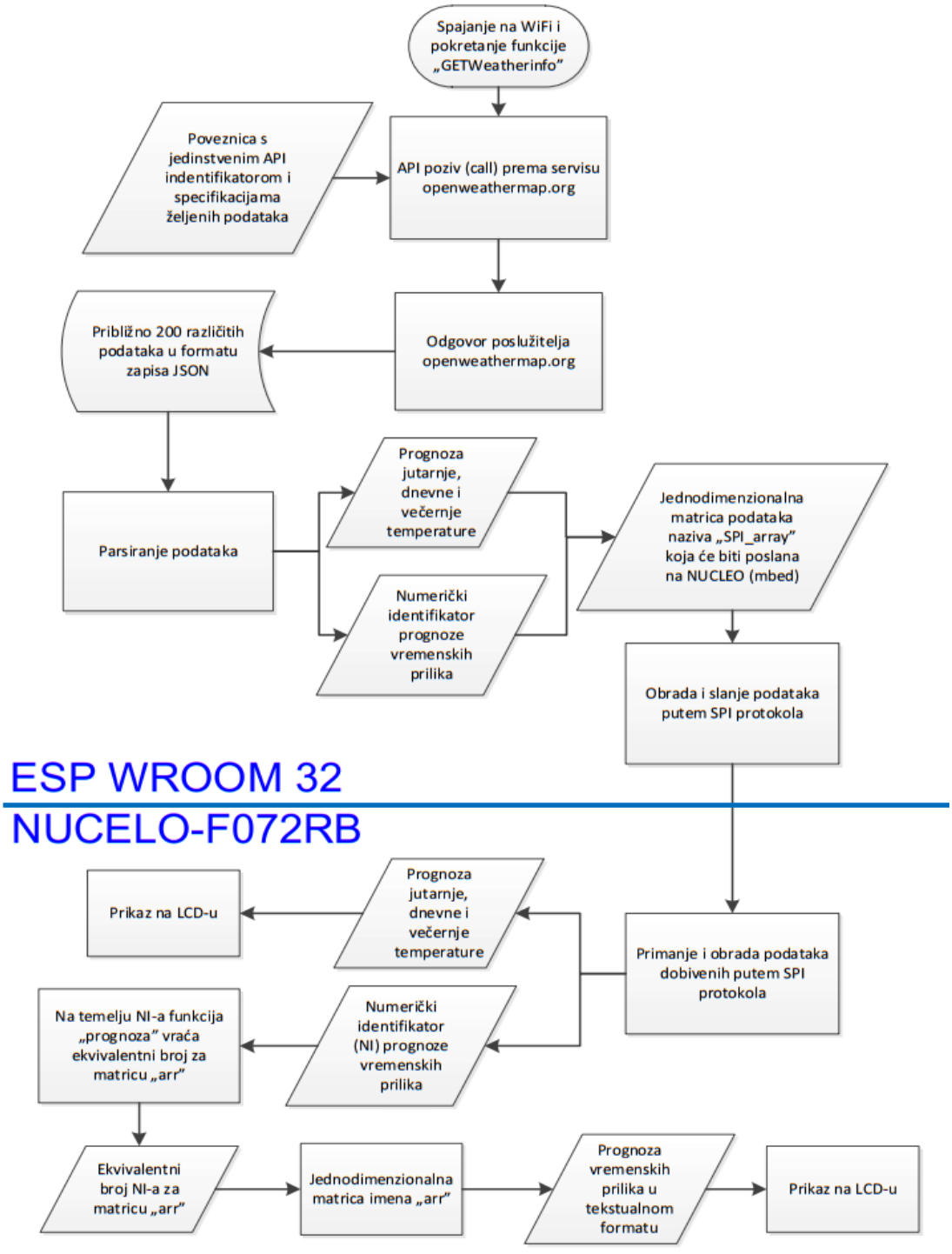
Ovime je završena inicijalizacija LCD-a te je primjer izgleda ekrana moguće vidjeti na slici ispod (donji red ekrana biti će popunjen sadržajem nakon desetak sekundi, detaljnije u nastavku).



Slika 3-1 Snimak ekrana poslije inicijalizacije

3.1 Dobavljanje podataka za prikaz

Za pribavljanje podataka o prognozi vremena s interneta bilo je potrebno napraviti široko istraživanje i napredniji algoritam, te je iz toga razloga ovaj dio koda izdvojen u zasebnu cjelinu. Naime, kako bi se lakše i smislenije opisao proces dobavljanja podataka za prikaz prognoze potrebno ga je detaljno opisati. Kao uvod u ovu problematiku preporuča se proučiti dijagram toka na slici 3.1-1 koja okvirno predstavlja cjelokupni proces.



Slika 3.1-1 Dijagram toka dobavljanja podataka

Kao što je vidljivo iz dijagrama toka, prvotno je potrebno mikrokontroler ESP-WROOM-32 (u daljnjem tekstu ESP32) spojiti na internet (u ovome slučaju bežičnom 2.4GHz mrežom) i potom pokrenuti funkciju GETWeatherinfo (na ESP32) koja je tipa void i ne prima ulazne parametre. Programski kôd za spajanje na internet putem bežične mreže je približno jednak standardiziranom kôdu za isto, te ga zbog toga nema smisla analizirati (dostupan na kraju seminarskog rada u kompletnom kôdu). Funkcija GETWeatherinfo pokreće se u dva različita slučaja kao i sve druge funkcije. Prvi slučaj je kada se pokrene funkcija void loop (jedna od dvije osnovne funkcije u Arduino programskom jeziku), dok je drugi slučaj napravljen za osvježavanje podatka na zahtjev korisnika (bilo je potrebno pri razvijanju samoga sustava, no ostavljeno je kao korisna značajka za krajnjeg korisnika). U prvom isječku programskog kôda primjetna je aktivacija funkcije za dohvaćanje podataka (eng. get) iz različitih područja, te se na samome kraju ti isti podaci šalju putem SPI protokola na NUCLEO mikrokontroler pozivom funkcije SPI_send. Ovaj autonomni proces odvija se svakih sat vremena odnosno 3600000 milisekundi.

```
285   void loop(){
286     GETWeatherinfo(); //funkcija za dohvacanje prognoze
287     GETAlarm(); //funkcija za dohvacanje posatavljenog alarma
288     GETLocalTime(); //funckija za dohvacanje trenutnog vremena
289     SPI_send(); //funkcija za slanje podataka na NUCELO
290     delay(3600000);
291   }
```

Sljedeći programski isječak je smješten u void setup funkciji i prikazuje relevantni dio kôda za osvježavanje podataka pritiskom fizičkog tipkala. Potrebno je napomenuti kako je ova funkcionalnost izvedena putem prekidne funkcije (eng. interrupt) koja na pinu broj 14 detektira padajući brid (na istoimeni pin spojen je pull-up otpornik, za više informacija pogledati shemu na slici 1.1-2).

```
278         attachInterrupt(14, update_ALL, FALLING);
```

Prije samog pokretanja funkcije GETWeatherinfo potrebno je pozvati biblioteke za: povezivanje na bežičnu mrežu, provjeru dostupnosti internetske veze („pinganje“), klijenta HTTP protokola (dohvaćanje podataka) i JSON raščlanjivač (izolacija korisnih podataka iz odgovora poslužitelja s kojeg se dohvaća vremenska prognoza).

```
001   #include <WiFi.h>
002   #include <ESP32Ping.h>
003   #include <HTTPClient.h>
004   #include <Arduino_JSON.h>
```

Nakon pozivanja biblioteka potrebno je postaviti varijablu koja u sebi sadrži poveznicu poslužitelja OpenWeather servisa od kojeg će se preuzeti podaci. S obzirom da istoimeni servis ima pregršt različitih paketa podataka i načina na koje će iste strukturirati, bilo je potrebno modificirati poveznicu kako bi dobili relevantne podatke. Modificirani dijelovi poveznice su: vrsta upita prema servisu, geografska širina i dužina područja (eng. latitude and longitude), označavanje neželjenih kategorija podataka (u svrhu smanjenja potrošnje internetskog prometa i memorije mikrokontrolera), jedinstveni ključ (eng. API key) i vrsta mjernih jedinica (npr. temperatura u °C). Za vrstu upita prema servisu odabran je model onecall (hrv. jedan poziv) s obzirom da nije poželjno dohvaćati periodičnu prognozu vremena, već samo u trenutku kada je pokrenuta funkcija. Geografska širina i dužina su postavljene na Grad Zagreb, te su uzete vrijednosti koje servis preporuča da se koriste (širina 15,978°, dužina 45,8144°). Kako bi se komunikacija optimizirala iskorištena je funkcija exclude (hrv. isključi, izdvoji) koja je eliminirala trenutačne vremenske prilike, minutnu i satnu prognozu (ove informacije nisu od interesa zbog limitirajućeg faktora veličine ekrana). Jedinstveni ključ (eng. API key) je nužan parametar koji servisu omogućava raspoznavanje klijenata i količinu podataka koju koriste (ograničen je broj poziva u minuti

i mjesecu). Za mjerne jedinice uzet je metrički sustav s obzirom da se isti koristi na području Republike Hrvatske.

```
013   const String poveznica =
"https://api.openweathermap.org/data/2.5/onecall?lat=45.8144&lon=15.978&exc
lude=current,minutely,hourly&appid=e5941c3f91ab77e9ac98e22589d3c7ba&units=m
etric";
```

Kako bi se dobiveni i potom obrađeni podaci mogli spremiti u matricu za slanje, istu je potrebno prethodno inicijalizirati.

```
023   int SPI_array[7];
```

Kao što je već prije napomenuto, funkcija GETWeatherinfo je tipa void i ne prima ulazne parametre. Na početku same funkcije se provjerava dostupnost internetske veze. U slučaju da ista nije dostupna, potrebno je ponovno pokušati spojiti ESP32 na bežičnu mrežu (ako mreža na koju se ESP32 spojio nema izlaz prema internetu, mikrokontroler će se od spojiti i spojiti natrag na mrežu, dok mu veza ne postane dostupna). Za razliku od prijašnjeg, u slučaju kada je veza dostupna ulazi se u kôd unutar if petlje.

```
207   void GETWeatherinfo(){
208         check_connection();
209         if(ping_res == true) {
```

Podaci koji se primaju od servisa za vremensku prognozu (u daljnjem tekstu OW servis) bit će potrebno prvotno pohraniti u varijablu. S obzirom da u Arduino jeziku postoji tip varijable string, ona je odabrana zbog velike količine različitih podataka koji će se dohvatiti s OW servisa.

```
210           String jsonBuffer;
```

Nakon inicijalizacije varijable slijedi poziv funkcije za komunikaciju putem HTTP protokola koja izvršava zahtjev prema OW servisu putem poveznice iz reda broj 13. Funkcija imena httpGETRequest kao ulazni podatak uzima poveznicu na poslužitelja s kojeg je potrebno dohvatiti podatke. Prethodno navedena funkcija detaljnije je objašnjena u poglavlju 3.1.1. Odgovor OW servisa sprema se u varijablu imena jsonBuffer.

```
211           jsonBuffer = httpGETRequest(poveznica.c_str());
```

Kako bi se podaci dobiveni iz odziva s OW servisa mogli raščlaniti, potrebno je za biblioteku Arduino_JSON označiti podatke koji su zapisani u JSON obliku.

```
213           JSONVar myObject = JSON.parse(jsonBuffer);
```

Kako bi se izbjegla obrada podataka neispravne strukture vrši se provjera istih kroz JSON raščlanjivač koji ih može označiti kao nedefinirane (eng. undefined), te s time prekinuti daljnju manipulaciju podatcima.

```
216           if (JSON.typeof(myObject) == "undefined") {
218               return;
219           }
```

Za svrhu manipulacije podacima koji su nam korisni potrebno je prvo upoznati se s dvije osnovne strukture u JSON formatu zapisa podataka: objekti i matrice. Objekti se označavaju vitičastim zagradama, a matrice uglatim zagradama. Reprezentativan kôd je dan u nastavku.

```
{
        "Naslov": "Seminarski rad",
        "Autori": [
          { "Josipović": "0246083635" },
          { "Marinić": "0246085944" }
        ]
}
```

Stvarni odziv OW servisa za reprezentativan datum 18.04.2021. prikazuje standardnu strukturu podataka. Dva isječka odziva prikazuju podatke korisne za ovaj seminarski rad (prognoza temperatura i numerička oznaka prognoze vremenskih prilika). Podaci koji su konkretno potrebni za ovaj seminarski rad su na linijama broj 15, 19, 20 i 36. S obzirom da cjelokupni odgovor OW servisa sadrži 276 redova nije prikladno prikazati ga u cijelosti, već je u nastavku prikazan isječak relevantan za ovaj projekt.

```
001   {
002      "lat":45.8144,
003      "lon":15.978,
004      "timezone":"Europe/Zagreb",
005      "timezone_offset":7200,
006      "daily":[
007           {
008                "dt":1618740000,
009                "sunrise":1618718676,
010                "sunset":1618767948,
011                "moonrise":1618731600,
012                "moonset":1618701240,
013                "moon_phase":0.19,
014                "temp":{
015                     "day":6.86,
016                     "min":5.47,
017                     "max":6.86,
018                     "night":5.68,
019                     "eve":6.39,
020                     "morn":5.47
021                },

034                "weather":[
035                     {
036                        "id":500,
037                        "main":"Rain",
038                        "description":"light rain",
039                        "icon":"10d"
040                     }
041                ],
```

Kako bi se potrebni podaci mogli izolirati kao zasebne varijable, potrebno je prvotno analizirati strukturu u kojoj se nalaze. Naime, temperature se nalaze u objektu imena „temp“ koji je nulti zapis u matrici koja se nalazi u objektu imena daily. U nastavku kôda vidljivo je na koji način su temperature izdvojene i pospremljene u dugačku cjelobrojnu varijablu (eng. long int). Ovaj format izabran je primarno iz razloga što biblioteka Arduino_JSON ima vrlo mali broj podržanih formata, no također i za eliminaciju decimalnih mjesta koja su funkcionalno nepotrebna, a također su i nepraktična zbog ograničenosti LCD-a. Nakon što su potrebni podaci izolirani, isti se spremaju u jednodimenzionalnu matricu koja je globalni spremnik za podatke koji će se slati putem SPI protokola na NUCLEO mikrokontroler.

```
227           //TEMPERATURE
228           long int temp_morn=myObject["daily"][0]["temp"]["morn"];
229           SPI_array[0] = temp_morn;
231
```

```
232             long int temp_day=myObject["daily"][0]["temp"]["day"];
233             SPI_array[1] = temp_day;
235
236             long int temp_eve=myObject["daily"][0]["temp"]["eve"];
237             SPI_array[2] = temp_eve;
```

Osim temperatura, također je potrebno izdvojiti podatak koji se nalazi na liniji broj 36 u odgovoru OW servisa. Taj podatak nam govori kakve će biti vremenske prilike za sljedeći dan, no u obliku gdje svaka vrsta vremenskih prilika ima svoj brojčani ekvivalent (prema tablici koja se nalazi na poveznici https://openweathermap.org/weather-conditions#Weather-Condition-Codes-2).

```
240             //VREMENSKI UVJETI
241             long int weather_id=myObject["daily"][0]["weather"][0]["id"];
242             SPI_array[3] = weather_id;
```

Kao što je i prije bilo napomenuto, u slučaju da ESP32 ne može pristupiti poslužitelju, pokušati će se ponovo spojiti na internet i pokrenuti funkciju iznova.

```
246           }else{
248                   wifiConnect();
249                   GETWeatherinfo();
250            }
251
252     }
```

Nakon završetka funkcije GETWeatherinfo pokreću se funkcije za dohvaćanje alarma i stvarnog vremena, te se time završava cjelokupni proces dobavljanja podataka s interneta. Prije pokretanja funkcije SPI_send, jednodimenzionalna matrica imena SPI_array je popunjena podacima o temperaturama i vremenskim prilikama uz pratnju ostalih podataka iz druge dvije funkcije. Funkcija SPI_send i sve povezano sa SPI protokolom pokriveno je u poglavlju broj 4.

3.1.1 Funkcija za dohvaćanje podataka s OW servisa

Kako bi se podatci koji se dobiju od OW servisa mogli dohvatiti, potrebno je ostvariti komunikaciju s OW poslužiteljem putem HTTP protokola. S obzirom da se radi o komunikaciji s internetskim servisom, funkcija za dohvaćanje podataka s OW servisa zahtjeva internetsku vezu. Iz toga razloga potrebno je za istu uspostaviti vezu prema internetu, odnosno poslužitelju, putem bežičnog sučelja za komunikaciju.

Pošto funkcija dohvaća podatke s prije spomenutog servisa, potrebno je inicijalizirati biblioteku za komunikaciju putem HTTP protokola.

```
003    #include <HTTPClient.h>
```

Funkcija za dohvaćanje podataka je tipa string i kao parametar za rad zahtjeva poveznicu (ili IP adresu) poslužitelja s kojeg se žele dohvatiti podaci. Za ulazni parametar se iz funkcije GETWeatherinfo prosljeđuje varijabla imena poveznica (red 13) u obliku stringa.

```
177    String httpGETRequest(const char* serverName){
```

Kako bi se funkcija mogla koristiti, prvotno je potrebno stvoriti prototipnu funkciju HTTP klijenta pod imenom http koji će kasnije biti korišten za komunikaciju putem istoimenog protokola.

```
178          HTTPClient http;
```

Nakon inicijalizacije klijenta započinje se komunikacija s OW servisom, odnosno sa specifičnom stranicom generiranom za ovaj projekt čija se poveznica nalazi u retku 13 i proslijeđena iz funkcije GETWeatherinfo u ovu funkciju pod imenom serverName

```
181          http.begin(serverName);
```

Naime, podatke je potrebno preuzeti na sami ESP32 mikrokontroler, te se iz tog razloga poziva funkcija za dohvaćanje podataka putem HTTP protokola metodom POST imena GET. Za istu je inicijalizirana cjelobrojna varijabla httpResponseCode u koju će se po završetku dohvaćanja podataka spremiti kôd uspješno obavljene komunikacije (200) ili greške (npr. 403, 404, 500, 503). Ako nije dobiven odgovor poslužitelja, u prije spomenutoj varijabli će ostati upisana vrijednost nula.

```
184          int httpResponseCode = http.GET();
```

Za dohvaćene podatke stvorena je varijabla imena payload tipa string koja je odabrana obzirom na veliku raznolikost i veličinu podataka koji će biti pohranjeni u istu.

```
186          String payload = "{}";
```

Prije pohranjivanja podataka provjerava se da li je komunikacija s poslužiteljem bila uspostavljena provjerom kôda komunikacije koji je spremljen u varijablu imena httpResponseCode.

```
188          if (httpResponseCode>0) {
```

U slučaju kada je vrijednost prije spomenute varijable veća od nule, dohvaćeni podaci pohranjuju se u varijablu imena httpResponseCode.

```
191            payload = http.getString();
192          }
```

U protivnome, kada uvjet unutar if funkcije nije zadovoljen, vrši se ponovni pokušaj spajanja na bežičnu mrežu i potom ponovno aktiviranje funkcije.

```
193          else {
```

```
197             wifiConnect();
198             httpGETRequest(poveznica.c_str());
199           }
```

Naposljetku se deinicijalizira funkcija za komunikaciju putem HTTP protokola kako bismo oslobodili resurse za druge funkcije.

```
201           http.end();
203           return payload;
204     }
```

3.2 Prikaz podataka na 16x2 LCD-u

Kao što je bilo prije navedeno, da bi se na LCD prikazale informacije, inicijalno zovemo dvije funkcije (display_main i update_date) u main funkciji. Prva funkcija imena display_main je tipa void i ne prima ulazne parametre. Za pravilno funkcioniranje iste, prvotno je potrebno inicijalizirati nekolicinu jednodimenzionalnih matrica koje predstavljaju moguće vremenske uvjete i imena doba dana za prognozirane temperature (svi ovi podaci prikazivati će se na engleskome jeziku s obzirom da u većini slučajeva imaju kraće nazive). Dodatni karakter za LCD i pomoćne varijable koje će biti upotrjebljene u nastavku su također inicijalizirani u ovome dijelu, te su detaljnije objašnjeni u nastavku ovoga rada.

```
43    char arr[20][30] = {
44          "No info","Storm", "Drizzle", "Rain", "Snow",
45          "Mist", "Smoke", "Haze", "Dust", "Fog",
46          "Sand", "Ash", "Haze", "Squall", "Tornado",
47          "Clear", "Clouds"
48    };
49
50    char day_state[3][4] = {"MOR","DAY","EVE"};
51
52    char stupnjevi=0b11011111; //binarni niz za prikaz „°“ na LCD-u
53    int looping=0;
54    bool trg = 0, info=0;
```

Na samome početku funkcije display_main (tipa void i ne prima ulazne parametre) primjetno je povećanje vrijednosti varijable looping za jedan, po svakoj aktivaciji funkcije. Varijabla se koristi za izmjenu prikaza triju temperaturnih stanja na LCD-u (jutarnja, dnevna i večernja temperaturna prognoza) kroz matricu day_state, te nakon inkrementa slijedi if funkcija koja varijablu vraća na početno stanje 0 jednom kada dođe do numeričke vrijednosti 3, kako bi se proces ponavljao.

```
240   void display_main(){
241         looping++;
242         if(looping==3){
243               looping=0;
244         }
245
```

Pomoćna varijabla trg služi za izmjenu prikaza informacija na LCD-u. Dvije osnovne izmjene na LCD-u su između prikaza sobne temperature i relativne vlage zraka, s time da se uz prikaz vlage veže komunikacija sa senzorom SHT30 (objašnjen u poglavlju broj 5).

```
246         trg = !trg;
```

Za slučaj kada se na LCD-u prikazuje izmjerena sobna temperatura (trg=1), također se i prikazuju (odnosno izmjenjuju) podaci o prognozi vremena i temperaturi (varijabla looping je limitirana na cijele brojeve 0, 1 i 2, te se njena vrijednost upotrebljava kako bi se iz jednodimenzionalnih matrica imena day_state i SPI_array dobili podaci o jutarnjoj, dnevnoj ili večernjoj temperaturnoj prognozi). Detaljnije informacije na koji način se izvodi dohvaćanje prognoze vremenskih prilika i temperatura objašnjeno je u poglavlju 3.1.1.

```
248         if(!trg){
```

Naredba lcd.locate služi za pozicioniranje kursora na LCD-u, konkretnije jedno mjesto dalje od datuma (koji zauzima stupce [0, 7]) u istome redu za potrebe ispisa vrijednosti sobne temperature i vlage.

```
249               lcd.locate(9,0);
```

Za ispis trenutačne sobne temperature poziva se prije definirana biblioteka za komunikaciju s LCD-om imena lcd s funkcijom unutar sebe imena printf, koja za svoje ulazne parametre prihvaća tekst s mogućnošću uvrštavanja varijabli. U ovom primjeru ispisan je tekst „Tin: “, nakon čega dolazi vrijednost varijable temp, te potom prije kreirani simbol za stupnjeve (°).

```
250                 lcd.printf("Tin:%.2d%c", temp, stupnjevi);
```

Po završetku prethodne linije kôda, kursor se pozicionira u nulti stupac sljedećeg reda (red broj jedan).

```
252                 lcd.locate(0,1);
```

Logička jedinica unutar varijable imena info označava dostupnost podataka koji su dobiveni od ESP32 mikrokontrolera, što će pokrenuti dio kôda unutar if funkcije.

```
253                 if(info){
```

U redu broj 254 ponovno se poziva funkcija ispisa teksta na ekran koja unutar sebe sadrži format ispisa triju varijabli. Prva varijabla, koja predstavlja ime dijela dana, ispisati će se na mjestu „%s“, a sadrži podatak iz matrice day_state koji je selektiran ovisno o looping varijabli (ovime je postignuta promjena prikaza prognozirane temperature prilikom svakog pokretanja ovoga dijela kôda). Potom slijedi separator u obliku dvotočke koji odvaja prethodni ispis od sljedećeg u kojem se iz matrice SPI_array uzima ekvivalentna prognozirana temperatura za prethodno ispisano doba dana. Na posljetku se ispisuje simbol za stupnjeve koji je potreban s obzirom da se radi o temperaturama.

```
254                     lcd.printf("%s:%.2d%c  ", day_state[looping],
SPI_array[looping], stupnjevi);
```

Kako bi se izbjegao utjecaj duljine prethodnog ispisa na sljedeći, stavljena je naredba za ponovno pozicioniranje kursora na mjestu ispisa nove informacije.

```
256                     lcd.locate(9,1);
```

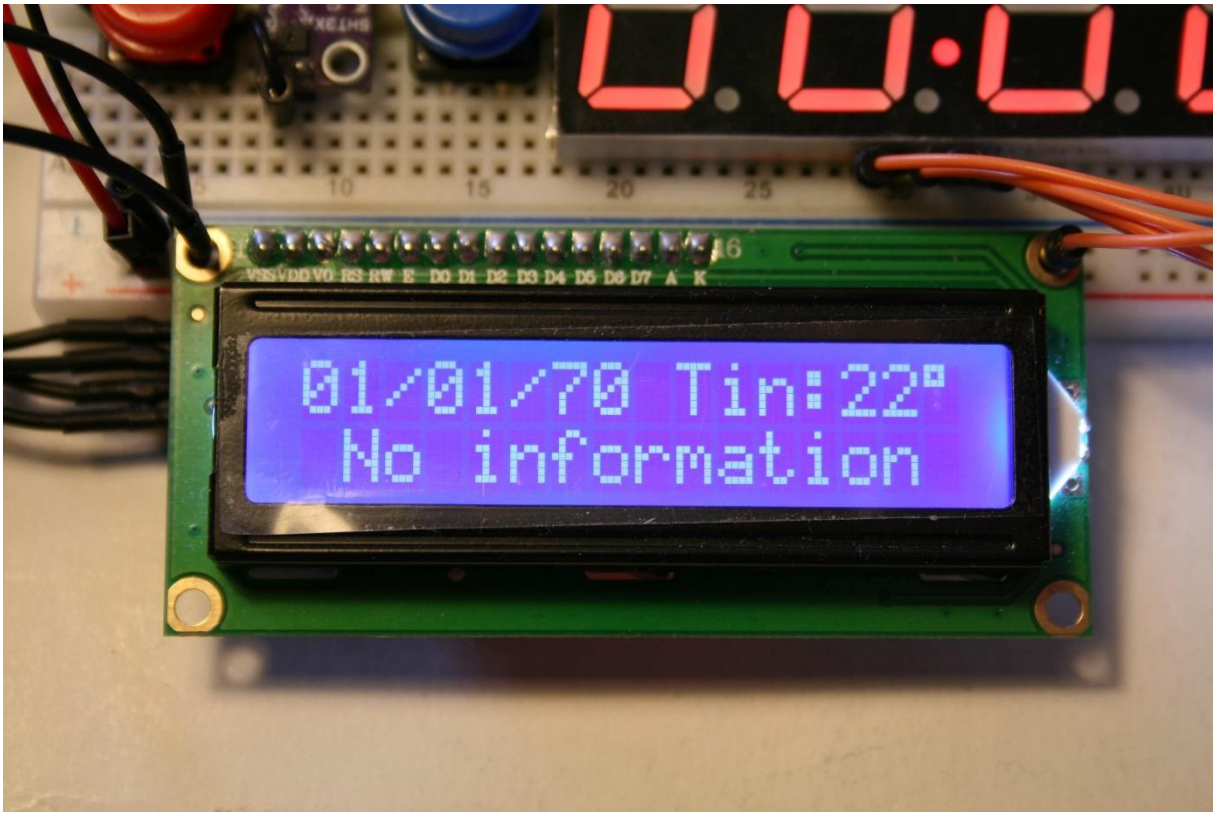
Funkcija imena prognoza (linije koda [213, 231]), kada pozvana, uzima podatak iz matrice SPI_array (polje br. 3, oznaka prognoze vremenskih prilika), te ga tehnikom „branchless programing“ sortira u prigodnu kategoriju za koju vraća cijeli broj (područje [0, 20]). Isti se potom uzima kao redni broj podatka iz matrice arr koja vraća prognozu vremenskih prilika u tekstualnom obliku, koji se potom direktno prosljeđuju za prikaz na LCD-u u donjem desnom kutu na poziciji od [9, 1] do najviše [15, 1](ovisno o duljini riječi).

```
257                     lcd.printf("%s", arr[prognoza()]);
```

Slika 3.2-1 Snimak ekrana s prikazom datuma, sobne temperature, prognozom vanjske temperature i vremenskih prilika

Također je potrebno napomenuti kako je implementirana sigurnosna funkcija u slučaju greške (eng. fail-safe), koja u slučaju kada se nije dogodio prijenos podataka sa ESP32 mikrokontrolera na LCD-u ispiše poruku „No information“ (hrv. nema informacija). Ovime se izbjegao prikaz neželjenih podataka na LCD-u u redu broj jedan (datum i podaci s SHT30 nisu zahvaćeni ovime).

```
258                 }else{
259                       lcd.printf(" No information ");
260                 }
```



Slika 3.2-2 Snimak ekrana za slučaj kada nisi dostupni podaci o prognozi i trenutačnom vremenu

U slučaju kada „if(!trg) “ funkcija nije zadovoljena, pokreće se funkcija za mjerenje sobne temperature i vlage senzorom SHT30 (opisano u poglavlju broj 5). Potom se novo dobivena relativna vrijednost vlažnosti zraka prikazuje na LCD-u do ponovnog pokretanja funkcije display_main, prilikom čega se prikaz mijenja na vrijednost temperature zraka.

```
262           }else{
263                 SHT30();
264                 lcd.locate(9,0);
265                 lcd.printf("Hin:%.2d%%", hum);
266
267           }
268     }
```

Funkcija display_main se osim u inicijalizaciji također poziva i iz funkcije prikaz svake 2000. iteracije te funkcije (odnosno približno svakih 10 sekundi) kako bi se izmijenili podaci koji se prikazuju na LCD-u (sobna temperatura i vlaga). Izvadak kôda iz funkcije prikaz je ponovno dan u nastavku kako bi se lakše analizirao programski tok.

```
79          izmjena++;
```

```
80          if(izmjena == 2000){
81                display_main();
82                izmjena = 0;
83          }else{
84                wait_us(4950);
85          }
86
```

Druga funkcija koja se inicijalno zove tijekom izvršavanja glavne funkcije (eng. main) vezana uz popunjavanje LCD-a s informacijama naziva se update_date. Funkcija je tipa void i ne prima ulazne parametre, te obavlja trivijalni posao postavljanja datuma u obliku dan/mjesec/kratka godina (eng. dd/mm/yy) na mjesta od [0, 0] do [8, 0], kao što je vidljivo i u prototipnoj verziji na 3.2-2. Ovaj dio koda napravljen je kao zasebna funkcija kako bi se dodatno optimizirao sustav, odnosno izbjeglo bespotrebno osvježavanje datuma svakih 20 sekundi, s obzirom da je isto potrebno samo jednom dnevno (odnosno 4320 puta manje).

```
234   void update_date(void){
235         lcd.locate(0,0);
236         lcd.printf("%.2d/%.2d/%.2d", day, month, year);
237   }
```

Za osvježavanje datuma svakodnevno u ponoć implementirana je if funkcija u while petlji unutar main funkcije koja u slučaju kada se ne odvija prijenos podataka putem SPI komunikacije vrši provjeru vrijednosti varijabli second, minute i hour, te kada su iste jednake nuli pokreće funkciju update_date (s obzirom da funkcija urica uvijek prethodi update_date funkciji u else dijelu koda i u sebi sadrži EPOCH_and_ALARM, sve varijable koje predstavljaju vrijeme su ažurirane i nije isto potrebno ponovo izvoditi u funkciji update_date ili prije nje za if funkciju).

```
286         while(1){
287               if(spi_port.receive()){
288                     SPI_receive();
289               }else{
290                     urica();
291
292                     if(second == 0 && minute == 0 && hour == 0){
293                           update_date();
294                     }
295               }//else
296         }//while(1)
```

4. SPI protokol i njegova implementacija

Serijsko periferno sučelje (eng. Serial Peripheral Interface, u daljnjem tekstu SPI) je protokol razvijen od strane tvrtke Motorola 1979. godine koji je s vremenom postao jedan od najčešće korištenih komunikacijskih protokola, primarno kod ugradbenih računalnih sustava. SPI pripada protokolima serijske komunikacije koji omogućuju istovremenu dvosmjernu komunikaciju (eng. full-duplex). Za ovaj projekt SPI protokol je odabran za prijenos podataka između dva mikrokontrolera, konkretno između NUCLEO-F072RB i ESP WROOM 32.

Izbor samoga protokola je dogovor između autora ovog seminarskog rada na temelju preferencije izbjegavanja UART protokola zbog svoje trivijalnosti i I2C protokola koji je bio potreban za senzor SHT30 i LCD ekran (uvjetovano pravilima kolegija „Komunikacijske tehnike u mehatronici“ koji nalaže da autori istog seminarskog rada ne mogu koristiti isti komunikacijski protokol). S obzirom da SPI protokol radi na principu glavnog i zavisnog (sporednog) komunikacijskog uređaja (eng. master and slave), bilo je potrebno odlučiti koji od mikrokontrolera će biti glavni, a koji zavisni. Zbog programskih nedostataka mikrokontrolera ESP32 u ulozi zavisnog člana komunikacije, odlučeno je da će konfiguracija biti takva da je ESP32 glavni komunikacijski uređaj, dok je NUCLEO-F072RB (u daljnjem tekstu NUCELO) zavisni komunikacijski uređaj. Za informacije kako je ostvarena veza između NUCLEO-a i ESP32 preporuča se konzultirati elektroničku shemu na slici 1.1-2.

4.1 Glavni komunikacijski uređaj (ESP32)

ESP32 ima funkciju glavnog komunikacijskog uređaja (eng. master), stoga je pozvana biblioteka imena „SPI“.

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

Za ispravno funkcioniranje SPI protokola prethodno je potrebno inicijalizirati varijable koje u sebi sadrže: brzinu komunikacije, aktivaciju jednog od dva sučelja za SPI i jednodimenzionalnu matricu za pohranjivanje podataka koji će biti poslani. Varijabla imena SPI_frekvencija ima vrijednost 1125000 što je 1.125 MHz, a definira frekvenciju komunikacije (tehničko ograničenje NUCLEO platforme na upravo tu frekvenciju). Mikrokontroler ESP32 na sebi posjeduje dva sučelja za komunikaciju putem SPI protokola, a to su vspi i hspi. Vspi sučelje odabrano je iz praktičnih razloga smještaja samoga sučelja na razvojnoj pločici. Posljednja globalna varijabla je jednodimenzionalna matrica SPI_array koja služi kao privremeni spremnik u koji sve funkcije spremaju podatke za slanje na NUCLEO.

```
020   //SPI
021   static const int SPI_frekvencija = 1125000;
022   SPIClass * vspi = NULL;
023   int SPI_array[7];
```

Nakon inicijalizacije varijabli slijedi inicijalizacija samoga SPI protokola. Za prethodno je potrebno stvoriti SPI klasu u kojoj se označava sučelje koje želimo (konkretno za slučaj ovog seminarskog rada vspi), potom pokrenuti inicijalizaciju samog sučelja i na kraju inicijalizirati jedan od izlaza koji ćemo koristiti za označavanje zavisnog komunikacijskog uređaja (eng. slave select).

```
273         //SPI inicijalizacija
274         vspi = new SPIClass(VSPI);
275         vspi->begin(); //SCLK = 18, MISO = 19, MOSI = 23, SS = 5
276         pinMode(5, OUTPUT); //VSPI SS
```

Inicijalizirani sustav je spreman za dalje operacije vezane uz slanje podataka putem SPI protokola. Kronološki se nakon inicijalizacije odvijaju druge funkcije primarno vezane uz dobavljanje podataka, te po završetku sve tri funkcije (dohvaćanje vremenske prognoze, alarma i trenutačnog vremena) se pokreće funkcija SPI_send.

```
281   void loop(){
282     GETWeatherinfo();
283     GETAlarm();
284     GETLocalTime();
285     SPI_send();
286     delay(3600000);
```

Funkcija „SPI_send“ je tipa void i ne prima ulazne parametre, već su sve potrebne varijable globalne ili se stvaraju lokalno u samoj funkciji.

```
081   void SPI_send(void){
```

S obzirom da se putem SPI protokola tipično šalje najviše 8 bitova po jednom prijenosu, odabran je tip jednodimenzionalne karakterne matrice (eng. char) zbog svoje ograničenosti na isti broj bitova. Jednodimenzionalna matrica imena data_out načinjena je u svrhu privremene pohrane podataka podijeljenih na po 8 bitova prije slanja na NUCLEO. Podatke je bilo moguće razdvajati unutar same petlje, gdje se obavlja slanje podataka, no prema iskustvu iz prakse, ovakva metoda separacije nije preporučljiva zbog sporije komunikacije s NUCLEO-m.

```
082     char data_out[16];
```

Varijabla imena place potrebna je za praćenje pozicije upisa separiranih podataka na za to predviđena mjesta unutar data_out matrice.

```
083     int place=0;
```

S obzirom da svi podaci nisu jednake veličine, napravljene su dvije odvojene for petlje kako bi se separirali. U prvoj petlji se obrađuju podaci vezani uz prognozu temperatura koji su tipično cijeli brojevi u području [-20, +40], te je samim time i više nego dovoljno 8 bitova, zbog čega se direktno pohranjuju na prva tri mjesta u data_out matrici. Kao što je bilo i prije naglašeno, varijabla place služi kao oznaka mjesta unutar matrice na koje ćemo spremiti podatke, te se povećava za vrijednost jedam sa svakim prolazom kroz funkciju. Varijabla imena i inicijalizirana je lokalno unutar for petlje kako bi njome označili koji podatak unutar SPI_array matrice petlja obrađuje.

```
086     for(int i=0; i<3; i++){
087         data_out[place] = SPI_array[i];
088         place++;
089     }
```

Sljedeća tri podatka za obradu su: numerički identifikator prognoze vremenskih prilika, alarm u EPOCH formatu i trenutačno vrijeme u EPOCH formatu. Sva tri podatka zahtijevaju do 4 bajta podataka, te su zbog toga odvojeni u zasebne for petlje koje će ih separirati u dijelove po 8 bitova, odnosno 1 bajt. Prva for petlja izvršava funkciju označavanja podataka unutar SPI_array matrice koji se nalaze na mjestima 3, 4 i 5. Kako bi se dobile upravo te tri vrijednosti, stavljeni su uvjeti koji specificiraju varijablu imena i, koja se stvara lokalno unutar funkcije, te kreće od vrijednosti 3. Također je postavljen uvjet da se funkcija ponavlja dok varijabla imena i ne bude jednaka ili veća od 6. Naposljetku je stavljen automatski inkrement varijable i za svaki prolazak kroz petlju. Druga for petlja separira svaku od tri varijable na 4 dijela po 8 bitova (tj. na 4 zasebna bajta). Podatak koji se obrađuje određuje varijabla i, kao što je već prethodno objašnjeno. Potom se taj podatak u binarnom ekvivalentu pomiče za 8*t mjesta u desno, kako bi prvo dobili LS bajt, a na kraju MS bajt. S obzirom da u privremenu matricu data_out želimo spremiti samo 8 bitova, potrebno je dobiveni podatak logičkom operacijom I skratiti na veličinu 0xFF (heksadekadski FF, binarno 11111111), odnosno 8 bitova. Obrađeni podatak se upisuje u prvo slobodno mjesto u matrici data_out, čija se evidencija prati varijablom place. Varijabla place se iz ovoga razloga nakon svakog upisa povećava za vrijednost jedinice.

```
092     for(int i=3; i<6; i++){
093         for(int t=0; t<4; t++){
094           data_out[place]=SPI_array[i]>>(8*t) & 0xFF;
095           place++;
096         }
097     }
```

Nakon završene separacije varijabli, u matrici data_out nalazi se ukupno 15 popunjenih polja s po 8 bitova podataka koji su spremni za slanje putem SPI protokola na NUCLEO. Za početak prijenosa podataka potrebno je: postaviti željenu frekvenciju (brzinu) komunikacije, odrediti da li će se prvo slati LS ili MS bit, te koji tip komunikacije signalizacije će se koristiti. Sva tri parametra mogu biti drugačije podešeni, te nijedan od njih neće imati značajni utjecaj na krajnju funkciju ovoga seminarskog rada dok god su parametri jednako podešeni na oba mikrokontrolera.

```
099     vspi->beginTransaction(SPISettings(SPI_frekvencija, MSBFIRST,
SPI_MODE0));
```

Kako bi se zavisni komunikacijski uređaj (NUCLEO) mogao pravovremeno pripremiti za primitak podataka, potrebno je isto nagovijestiti spuštanjem „SS“ linije (eng. slave select) na pinu br. 5.

```
100     digitalWrite(5, LOW);
```

U praksi se pokazalo nužnim dati zavisnom komunikacijskom uređaju vremena kako bi završio operacije koje prethode provjeri početka komunikacije, jer prethodno nije napravljeno kao prekid (eng. Interrupt). Trajanje ove odgode je u milisekundama.

```
101     delay(15);
```

Nakon što se odgoda izvršila, započinje se prijenos podataka putem MOSI linije. Petlja za slanje podataka izvedena je na način da obavi ukupno 15 ponavljanja (jednako ukupnom broju polja popunjenih korisnim podacima u matrici data_out), tijekom kojih se varijabla i povećava za vrijednost jedan pri svakom ponavljanju funkcije (limiti [0, 15>). Prethodna odluka donesena je na temelju potrebe da varijabla imena i označava koji podatak iz matrice data_out će se slati SPI protokolom na NUCELO. Također je napravljena odgoda od 15 milisekundi kako bi bili sigurni da je NUCLEO stigao procesuirati podatke i obaviti ostale procese (eksperimentalno se pokazala potreba za istim).

```
104     for(int i=0; i<15; i++){
105         vspi->transfer(data_out[i]);
106         delay(15);
107     }
```

Po završetku prijenosa svih podataka radi se odgoda od 10 milisekundi. Nakon isteka odgode, stanje na liniji SS mijenja se u logičku jedinicu koja je mirno stanje za tip komunikacije nula (eng. mode), te se također na ESP32 obavi procedura završetka komunikacije.

```
109     delay(10);
110     digitalWrite(5, HIGH);
111     vspi->endTransaction();
112   }
```

U poglavlju sljedećem poglavlju (4.2) objašnjen je način na koji NUCLEO dobiva, obrađuje i pohranjuje podatke zaprimljene sa ESP32 mikrokontrolera.

4.2 Zavisni komunikacijski uređaj

NUCLEO ima funkciju zavisnog komunikacijskog uređaja (eng. slave), no za razliku od Arduino programskog jezika (ESP32), nije potrebno pozivati posebnu biblioteku, već je sve potrebno uključeno u osnovnu biblioteku imena mbed.

```
2       #include "mbed.h"
```

S obzirom da NUCLEO također ima više sučelja za komunikaciju putem SPI protokola, potrebno je definirati koji ulazi/izlazi se koriste za komunikaciju prema ESP32 mikrokontroleru. U slučaju ovog seminarskog rada koriste se izlazi PA_7 za MOSI, PA_6 za MISO, PA_5 za SCLK i PA_4 za SS liniju, a prototipna funkcija je nazvana spi_port. Za detaljnije informacije u svezi fizičke izvedbe ožičenja preporuča se konzultirati sliku 1.1-2.

```
18      SPISlave spi_port(PA_7, PA_6, PA_5, PA_4);
```

Kako bi se primljeni podaci mogli pohraniti i kasnije koristiti globalno u funkcijama koje ih dalje obrađuju, stvorena je jednodimenzionalna matrica imena SPI_array s ukupno 6 polja, koja podatke sprema u proširenom formatu (eng. long). Prošireni format odabran je iz potrebe spremanja 4 bajta za varijable Alarm_EPOCH i Time_EPOCH (primjer dekadskog podatka 1619094174 u binarnome obliku 01100000 10000001 01101010 10011110).

```
34      long SPI_array[6]={};
```

Prethodno prvoj upotrebi SPI komunikacije, potrebno je podesiti: najveću količinu podatka koja se očekuje prihvatiti po jednom prijenosu, tip signalizacije na sabirnici (mirno/aktivno stanje za START, STOP i ponovljeni START), te frekvenciju (brzinu) komunikacije koja će se koristiti sa ESP32 mikrokontrolerom. Kao što je već prije napomenuto, u prethodnom poglavlju najveća veličina podatka koja se odjednom može poslati je standardno uzeta 1 bajt (8 bitova), dok za tip signalizacije nije bitno koji se izabere, pod uvjetom da je na oba komunikacijska uređaja jednako podešen. Frekvencija (brzina) komunikacije uvjetovana je ograničenjem NUCLEO platforme koja ne podržava standardnu frekvenciju od 1MHZ (1Mbit/s), već je najbliža vrijednost od 1.125 MHz (1.125 Mbit/s).

```
271         spi_port.format(8,0);
272         spi_port.frequency(1125000);
```

S prethodnim je završena inicijalizacija SPI protokola. S obzirom da je NUCLEO u ulozi zavisnog komunikacijskog uređaja (eng. slave), potrebno je konstantno provjeravati stanje na SS liniji. Ovaj proces se odvija unutar while petlje koja se nalazi u main funkciji na način da je za uvjet if provjere postavljena funkcija spi_port.receive, koja vraća logičku jedinicu u slučaju detekcije početka prijenosa podataka. Potrebno je napomenuti kako se ova provjera ne izvodi uz pomoć prekida (eng. interrupt), s obzirom da isto nije preporučljivo iz razloga koji su preširoko područje za obraditi unutar ovog seminarskog rada.

```
286         while(1){
287             if(spi_port.receive()){
288                 SPI_receive();
289             }else{
290                 urica();
291
292                 if(second == 0 && minute == 0 && hour == 0){
293                     update_date();
294                 }
295             }//else
296         }//while(1)
```

Kada se detektira spuštanje SS linije, funkcija spi_port.receive vraća logičku jedinicu u if provjeru, te time omogućava pokretanje SPI_receive funkcije. Funkcija SPI_receive je tipa void, te ne prima ulazne parametre, već koristi prije spomenutu globalnu varijablu SPI_array u koju pohranjuje primljene, te potom obrađene podatke.

```
179    void SPI_receive(){
```

Slično metodi rabljenoj na ESP32 mikrokontroleru, koristi se jednodimenzionalna matrica imena data_in koja je karakternog tipa (eng. char). Ovaj format odabran je na temelju najveće količine podatka koju može pohraniti, koja iznosi 8 bitova. Varijabla place služi za redoslijedno upisivanje primljenih podataka u data_in matricu.

```
180          char data_in[16];
181          int place=0;
```

Jedan od načina na koji se mogu prihvaćati i spremati podaci koji se primaju putem SPI protokola je imati petlju koja će se ponoviti jednaki broj puta koliko podataka pristiže, te svaki od njih spremiti na zasebno mjesto u matrici uzimajući pomoćnu varijablu (u ovom konkretnom slučaju varijablu i), koja će svaki novi podatak pomicati za jedno mjesto dalje prilikom spremanja. Sa ESP32 dobiva se ukupno 15 podataka po 8 bitova (15 bajtova). S obzirom na pravilo da se trajanje komunikacije treba svesti na minimalno moguće, podaci koji se dobiju putem SPI komunikacije se neobrađeni direktno spremaju u matricu.

```
184          for(int i=0; i<15; i++){
185                data_in[i] = spi_port.read();
186          }
```

Po završetku komunikacije započinje se obradom primljenih informacija. Prva tri dobivena podatka su prognoze temperatura koje su veličine 8 bitova, te ih se samim time može direktno zapisati kao već obrađene podatke u matricu SPI_array. Varijabla place se ovdje ne inicijalizira lokalno unutar same for petlje, već će rabi unutar cjelokupne funkcije SPI_receive kako bi se pratio redoslijed obrade podataka.

```
189          for(place=0; place<3; place++){
190                SPI_array[place] = data_in[place];
191          }
```

Podaci koji se nalaze u poljima matrice [3, 15] su veličine 4 bajt-a (32 bita) i samim time zahtijevaju napredniji algoritam obrade. For petlja ograničena je na ukupno tri ponavljanja s obzirom da se obrađuju tri različita podatka. Prethodno slanju podataka na ESP32 mikrokontroleru, podaci su bili rastavljeni na 4 dijela od kojih je svaki bio po 8 bitova. Kako bi se podatak rekonstruirao, potrebno je voditi računa o dijelovima koji se uzimaju iz data_in matrice i za koliko ih se mjesta pomiče prije spajanja svih dijelova uz pomoć logičke operacije I. Obrađeni podatak sprema se u matricu SPI_array na mjesto koje određuje lokalna pomoćna varijabla i. Uz pomoć varijable place, kao što je i prije navedeno, vodi se računa o dijelovima koji se uzimaju za rekonstrukciju podatka, te se ista na kraju petlje povećava za vrijednost 4, što je ekvivalent broju dijelova po jednoj cijeloj varijabli (4 bajta).

```
194          for(int i=3; i<6; i++){
196                SPI_array[i] = (long)data_in[place] |
(long)data_in[place+1]<<8 | (long)data_in[place+2]<<16 |
(long)data_in[place+3]<<24;
197                place = (place + 4);
198          }
```

Kako bi se izbjeglo kašnjenje sata poželjno je odmah po dobivanju sastavljenog podatka trenutačnog vremena u EPOCH formatu upotrijebiti isti za sinkronizaciju NUCLEO-vog ugrađenog sata (RTC-a). S obzirom da se vrijeme ne prikazuje u formatu zapisa EPOCH, isti je potrebno pretvoriti u format

pogodan za krajnjeg korisnika (minute, sat, dan, mjesec, godina), zbog čega se pokreće funkcija EPOCH_and_ALARM.

```
200         set_time(SPI_array[5]);
201         EPOCH_and_ALARM();
```

U poglavlju 3.2 opisana je značajka koja u slučaju da se nije dogodio prijenos podataka sa ESP32, na LCD-u ispiše tekst „No information“. Ova značajka se isključuje u slučaju da je varijabla info postavljena u logičku jedinicu, što je u ovome trenutku potrebno napraviti s obzirom da je prijenos upravo završio i podaci su spremni za daljnju obradu i prikaz.

```
203         info = 1;
```

Kako bi se dobiveni podaci momentalno prikazali na ekranu, prvotno se aktivira funkcija čišćenja ekrana, a potom funkcije za popunjavanje LCD-a sadržajem (datum, sobna temperatura/vlaga, prognoze temperatura i prognoza vremenskih prilika).

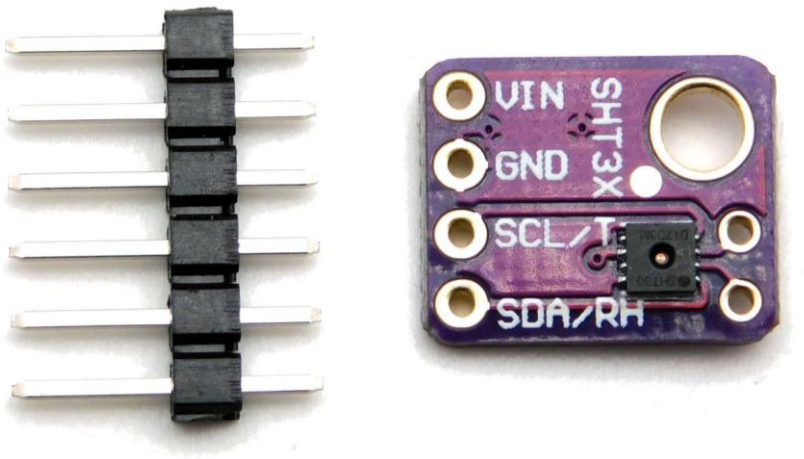
```
205         lcd.cls();
206         display_main();
207         update_date();
208
209
210   }
```

5. Senzor vlage i temperature zraka (SHT30)

Za senzor vlage i temperature izabran je SHT30 (eng. Sensor Humidity Temperature) od proizvođača SENSIRION koji se nalazi na razvojnoj pločici s ugrađenim otpornicima za I2C komunikaciju i izvodima pogodnim za korištenje na prototipnoj pločici (eng. breadboard). Senzor je primarno odabran zbog komunikacijskog protokola I2C kako bi se mogao koristiti u zajedničkoj sabirnici s LCD-om, koji također radi putem prije spomenutog protokola (LCD primarno koristi paralelnu komunikaciju, no za ovaj seminarski rad izabrana je verzija sa sučeljem koje obavlja konverziju paralelne komunikacije u serijsku I2C). Prema proizvođačevoj tablici podataka (eng. datasheet) (naziv podatkovnog lista: Sensirion_ Humidity_Sensors_SHT3x_Datasheet_digital-971521) osnovne karakteristike senzora su:

- U potpunosti kalibriran
- Podržane razine napajanja [2.4, 5.5]V
- I2C komunikacija do najviše 1 MHz i dvije korisnički birane adrese
- Prosječna točnost ±1,5% za vlagu, ±0,2°C za temperaturu zraka
- Relativno brzo vrijeme pokretanja
- Kompaktno pakiranje s 8 izvoda

Prije početka komentiranja kôda vezanog uz senzor, potrebno je proučiti njegovu dokumentaciju kako bi se njime moglo pravilno rukovati. Razvojna pločica na kojoj se nalazi senzor ima ukupno 6 izvoda (vidljivo na slici broj 5-1). Za funkcionalnost svakoga izlaza razvojne pločice preporuča se konzultirati tablicu broj 5-1.



Slika 5-1 Izgled razvojne pločice sa senzorom SHT30 bez zalemljenih priključaka, izvor: https://nettigo.eu/products/sensirion-sht30-humidity-and-temperature-sensor-with-i2c-interface

Izlazi razvojne pločice	Opis/namjena
VIN	Napajanje sklopa, 3.3[V]
GND	Nula elektronike
SCL/T	Linija za prijenos serijskog takta (eng. Serial Clock) I2C komunikacije
SDA/RH	Linija za prijenos serijskih podataka (eng. Serial Data) I2C komunikacije
AL	Linija za postavljeni alarm u slučaju prelaska zadane vrijednosti
AD	Izlaz za odabir I2C adrese, adresa 0x44 za GND, 0x45 za VIN

Tablica 5-1 Tumačenje izlaza razvojne pločice SHT30

S obzirom na prethodnu tablicu (5-1), napravljena je elektronička shema na slici 1.1-2. Iz sheme je vidljivo da je pin imena AD razvojne pločice sa senzorom SHT30 spojen direktno na nulu elektronike,

te je time za I2C adresu odabrana heksadekadska vrijednost 44. Prethodno funkciji čija je namjena komunikacija sa senzorom, potrebno je napraviti prototip I2C komunikacije na odgovarajućim izlazima. Za SHT30, kao i za LCD, koristi se isto I2C1 sučelje, odnosno digitalni izlazi D14 i D15. Prototipnoj funkciji je dano ime SHT3X.

```
19    I2C SHT3X(D14, D15); //SDA, SCL
```

Kako bi se dobivene vrijednosti mogle rabiti van funkcije za dohvaćanje podataka sa senzora SHT30, stvorene su globalne cjelobrojne varijable imena temp i hum.

```
37    int temp, hum;
```

Također je potrebno inicijalizirati parametre I2C komunikacije. S obzirom da je poželjno što je brže moguće izvođenje funkcije i komunikacije (zbog osvježavanja numeričkog ekrana), uzeta je maksimalno dopuštena frekvencija (brzina) komunikacije od 4kHz, obzirom na ograničenja NUCLEO-a i SHT30.

```
275         SHT3X.frequency(400000);
```

Funkcija za dohvaćanje podataka sa senzora (tipa void i ne prima ulazne parametre) pokreće se prilikom izvođenja funkcije imena display_main, te je taj proces detaljnije pojašnjen u poglavlju broj 3.

```
148   void SHT30(void){
```

Na samome početku funkcije inicijaliziraju se varijable potrebne za rad. Varijabla adr predstavlja adresu senzora za I2C komunikaciju (0x44 ako je pin AD spojen na GND).

```
149         int adr=0x44;
```

Datasheet SHT3x-DIS

Condition		Hex. code	
Repeatability	Clock stretching	MSB	LSB
High	enabled	0x2C	06
Medium			0D
Low			10
High	disabled	0x24	00
Medium			0B
Low			16

Tablica 5-2 Heksadekadske oznake različitih načina rada senzora, izvor: Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital-971521

Jednodimenzionalna matrica imena podaci sadrži podatke koji predstavljaju postavke za sami senzor. Prema tablici na slici iznad, nulti podatak u matrici će na senzoru omogućiti tzv. rastezanje sata (eng. clock stretching), kako bi NUCLEO imao uvid u stanje senzora tj. kada je isti završio mjerenje. Sljedeći podatak u matrici odnosi se na ponovljivost, odnosno točnost mjerenja, koje izvodi senzor. S obzirom da su za krajnju funkcionalnost ovog seminarskog rada potrebne samo okvirne vrijednosti, ponovljivost je stavljena na najnižu vrijednost (eng. low).

```
150         char podaci[2]= {0x2C, 0x10};
```

Povratni podaci sa senzora vraćaju se u obliku prema slici 5-3, stoga je jedan od načina za njihovo spremanje i obradu prvotno spremanje u privremenu matricu.

```
151         char data[6]= {};
```

Cjelobrojne varijable imena chk i pass objašnjene su uz ostatak kôda u nastavku.

```
152         int chk, pass=0;
```

`int write (int address, const char *data, int length, bool repeated=false)`	**Master-transmitter** mod: Slanje podatka određenom slaveu na adresi `address`; `*data` označava pokazivač na niz podataka; `length` označava broj bajtova podataka koji će se poslati; `repeated` je logička varijabla koja označava ponovljeni start nakon kraja prijenosa, ako je u logičkoj jedinici, tj. `true`; Vraća `0` ako je poslano `length` bajtova, inače broj različit od `0`
`int read (int address, char *data, int length, bool repeated=false)`	**Master-receiver** mod: Čitanje podatka koji šalje slave; `address` je adresa slavea; `*data` pokazivač na niz podataka; `length` je broj bajtova podataka; `repeated` je logička varijabla koja označava ponovljeni start nakon kraja prijenosa, ako je u logičkoj jedinici, tj. `true`; Funkcija vraća `0` ako je uspješno primljeno `length` bajtova, inače broj različit od `0`

Slika 5-2 Opis prototipnih funkcija I2C komunikacije, izvor: 06_Serijska_komunikacija_I2C autora Dr. sc. Tomislav Pavlović

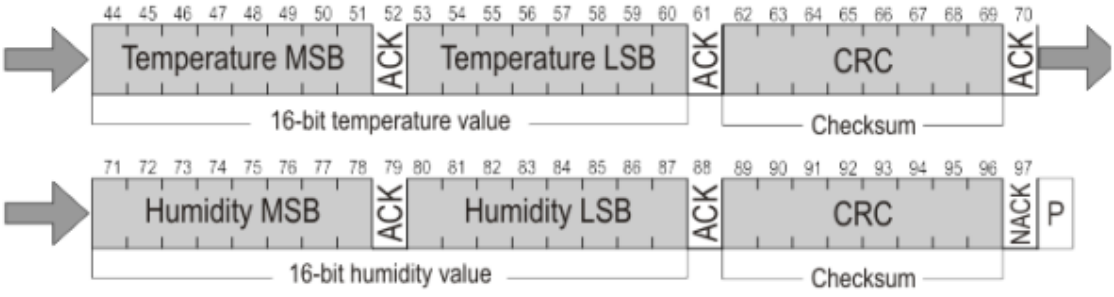
Kako bi poslali postavke mjerenja prema senzoru, potrebno je koristiti I2C funkciju za slanje podataka, čija se sintaksa može pogledati na slici iznad. Primjetno je da funkcija zahtjeva adresu koju je potrebno binarno pomaknuti za jedno mjesto u lijevo, kako bi se LS bitu oslobodilo mjesto za oznaku pisanja (eng. write bit). Zatim je potrebno u matrici karakternog tipa (eng. char) navesti podatke koji se žele poslati, te iza imena matrice ukupni broj bajtova koji se šalju. Za slučaj ovog seminarskog rada bilo je potrebno na kraju komunikacije imati ponovljeni start, te je iz toga razloga stavljena logička jedinica na kraj u obliku riječi istina (eng. true). S obzirom da je eksperimentalno utvrđena povremena prisutnost grešaka u komunikaciji i time uzrokovana nedostupnost senzora za komunikaciju, napravljena je do while petlja koja će u vremenskim intervalima od po 3 sekunde napraviti maksimalno pet pokušaja dobivanja logičke nule u chk varijabli, prije nego što prekine petlju i nastavi dalje izvoditi funkciju s podacima od prijašnje komunikacije ili NULL podacima. U slučaju kada je dobivena logička nula u chk varijabli, radi se odgoda daljnje komunikacije kako bi senzor stigao obaviti mjerenje.

```
154         do{
155               chk = SHT3X.write(adr<<1, podaci, 2, true);
157               if(pass != 0){
158                     wait(3);
159                     pass++;
160               }else{
161                     pass++;
162               }
163
164         }while(chk != 0 && pass != 5);
165
166         wait_ms(10);
```

Nakon 10 milisekundi rabi se I2C naredba za primitak podataka, čija je sintaksa vidljiva na slici 5-3. Prema sintaksi naredbe za dobivanje informacija sa zavisnog komunikacijskog uređaja (eng. slave), unutar zagrada potrebno je staviti: adresu uređaja od kojeg želimo dobiti odgovor binarno pomaknutu za jedno mjesto ulijevo, jednodimenzionalnu matricu u karakternom formatu koja će pohraniti dolazne podatke, količinu dolaznih podataka iskazanu brojem bajtova i logičku oznaku ponovljenog početka (ne rabi se jer je sljedeće mjerenje za 20 sekundi).

```
167         chk = SHT3X.read(adr<<1, data, 6, false);
```

Tijekom izvršavanja linije br. 167, primljeni su podaci u redoslijedu koji predstavlja slika 5-3, te su istim tim redoslijedom spremljeni u matricu data.



Slika 5-3 Struktura povratnih podataka sa SHT30

Prema službenim formulama iz prije navedene dokumentacije senzora (izvadak na slici ispod), napravljene su jednadžbe za preračunavanje dobivenih podataka u podatke temperature i relativne vlažnosti. Kod obje formule ubačen je algoritam za spajanje podataka, obzirom da se prethodno ovim jednadžbama podaci nalaze razdvojeni u matrici data.

Relative humidity conversion formula (result in %RH):

$$RH = 100 \cdot \frac{S_{RH}}{2^{16} - 1}$$

Temperature conversion formula (result in °C & °F):

$$T[°C] = -45 + 175 \cdot \frac{S_T}{2^{16} - 1}$$

Slika 5-4 Matematički izrazi za dobivanje relativne vlažnosti i temperature od podataka sa SHT30

```
170         temp=(-45+175*(data[0]<<8 | data[1])/65535);
171         hum=(100*(data[3]<<8 | data[4])/65535);

176     }
```

Završetkom funkcije, u varijabli temp se nalazi vrijednosti temperature (u °C), dok se istovremeno u varijabli hum nalazi vrijednost relativne vlažnosti zraka (u %).

6. Dohvaćanje trenutačnog vremena putem NTP-a

NTP (eng. Network Time Protocol) je mrežni protokol namijenjen računalnim sistemima za sinkronizaciju satova, koji koristeći razne algoritme omogućava relativno dobru točnost (prosječno 10 milisekundi greške). Što se tiče ovog seminarskog rada, NTP protokol se koristi za komunikaciju sa servisom ntp.org, odnosno s europskim poslužiteljem na adresi 2.europe.pool.ntp.org. Potrebno je naglasiti kako se kôd opisan u ovome poglavlju odrađuje na ESP32 mikrokontroleru. Prije upotrebe ovog protokola nužno je pozvati biblioteke: standardnu biblioteku za spajanje na bežičnu mrežu (WiFi.h i ESP32Ping.h), biblioteka za HTTP protokol (HTTPClient.h) i biblioteka za manipulaciju podacima vezanim uz vrijeme (time.h).

```
001   #include <WiFi.h>
002   #include <ESP32Ping.h>
003   #include <HTTPClient.h>
005   #include <time.h>
```

Nakon označavanja biblioteka, potrebno je inicijalizirati globalne varijable koje sadrže komunikacijske parametre. Prvi parametar imena ntpServer u sebi sadrži adresu poslužitelja s kojeg će se dohvatiti podaci. Sljedeća dva parametra određuju vremensku zonu u kojoj se krajnji korisnik nalazi (gmzOffset_sec) i razliku između zimsko/ljetnog računanja vremena (daylightOffset_sec).

```
016   const char* ntpServer = "2.europe.pool.ntp.org";
017   const long gmtOffset_sec = 3600;
018   const int daylightOffset_sec = 3600;
```

Kao i u drugim funkcijama, podatak koji će ova funkcija generirati se pohranjuje u jednodimenzionalnu matricu imena SPI_array za slanje na NUCLEO putem SPI protokola.

```
023   int SPI_array[7];
```

Funkcija za dohvaćanje vremena u loop petlji kao i u update_ALL funkciji pokreće se nakon funkcije za dohvaćanje postavljenog alarma, kao zadnja funkcija prethodno SPI komunikaciji s NUCLEO-m. Položaj funkcije je pomno odabran kako bi se izbjeglo veće kašnjenje numeričkog sata na NUCLEO-u.

Kao i većina drugih funkcija, GETLocalTime je tipa void i ne prima ulazne parametre, već se koristi prije spomenutim globalnim varijablama. Za potrebe rada funkcije potrebno je inicijalizirati dvije lokalne varijable. Prva varijabla tipa time_t (eng. time type) namijenjena je pohranjivanju vremena u formatu EPOCH, dok su druge dvije tipa struct, što označava listu varijabli kojima se manipulira kroz pokazivače.

```
116   void GETLocalTime(){
117     time_t EPOCH;
118     struct tm timeinfo;
```

Kako bi se dohvatili podaci o trenutnom vremenu dovoljno je pozvati funkciju imena getLocalTime s pokazivačem na memorijsku lokaciju varijable tipa struct, imena timeinfo. Funkcija getLocalTime u slučaju kada uspješno dohvati podatke vraća logičku jedinicu i time ne aktivira if funkciju, koja bi u protivnome javila grešku.

```
120     if (!getLocalTime(&timeinfo)) {
121         Serial.println("Greska u dohvacanju vremena");
122     }
```

S obzirom da ESP32 također ima ugrađeni RTC (eng. Real time clock), poželjno je isti sinkronizirati s dobivenim podatkom.

```
123       time(&EPOCH);
```

Kako bi se dobilo vrijeme prigodno za vremensku zonu GMT+1 i zimsko/ljetno računanje vremena, potrebno je na dohvaćeno trenutno vrijeme zbrojiti prethodno navedene vremenske pomake.

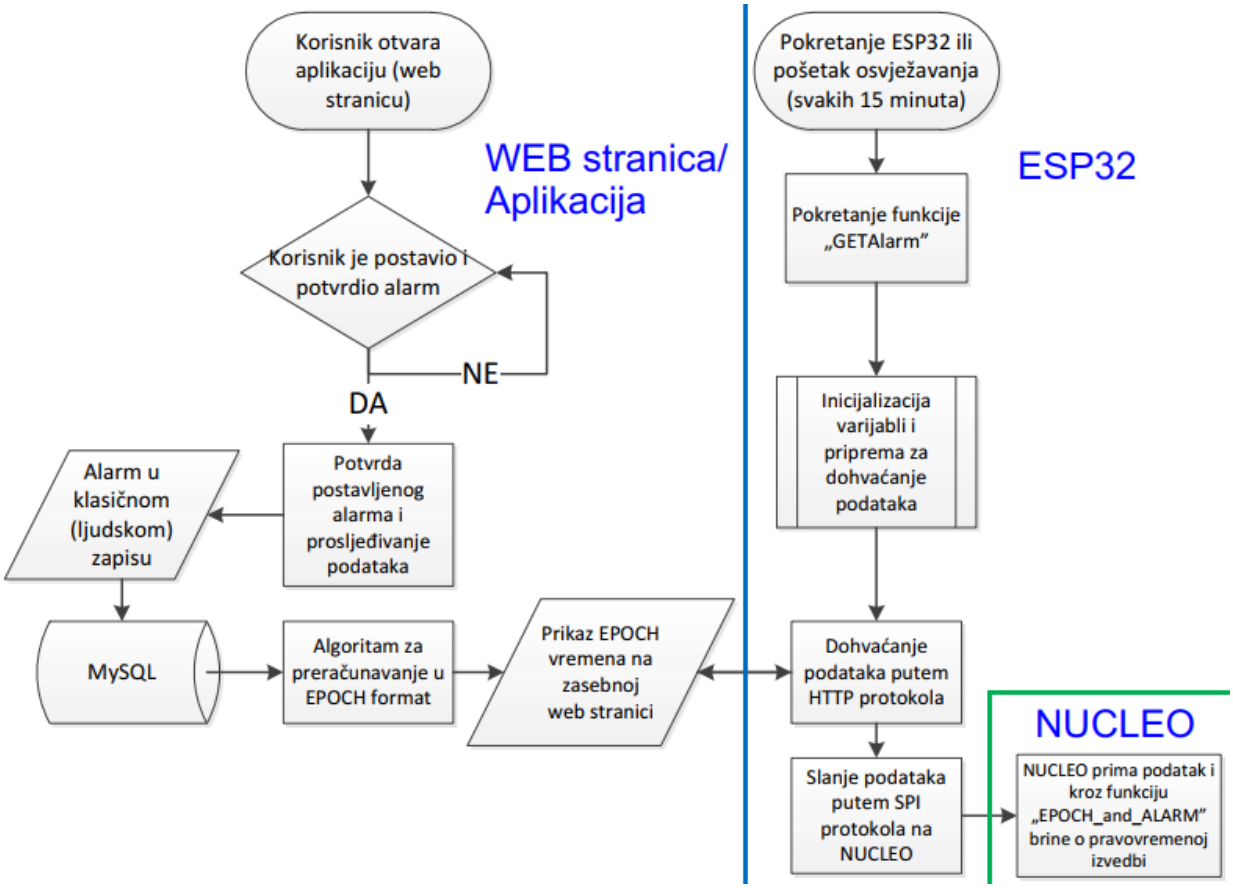
```
124       EPOCH = EPOCH + gmtOffset_sec + daylightOffset_sec;
```

Izvršavanjem kôda na liniji br. 124, funkcija je gotova sa svojim radom, te trenutačno vrijeme u EPOCH formatu sprema u matricu SPI_array.

```
126       SPI_array[5] = EPOCH;
127     }
```

7. Korisnička budilica

S obzirom da su sve funkcije ovoga projekta naprednije od klasičnog sata/budilice, sa sličnom mišlju napravljena je budilica koja se ne podešava fizički na samome satu/budilici, već putem mobilnog uređaja (obzirom da bi izrada mobilne aplikacije bila preširoki i nepotrebni dio posla, postavljanje alarma se odvija putem web stranice). Na slici ispod vidljiv je dijagram toka ovoga sustava, koji će biti detaljnije opisan po poglavljima.



Slika 7-1 Dijagram toka korisničke budilice

7.1 Internetska (web) stranica – korisnički upis podataka

Internetska (web) stranica zamišljena je kao prototip, odnosno primjer koncepta (eng. proof of concept) za postavljanje budilice. Web stranica je iz toga razloga napravljena minimalistički i uključuje naslov i podnaslov, dva polja za upisati/označiti vrijeme i datum alarma, te gumb za potvrdu slanja u bazu. Nakon desnog klika miša na „Submit“ gumb, korisnika se prosljeđuje na stranicu koja u slučaju uspješnog postavljanja alarma obavijesti korisnika tekstom „Novi alarm uspješno dodan“. U nastavku je dan detaljniji opis HTML kôda upotrjebljenog za prikaz stranice, kao i slika izgleda stranice u internetskom pregledniku.

Na liniji br. 1 stavljena je oznaka inačice HTML jezika, koja internetskom pregledniku određuje na koji način će kôd u nastavku interpretirati. Potom slijedi otvaranje html taga kao oznaka početka kôda i head taga koji je oznaka početka zaglavlja stranice.

```
01    <!DOCTYPE html>
02    <html>
03    <head>
```

Tag title služi za prikaz teksta koji se nalazi između otvorenog i zatvorenog title taga na mjestu kartice web stranice.

```
04          <title>Dodavanje alarma</title>
```

Kako bi web stranica imala tekst, polja i gumb poravnate po horizontalnoj sredini, napravljen je globalni stil za tijelo stranice (eng. body) koji u sebi sadržava funkciju za poravnavanje teksta (eng. text-align), s parametrom sredine (eng. center).

```
05          <style>
06                body {
07                      text-align: center;
08                }
09          </style>
```

Nakon završetka uređivanja zaglavlja, zatvara se tag istog i otvara tag tijela stranice (eng. body).

```
10    </head>
11
12    <body>
```

Tagovi „h1“ i „h2“ namijenjeni su za jednostavne naslove, te su numerirani po veličini fonta (h1>h2).

```
14    <h1>Dodavanje novoga alarma</h1>
15    <h2>Molimo upišite vrijeme novoga alarma:</h2>
```

Kako bi internetski preglednik znao da u nastavku slijedi forma, odnosno dijaloški okviri za upis podataka, potrebno je otvoriti tag form i definirati metodu prijenosa podataka (eng. method) i dokument koji će iste preuzeti (eng. action).

```
17    <form action="fni02u8FFFF2983B.php" method="post">
```

Koriste se dva polja za upis podataka, te ih je potrebno označiti tekstom ispred polja za upis. Za osnovno ispisivanje teksta (zadani font i veličina slova) u HTML jeziku nije definirana posebna sintaksa, već se upisom same riječi unutar kôda, ista ispisuje. Kako bi definirali polja za upis, koristi se tag input s parametrima za tip podataka (eng. type) i imenom podataka (eng. name). Tag br koristi se za prelazak u novi red na stranici.

```
19      Vrijeme: <input type="time" name="vrijeme" /><br><br>
20      Datum: <input type="date" name="datum" /><br><br>
```

Kako bi se završio input tag i samim time forma, potrebno je imati gumb za potvrdu slanja podataka (eng. submit). Pritiskom na istoimeni gumb, podaci se šalju POST metodom u skriptu imena „fni02u8FFFF2983B.php“, koja će iste upisati u MySQL bazu podataka. Obrada prije spomenute skripte nalazi se u sljedećem poglavlju.

```
22      <input type="submit" />
```

Na posljetku je potrebno zatvoriti prethodno otvorene tagove (forma, tijelo i završetak HTML koda).

```
24      </form>
25
26      </body>
27      </html>
```

Slika 7.1-1 Snimak ekrana internetske stranice za postavljanje budilice

7.2 Internetska (web) stranica – upis alarma u bazu podataka (MySQL)

Podaci koje je korisnik predao klikom na tipku „Submit“ predani su skripti u nastavku kao globalne varijable s imenima vrijeme i datum. Prethodno svemu potrebno je otvoriti tag html, a potom body (obzirom da nemamo potrebe za oblikovanjem ili prikazom podataka u zaglavlju).

```
01    <html>
02    <body>
```

Kako bi se upisani podatci mogli učitati u bazu podataka, jedna od mogućnosti je koristiti programski jezik PHP (kratica od eng. Hypertext Preprocessor), koji za svoj početak koristi sintaksu vidljivu na liniji broj 03.

```
03    <?php
```

Za pristupanje bazi podataka potrebno je biti u lokalnoj domeni spram iste iz sigurnosnih razloga. S obzirom na prethodno, a i na činjenicu da web stranica uistinu je u lokalnoj domeni spram baze podataka, definira se varijabla servername koja sadržava adresu baze (ime localhost ima ekvivalent adrese 127.0.0.1, koja je univerzalna adresa lokalne mreže/domene).

```
04    $servername = "localhost";
```

Pošto na poslužitelju postoji više baza podataka, potrebno je definirati ime baze i podatke za prijavu u istu (stvarni podaci za prijavu u bazu podataka nikako ne pridonose kvaliteti ovoga rada, te su iz toga razloga zamijenjeni zvjezdicama).

```
05    $dbname = "********";
06    $username = "********";
07    $password = "********";
```

S inicijaliziranim varijablama koje u sebi sadržavaju sve potrebne podatke, moguće je uspostaviti novu komunikaciju pod imenom conn.

```
09    $conn = new mysqli($servername, $username, $password, $dbname);
```

Kako bi se provjerila uspješnost povezivanja na bazu, implementirana je if funkcija koja u svojoj provjeri ima funkciju namijenjenu ispitivanju povezanosti. U slučaju kada veza nije uspostavljena, funkcija vraća logičku jedinicu i time aktivira dio kôda unutar if funkcije, koja će zaustaviti proces povezivanja i ispisati grešku koja se dogodila.

```
11    if ($conn->connect_error) {
12      die("Connection failed: " . $conn->connect_error);
13    }
```

U slučaju da se prethodno nije dogodilo, odnosno ako se skripta uspješno povezala na bazu podataka, radi se nova varijabla sql koja će sadržavati kôd za manipulaciju podacima unutar baze. Kôd započinje naredbom INSERT, koja je funkcija za upis podataka u bazu. Nakon prije spomenute funkcije potrebno je naredbom INTO označiti u koju tablicu i polja unutar baze želimo upisati podatke. Naposljetku, naredbom VALUES se označe varijable koje se spremaju u prije definirana polja unutar baze. Prilikom definiranja varijabli koje se žele pohraniti u bazu, potrebno je obratiti pažnju na to da je njihov redoslijed identičan kao i redoslijed definiranih polja u koja se upisuju. Također, u slučaju da su varijable predane skripti iz druge skripte, potrebno je navesti njihov izvor, odnosno protokol, kojim su se iste prenijele.

```
15    $sql="INSERT INTO Seminarski_alarm (vrijeme, datum)
16
```

```
17    VALUES ('$_POST[vrijeme]','$_POST[datum]')";
```

Po završetku stvaranja varijable imena sql, ista se šalje u terminal baze (eng. query) s uvrštenim varijablama. U slučaju da je naredba sintaktički ispravna, odnosno ako je MySQL vratio pozitivnu povratnu informaciju, podaci su upisani u bazu i funkcija vraća logičku jedinicu koja omogućava ispis teksta „Novi alarm uspješno dodan“.

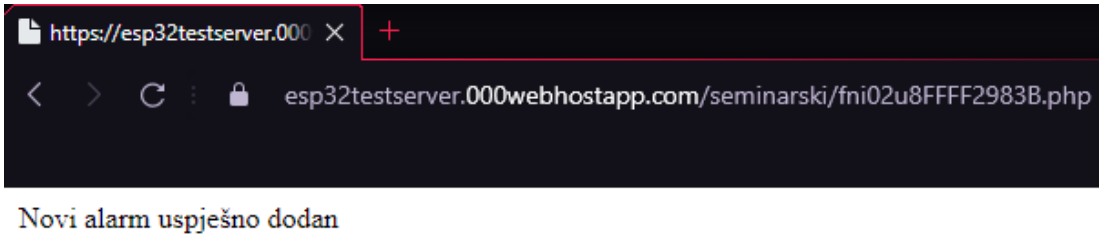
```
19    if ($conn->query($sql)) {
20      echo "Novi alarm uspješno dodan";
```

U protivnome se ispisuje tekst „Greška: “ i detaljni opis problema koji se dogodio.

```
21    } else {
22      echo "Greška: " . $sql . "<br>" . $conn->error;
23    }
```

Po završetku potrebno je zatvoriti vezu prema bazi (ušteda resursa) i zatvoriti: PHP dio koda, tijelo HTML koda i sami HTML.

```
25    $conn->close();
26    ?>
27    </body>
28    </html>
```



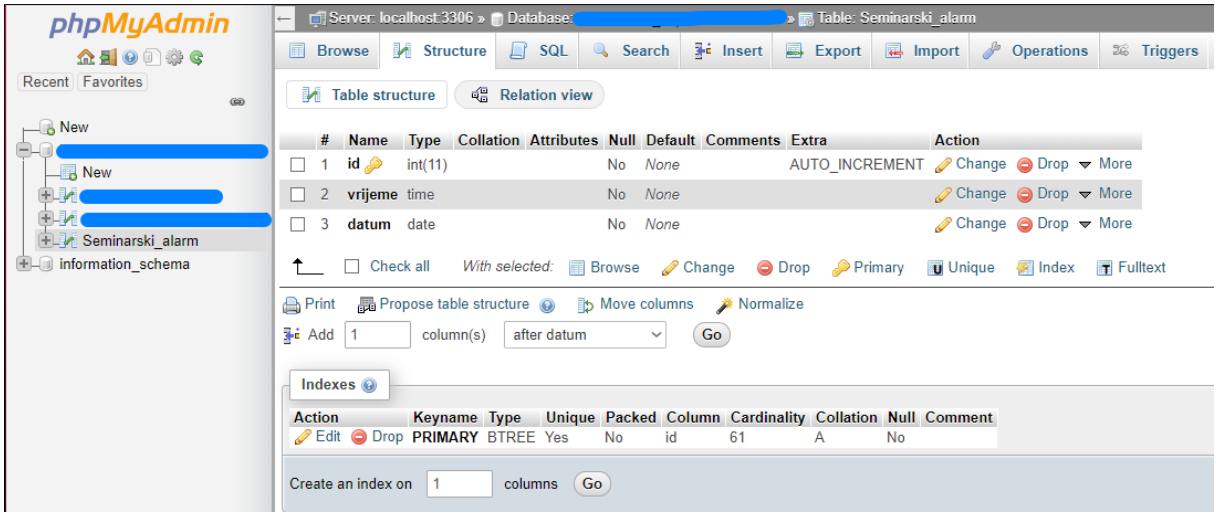
Slika 7.2-1 Snimak ekrana uspješno pohranjenog vremena postavljenog budilice

7.3 MySQL baza podataka

MySQL baza podataka je jedan od najčešće upotrebljavanih programskih rješenja za skladištenje podataka. Razlozi prethodnom su: besplatan sustav, otvoreni kôd, relativno dobre performanse i stabilnost, te dobro dokumentirane ekstenzije i moduli. Također je potrebno napomenuti da je u mnogim programskim jezicima (npr. PHP, Python, Java) napravljena podrška za komunikaciju s MySQL bazom podataka. Razlog više zašto je MySQL odabran za ovaj seminarski rad je standardna implementiranost u gotovo sve internetske poslužitelje, pa tako i na 000webhost.com (u daljnjem tekstu host), za kojeg sam se odlučio zbog prijašnje suradnje i iskustva.

Za upotrebu servisa je dovoljno izraditi besplatni račun i rezervirati domenu (za potrebe ovog seminarskog rada stvorena je: esp32testserver.000webhostapp.com). Nakon procesa izrade korisničkog računa, kroz cPanel (sustav često korišten od strane raznih internetskih poslužitelja za upravljanje internetskim stranicama) stvorena je nova baza s pripadajućim imenom i podacima za prijavu.

Kroz sustav phpMyAdmin (jedan od sustava za upravljanje MySQL bazom podataka, napisan u PHP programskom jeziku) je stvorena tablica imena Seminarski_alarm, u kojoj su zatim stvorena polja: id (tipa cjelobrojne varijable), vrijeme (tipa varijable namijenjen spremanju vremena) i datum (tipa varijable namijenjen spremanju datuma). Ovime je stvorena u potpunosti funkcionalna baza podataka koja se koristi za pohranjivanje i čitanje podataka.



Slika 7.3-1 Snimak ekrana strukture tablice kroz sučelje phpMyAdmin

7.4 Internetska (web) stranica – ispis podataka za ESP32

Baze podataka se vrlo rijetko otvaraju za direktan pristup s interneta, te većina internetskih poslužitelja zabranjuje tu opciju iz sigurnosnih razloga. Kako bi se podaci bez obzira na prethodno svejedno mogli dohvatiti, potrebno je napraviti stranicu koja će na zahtjev ispisivati podatke prema predodređenim pravilima ili prihvaćati unos podatka kao niz koji bi se inače stavio direktno u MySQL terminal. S obzirom da je za ovaj projekt bilo dovoljno dohvatiti samo zadnji postavljeni alarm, odabrana je prva opcija, te je u nastavku objašnjen PHP kôd koji obavlja tu funkcionalnost. Također, za buduću analizu ovoga seminarskog rada potrebno je napomenuti kako se datoteka koja obavlja prethodno navedenu funkcionalnost naziva „index.php“.

Kao što je i prije rečeno, PHP kôd je potrebno započeti sa „<?php“ i završiti sa „?>“.

```
01    <?php
```

S obzirom da ova skripta nema bitne relacije sa skriptom u kojoj su već bili definirani podaci za prijavu na bazu, potrebno je iste ponovo inicijalizirati (za detalje konzultirati poglavlje 7.2). Isto je primjenjivo i na linije od broja 8 do 12.

```
03    $servername = "localhost";
04    $dbname = "**************";
05    $username = "************";
06    $password = "************";
07
08    $conn = new mysqli($servername, $username, $password, $dbname);
09
10    if ($conn->connect_error) {
11      die("Connection failed: " . $conn->connect_error);
12    }
```

Za označavanje redaka u bazi podataka koristi se naredba SELECT, nakon koje je potrebno navesti sve redove koji se žele dohvatiti, odvojene zarezom. Potom je potrebno označiti u kojoj tablici će se prethodni podaci tražiti. Kako izvršena naredba ne bi vratila sve zapise iz baze, potrebno je ograničiti rezultat na samo onaj čiji je id broj jednak najvećem id broju (funkcija max s parametrom id), odnosno onaj koji je zadnji upisan u bazu. Naredba FROM je morala biti ponovljena iz razloga što se maloprije opisana radnja također može obavljati s parametrima iz drugih tablica.

```
14    $sql = "SELECT vrijeme, datum FROM Seminarski_alarm WHERE id=(SELECT
max(id) FROM Seminarski_alarm)";
```

Linija br. 16. također je objašnjena u poglavlju 7.2. Ukratko, njena funkcija je izvršiti naredbe iz retka 14 i provjeriti da li su se iste uspješno obavile.

```
16    if ($result = $conn->query($sql)) {
```

U slučaju kada jesu, pokreće se while petlja koja će se izvršavati sve dok će postojati podaci koji se mogu dohvatiti, odnosno dok funkcija fetch_assoc vrati logičku nulu.

```
17          while ($row = $result->fetch_assoc()) {
```

Kako bi se lakše manipuliralo podacima dobivenim iz baze, stvorene su varijable row_vrijeme i row_datum koje pohranjuju istoimene podatke.

```
18                $row_vrijeme = $row["vrijeme"];
19                $row_datum = $row["datum"];
```

Za pretvaranje datuma alarma iz ljudski čitljivog formata (eng. human-readable) u EPOCH format, upotrebljena je funkcija strtotime, s datumom kao ulaznim podatkom.

```
21                  $EPOCH_datum = strtotime($row_datum);
```

Kako je vrijeme (sati, minute i sekunde) daleko lakše za pretvoriti u EPOCH format, dovoljno je iste prvotno razdvojiti na zasebne članove. Ovu funkciju obavlja naredba explode, te je kao prvi parametar razdvajanja stavljena dvotočka, a kao drugi varijabla u kojoj je sadržano vrijeme alarma u formatu hh:mm:ss (sati:minute:sekunde). Nakon izvršenja kôda na liniji broj 22, stvorena je jednodimenzionalna matrica u kojoj su zasebno spremljena sva tri podatka.

```
22                  $EPOCH_vrijeme = explode(":", $row_vrijeme);
```

Kako bi se izračunao ukupni broj sekundi od 01.01.1970. do postavljenog alarma (EPOCH format), potrebno je na već preračunati datum dodati broj sati (EPOCH_vrijeme[0]) uvećan 3600 puta (broj sekundi u jednome satu), potom dodati broj minuta (EPOCH_vrijeme[1]) uvećan 60 puta (broj sekundi u jednoj minuti), te zbrojiti s brojem sekundi (EPOCH_vrijeme[3]).

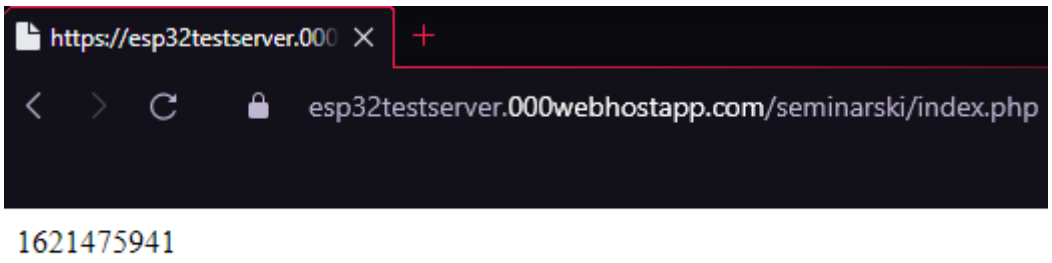
```
25                  $EPOCH_total = $EPOCH_datum + $EPOCH_vrijeme[0]*3600 +
$EPOCH_vrijeme[1]*60 + $EPOCH_vrijeme[0];
```

Po završetku linije broj 25, rezultat novo nastale varijable EPOCH_total ispisuje se korisniku kao jedini vidljivi rezultat ove skripte. Ispis koji je obavljen linijom 26 nije namijenjen za krajnjeg korisnika, već za ESP32 mikrokontroler koji će isti dohvatiti. Ovaj proces objašnjen je u sljedećem poglavlju.

```
26                  echo $EPOCH_total;
27              }
```

Naposljetku je poželjno očistiti sve dohvaćene podatke za sljedeću sesiju, terminirati vezu prema bazi za uštedu resursa i označiti kraj PHP koda („?>“).

```
28              $result->free();
29      }
30      $conn->close();
31      ?>
```



Slika 7.4-1 Snimak ekrana internetske stranice za ispis EPOCH vremena budilice

7.5 ESP32 – Dohvaćanje alarma s internetske stranice

U prethodnom poglavlju opisan je proces koji kao svoj rezultat ima ispis podatka EPOCH vremena namještene budilice. Kako bi se isti podatak prenio na NUCLEO, koji izvodi funkciju budilice, prethodno ga je potrebno dohvatiti s poslužitelja i obraditi, a potom poslati putem SPI protokola. Funkcija unutar ESP32 mikrokontrolera koja obavlja dohvaćanje i obradu podatka objašnjena je u nastavku.

Obzirom da i funkcija za dohvaćanje budilice zahtjeva internetsku vezu, potrebno je za istu inicijalizirati biblioteke WiFi i ESP32Ping.

```
001   #include <WiFi.h>
002   #include <ESP32Ping.h>
```

Funkcija za dohvaćanje podatka s prije spomenute internetske stranice također koristi prethodno inicijaliziran HTTP protokol.

```
003   #include <HTTPClient.h>
```

Kao i u drugim funkcijama, podatak koji će ova funkcija generirati sprema se u jednodimenzionalnu matricu imena SPI_array za slanje na NUCLEO putem SPI protokola.

```
023   int SPI_array[7];
```

Funkcija za dohvaćanje postavljene budilice pokreće se kao i ostale funkcije iz while petlje ili iz funkcije update_ALL.

```
281   void loop(){
282     GETWeatherinfo();
283     GETAlarm(); //dohvacanje postavljene budilice
284     GETLocalTime();
285     SPI_send();
286     delay(3600000);
287   }
```

Prije same funkcije inicijalizirana je globalna varijabla imena prolaz (kako bi se zadržala njena vrijednost bez obzira na stanje funkcije), koja u slučaju greške vodi računa o broju pokretanja funkcije (dodatno objašnjenje prilikom upotrjebe iste u nastavku).

```
131   int prolaz=0;
```

Funkcija za dohvaćanje postavljene budilice imena GETAlarm je tipa void i za svoje funkcioniranje ne prihvaća ulazne parametre (vrijednosti).

```
132   void GETAlarm(void){
```

S obzirom da funkcija mora imati osiguranu povezanost na internet, poželjno je provjeriti da li je veza dostupna.

```
133     if ((WiFi.status() == WL_CONNECTED)) {
```

U slučaju kada je veza dostupna, stvara se prototipna funkcija za HTTP klijenta imena http, koji će kasnije biti korišten za komunikaciju istoimenim protokolom.

```
134         HTTPClient http;
```

Nakon inicijalizacije klijenta započinje se komunikacija s internetskim poslužiteljem (eng. begin), na kojem se izvršava skripta iz poglavlja 7.4, koja dohvaća podatke iz baze, te nakon obrade istih ispisuje EPOCH vrijeme namještene budilice.

```
136         http.begin("https://esp32testserver.000webhostapp.com/
seminarski/index.php");
```

Naime, podatak je potrebno preuzeti na sami ESP32 mikrokontroler, te se iz tog razloga poziva funkcija za dohvaćanje podataka putem HTTP protokola imena GET. Za istu je inicijalizirana cjelobrojna varijabla httpCode u koju će se po završetku dohvaćanja podatka spremiti kôd uspješno obavljene komunikacije (200) ili greške (npr. 403, 404, 500, 503). Ako nije dobiven odgovor poslužitelja, u prije spomenutu varijablu će biti upisana vrijednost nula.

```
137         int httpCode = http.GET();
```

U slučaju da je dobiven odgovor od poslužitelja (varijabla httpCode u sebi ima spremljenu vrijednost veću od nule), stvara se varijabla imena payload u koju se sprema dohvaćen podatak.

```
139         if (httpCode > 0) {
140               String payload = http.getString();
```

Kako bi se nad dobivenim podatkom mogla vršiti daljnja manipulacija, potrebno ga je pretvoriti u cjelobrojni tip varijable (eng. int) funkcijom imena payload.toInt.

```
144               int alarm_EPOCH = payload.toInt();
```

Na posljetku if funkcije, dohvaćeni podatak sprema se u jednodimenzionalnu matricu imena SPI_array za slanje putem SPI komunikacije na NUCLEO mikrokontroler.

```
147               SPI_array[4] = alarm_EPOCH;
148           }
```

U slučaju neuspješne komunikacije s poslužiteljem, aktivira se else funkcija koja varijabli prolaz povećava vrijednost za jedan. Naknadno tome izvršava se provjera vrijednosti prije spomenute varijable, te u slučaju da je njena vrijednost strogo manja od vrijednosti broja tri, cjelokupna funkcija GETAlarm pokreće se ponovo. Ako prethodni uvjet nije zadovoljen, varijabla prolaz se izjednačava s nulom, te se potom aktivira sekvenca za: odspajanje s bežične mreže, čišćenje relevantnih privremeno stvorenih varijabli, te potom ponovni pokušaj spajanja na mrežu.

```
150         else {
152           prolaz = prolaz + 1;
153
154           if(prolaz<3){
155               GETAlarm();
156           }else{
157               prolaz=0;
158               wifiDisconnect();
159               wifiClear();
160               wifiConnect();
161               GETAlarm();
162           }
163         }
```

Na posljetku funkcije poželjno je osloboditi resurse, te se iz toga razloga terminira veza prema poslužitelju.

```
165         http.end();
```

U slučaju da se na samome početku funkcije detektirala greška bežične mreže, pokreće se sekvenca otklanjanja grešaka i ponovnog povezivanja, te naposljetku pokretanje funkcije GETAlarm.

```
166     }else{
167           wifiDisconnect();
```

```
168             wifiClear();
169             wifiConnect();
170             GETAlarm();
171         }
172     }
```

Literatura

https://lms-2020.tvz.hr/course/view.php?id=425

https://lms-2020.tvz.hr/course/view.php?id=424

https://www.st.com/en/evaluation-tools/nucleo-f072rb.html

https://www.espressif.com/en/products/socs/esp32

https://www.arduino.cc

Popis slika

Popis tablica

Cjelokupni programski kôd za platformu NUCLEO

```
#include "mbed.h"
#include "TextLCD.h"

//shift registar
DigitalOut data(D10); //SER
//OE ne koristimo
DigitalOut RCLK(D9); //istoimeni
DigitalOut SRCLK(D8); //istoimeni
//SRCLRn ne koristimo

BusOut segme(D7, D6, D5, D4); //segmenti numerickog LCD-a

DigitalOut BUZZ(D2); //buzzer za alarm
DigitalOut snooze(D3); //gumb za prekidanje alarma

                //D11,  D12,  D13,  A2 po Arduino header-u
SPISlave spi_port(PA_7, PA_6, PA_5, PA_4); //MOSI, MISO, SCLK, SS //SPI
prema ESP32
I2C SHT3X(D14, D15); //SDA, SCL //I2C za senzor SHT30
Serial pc(USBTX, USBRX); //komunikacija prema PC-u
I2C i2c_lcd(I2C_SDA, I2C_SCL); //SDA, SCL //I2C setup za LCD16x2


struct tm  ts;
int second, minute, hour, day, month, year;
time_t seconds;

//ovo saljemo na shift registar koji onda pali dijelove segmenta koje smo
oznacili s logickom 1
//idu po redu, broj 0, broj 1, broj 2....broj 9
//const int array_data[] = {0b11000000, 0b11111001, 0b10100100, 0b10110000,
0b10011001, 0b10010010, 0b10000010, 0b11111000, 0b10000000, 0b10010000};
const int array_data[] = {0b00111111, 0b00000110, 0b01011011, 0b01001111,
0b01100110, 0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111};

//podaci sa ESP32
long SPI_array[6]={}; //ima 5 cjelina --> temp_morn, temp_day, temp_eve,
weather_id, Alarm_EPOCH, Time_EPOCH

//podaci sa SHT30
int temp, hum; //temperatura(°C) i vlaznost(%)

//LCD16x2
TextLCD_I2C lcd(&i2c_lcd, 0x4E, TextLCD::LCD16x2); //2C bus, PCF8574
Slaveaddress, LCD Type

//Moguci vremenski uvjeti
char arr[20][30] = {
    "No info","Storm", "Drizzle", "Rain", "Snow",
    "Mist", "Smoke", "Haze", "Dust", "Fog",
    "Sand", "Ash", "Haze", "Squall", "Tornado",
    "Clear", "Clouds"
};

char day_state[3][4] = {"MOR","DAY","EVE"}; //Prikazujemo 3 prognozirane
temperature
```

```
char stupnjevi=0b11011111; //kod za LCD za prikaz "°" kod prikaza
temperature
bool trg = 0, info=0; //pomocne varijable

int izmjena=0;
void display_main(void);
void update_date(void);

//funkcija koja prima koji broj zelimo prikazati i koji segment upaliti
void prikaz(char podaci, char segx){
    int out=0;

    //saljemo 8 bit podatak na shift registar osnovnom serijskom
komunikacijom
    for(int i=7;i>=0;i--){
        out=podaci>>i & 0b00000001; //kako se petlja vrti izoliramo jedan
po jedan podatak
        data=out; //postavimo podatak na pin
        wait_us(1);
        SRCLK=1; //obavijestimo shift registar da je podatak postavljen
        wait_us(1);
        SRCLK=0; //spustimo obavijest da odemo na sljedeci podatak
        out=0;
    }

    //jednom kada smo prenjeli kompletan podatak na shift registar
ukljucujemo
    //izlaze registra kako bi nam se podatak prikazao na numerickom ekranu
    RCLK=1; //clock za storage
    segme=segx; //paljenje segmenta numerickog ekrana

    izmjena++;
    if(izmjena == 2000){ //otprilike 10 sekundi (2000*5*10^-3)
        display_main(); //osvjezavanje LCD-a
        izmjena = 0; //resetiranje brojaca
    }else{
        wait_us(4950); //cekanje prije novog osvjezavanja numerickog ekrana
    }

    RCLK=0; //micanje podatka s izlaza shift registra
}

//pretvorba EPOCH u "normalno" vrijeme
void EPOCH_CP(void){
    time_t seconds = time(NULL);

    char second_S[3]; //mm
    char minute_S[3]; //mm
    char hour_S[3]; //hh
    char day_S[2]; //DD
    char month_S[2]; //MM
    char year_S[4]; //YY

    strftime(second_S, 3, "%S\n\r", localtime(&seconds));
    strftime(minute_S, 3, "%M\n\r", localtime(&seconds));
    strftime(hour_S, 3, "%H\n\r", localtime(&seconds));
    strftime(day_S, 3, "%d\n\r", localtime(&seconds));
    strftime(month_S, 3, "%m\n\r", localtime(&seconds));
    strftime(year_S, 4, "%y\n\r", localtime(&seconds));

    //naredba atoi nam sluzi za pretvorbu iz stringa (char) u int
```

```
    second = atoi(second_S);
    minute = atoi(minute_S);
    hour = atoi(hour_S);
    day = atoi(day_S);
    month = atoi(month_S);
    year = atoi(year_S);

    //pc.printf("Time: %d : %d\n\r", hour, minute);
    //pc.printf("Time: %d-%d-%d\n\r", day, month, year);

    //ALARM //u slucaju da nitko ne prekine alarm (gumbom) on ce se sam
zgasiti nakon 200 sekundi
    int trajanje=(SPI_array[4]+200);

    if(snooze == 1){ //gasenje alarma preko gumba
        SPI_array[4]=0;
        trajanje = 0;
        }

    //alarm se moze upaliti jedino ako je vrijednost EPOCH alarma veca od
trenutnog EPOCH-a ali u
    //isto vrijeme i manja od alarma+200 sekundi (trajanje alarma)
    //da bi se izbjegao alarm prilikom paljenja sustava (kada su sve
varijable nula) i resetiranja gumbom
    //dodano je da EPOCH alarma mora biti razlicit od nule
    if(seconds>=SPI_array[4] && seconds<=trajanje && SPI_array[4] != 0){
        BUZZ=!BUZZ;

    }else{
        BUZZ=0;
    }
}

//zove funkciju koja gura podatke u shift registar i pali korespondirajuci
segment
void urica(void){
    EPOCH_CP(); //refresh podataka za prikaz vremena
    prikaz(array_data[hour/10], 0b0111); //racunica i guranje serijom u
registar
    prikaz(array_data[hour%10], 0b1011); //racunica i guranje serijom u
registar
    prikaz(array_data[minute/10], 0b1101); //racunica i guranje serijom u
registar
    prikaz(array_data[minute%10], 0b1110); //racunica i guranje serijom u
registar
}

void SHT30(void){
    int adr=0x44; //za SHT3x adresa je 0x44 ako je GND ili 0x45 ako je 3.3V
    char podaci[2]= {0x2C, 0x10};
    char data[6]= {};
    int chk, pass=0; //varijable check (od I2C), temperatura i vlaznost

    do{
        chk = SHT3X.write(adr<<1, podaci, 2, true);
        //pc.printf("chk:%d pass:%d\n\r", chk, pass);
        if(pass != 0){
            wait(3);
            pass++;
        }else{
            pass++;
```

```
        }

    }while(chk != 0 && pass != 5);

    wait_ms(10);
    chk = SHT3X.read(adr<<1, data, 6, false);

    //spajanje MSB i LSB te sluzbene formule za tumacenje vrijednosti
    temp=(-45+175*(data[0]<<8 | data[1])/65535);
    hum=(100*(data[3]<<8 | data[4])/65535);

    //pc.printf("temp: %d °C\n\r", temp);
    //pc.printf("hum: %d %%\n\r", hum);
    //pc.printf("\n\r");
}

//dohvacanje podataka s ESP32
void SPI_receive(){
    char data_in[16]; //spremnik za prihvat svih dolaznih podataka
    int place=0; //pomocna varijabla

    //dobivamo sve podatke s ESP32 i spremamo ih
    for(int i=0; i<15; i++){
        data_in[i] = spi_port.read(); //primljeni podatak
    }

    //podaci temp_morn, temp_day, temp_eve sadrze 1 bajt (8 bitova)
podataka te se oni prvi odvajaju iz spremnika dolaznih podataka
    for(place=0; place<3; place++){
        SPI_array[place] = data_in[place];
    }

    //podaci weather_id, Alarm_EPOCH, Time_EPOCH sadrze 4 bajt-a (32 bita)
podataka te trebaju drugaciju obradu od prethodnih podataka
    for(int i=3; i<6; i++){
        //pomicanje i spajanje dijelova podatka
        SPI_array[i] = (long)data_in[place] | (long)data_in[place+1]<<8 |
(long)data_in[place+2]<<16 | (long)data_in[place+3]<<24;
        place = (place + 4); //vodenje racuna o poziciji unutar spremnika
za prihvat ulaznih podataka
    }

    set_time(SPI_array[5]); //sinkronizacija unutarnjeg (NUCLEO) RTC-a s
pristiglim podatkom
    EPOCH_CP(); //separacija vremenskog stringa u zasebne varijable

    info = 1; //micanje teksta "No information" s LCD-a

    lcd.cls();
    display_main(); //inicijalno postavljanje podataka na LCD
    update_date(); //inicijalno postavljanje datuma na LCD


}

//LCD16x2
int prognoza(){
    //branchless programing
    return 0+1*(SPI_array[3] >= 200 && SPI_array[3] <= 232)+
    2*(SPI_array[3] >= 300 && SPI_array[3] <= 321)+
    3*(SPI_array[3] >= 500 && SPI_array[3] <= 531)+
```

```
    4*(SPI_array[3] >= 600 && SPI_array[3] <= 622)+
    5*(SPI_array[3] == 701)+
    6*(SPI_array[3] == 711)+
    7*(SPI_array[3] == 721)+
    8*(SPI_array[3] == 731)+
    9*(SPI_array[3] == 741)+
    10*(SPI_array[3] == 751)+
    11*(SPI_array[3] == 761)+
    12*(SPI_array[3] == 762)+
    13*(SPI_array[3] == 771)+
    14*(SPI_array[3] == 781)+
    15*(SPI_array[3] == 800)+
    16*(SPI_array[3] >= 801 && SPI_array[3] <= 804);
}

//napravljeno kao zasebna funkcija da ne troši procesorsko vrijeme jer se
rijetko mijenja
void update_date(void){
    lcd.locate(0,0); //pozicioniranje na LCD-u, (stupci, redovi)
    lcd.printf("%.2d/%.2d/%.2d", day, month, year);
}

int looping=0; //pomocna varijabla koja omogucuje izmjenu temperaturnih
prognoza
void display_main(){
    looping++;
    if(looping==3){
        looping=0;
    }

    trg = !trg; //pomocna varijabla za izmjenu izmedu Tin i Hin

    if(!trg){
        lcd.locate(9,0);
        lcd.printf("Tin:%.2d%c", temp, stupnjevi);

        lcd.locate(0,1); //prelazimo u sljedeci red
        if(info){ //ako je bio obavljen prijenos podataka s ESP32
            lcd.printf("%s:%.2d%c  ", day_state[looping],
SPI_array[looping], stupnjevi);

            lcd.locate(9,1);
            lcd.printf("%s", arr[prognoza()]);
        }else{//ako nije bio obavljen prijenos podataka s ESP32
            lcd.printf(" No information ");
        }

    }else{
        SHT30(); //dohvacanje nove temperature i vlaznosti s senzora
        lcd.locate(9,0);
        lcd.printf("Hin:%.2d%%", hum);

    }
}

int main(){
    spi_port.format(8,0); //SPI -> 8 bitni podatak, mod rada 0
    spi_port.frequency(1125000); // brzina SPI komunikacije

    i2c_lcd.frequency(400000); //Fast-mode da bi se skratio treptaj
numeriskoga ekrana
```

```
    SHT3X.frequency(400000);

    EPOCH_CP(); //refresh podataka za vrijeme

    //LCD16x2
    lcd.setCursor(TextLCD::CurOff_BlkOff);
    lcd.setBackLight(TextLCD::LightOn);

    display_main(); //inicijalno postavljanje podataka na LCD
    update_date(); //inicijalno postavljanje datuma

    while(1){
        if(spi_port.receive()){ //cekamo da master signalizira pocetak
            SPI_receive();
        }else{
            urica();

            if(second == 0 && minute == 0 && hour == 0){ //refresh datuma u
ponoc
                update_date();
            }
        }//else
    }//while(1)
}//int main()
```

Cjelokupni programski kôd za platformu ESP-WROOM-32

```
#include <WiFi.h> //wifi veza
#include <ESP32Ping.h> //brze povezivanje na wifi
#include <HTTPClient.h> //prognoza
#include <Arduino_JSON.h> //za izvlacenje podataka iz odgovora openweathr-a
#include <time.h> //dohvacanje trenutnog vremena za NUCELO
#include <SPI.h> //Master (ESP32) prema NUCLEO-u

//WiFi podaci
const char* ssid = "*********";
const char* pass = "*********";

//openweathermap.org API call
const String poveznica =
"https://api.openweathermap.org/data/2.5/onecall?lat=45.8144&lon=15.978&exc
lude=current,minutely,hourly&appid=e5941c3f91ab77e9ac98e22589d3c7ba&units=m
etric";

//podaci za Network Time Protocol (NTP)
const char* ntpServer = "2.europe.pool.ntp.org"; //EU server
const long gmtOffset_sec = 3600; //GMT+1 zona
const int daylightOffset_sec = 3600; //ljetno/zimsko racunanje vremena

//SPI
static const int SPI_brzina = 1125000; // 1.125 MHz, NUCLEO ne moze 1MHz
SPIClass * vspi = NULL; //SPI
int SPI_array[7]; //array u koji se spremaju svi podaci koji ce se poslati
preko SPI-a

//Gumb za refresh podataka
const int buttonPin = 14;
int buttonState = 0;


////// Kod vezan za spajanje na wifi \\\\\\

bool wifiStatus=false;
void wifiClear(){
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    wifiStatus = WiFi.status() == WL_CONNECTED;
    delay(100);
}

void wifiDisconnect(){
    WiFi.disconnect();
    WiFi.mode(WIFI_OFF);
    wifiStatus = false;
}

bool ping_res=false;
void wifiConnect(){
    WiFi.setSleep(false);
    IPAddress ip(1, 1, 1, 1); // The remote ip to ping

    wifiStatus=false;
    int loop_counter=0;

    do{
```

```
        Serial.println("Wifi connecting...");
        WiFi.begin(ssid, pass);
        delay(100);

        ping_res = Ping.ping(ip);
        Serial.println(ping_res);

        loop_counter++;

        if(loop_counter>=5){
            wifiDisconnect();
            wifiClear();
        }

        if(loop_counter>=10){
            ESP.restart();
        }
    }while(ping_res == false);
   Serial.println("Wifi connection established");
}

void check_connection(){
  IPAddress ip(1, 1, 1, 1); // The remote ip to ping
  ping_res = Ping.ping(ip);
}

////// kraj koda vezanog uz spajanje na wifi, SPI slanje podataka u
nastavku \\\\\\

//funkcija za slanje podataka putem SPI-a
//svi podaci koji se trebaju poslati su spremljeni u SPI_array
void SPI_send(void){
  char data_out[16]; //8 bitova po jednom prijenosu, u ovu varijablu
spremamo razdvojene podatke
  int place=0; //varijabla za pracenje mjesta upisa podataka u array

  //prva tri podatka (temperature) su velicine 1 bajta
  for(int i=0; i<3; i++){
    data_out[place] = SPI_array[i]; //direktno prebacivanje
    place++; //pracenje pozicije
  }

  //druga tri podatka su velicine 4 bajta
  for(int i=3; i<6; i++){ //tri podatka
    for(int t=0; t<4; t++){ //rastavljanje 4 bajta u dijelove po 1 bajt
      data_out[place]=SPI_array[i]>>(8*t) & 0xFF;
      place++;
    }
  }

  vspi->beginTransaction(SPISettings(SPI_brzina, MSBFIRST, SPI_MODE0));
//MS bit ide prvi, MOD 0
  digitalWrite(5, LOW); //spustanje "Slave select" linije kako bi se NUCLEO
znao pripremiti
  delay(15);

  //jedan po jedan podatak (velicine 1 bajt) se vadi iz matrice te šalje
prema NUCLEO-u
  for(int i=0; i<15; i++){
    vspi->transfer(data_out[i]);
    delay(15);
```

```
  }

  delay(10);
  digitalWrite(5, HIGH); //dizanje "Slave select" linije kako bi NUCLEO
znao da je prijenos zavrsio
  vspi->endTransaction();
}
////// kraj koda vezanog uz slanje podataka putem SPI protokola, dohvacanje
vremena(EPOCH) u nastavku \\\\\\

//Dohvacanje trenutacnog EPOCH vremena
void GETLocalTime(){
  time_t EPOCH;
  struct tm timeinfo;

  if (!getLocalTime(&timeinfo)) {
    Serial.println("Greska u dohvacanju vremena");
  }
  time(&EPOCH); //sinkonizacija s ESP32 RTC-om
  EPOCH = EPOCH + gmtOffset_sec + daylightOffset_sec; //dodavanje GMT+1 i
ljetno/zimsko racunanje vremena

  SPI_array[5] = EPOCH; //spremanje u matricu za slanje
}

////// kraj koda vezanog uz dohvacanje alarma s online baze, dohvacanje
prognoze u nastavku \\\\\\

int prolaz=0;
void GETAlarm(void){
  if ((WiFi.status() == WL_CONNECTED)) { //Provjera statusa internetske
veze
    HTTPClient http;


http.begin("https://esp32testserver.000webhostapp.com/seminarski/index.php"
); //link na bazu
    int httpCode = http.GET(); //dohvacanje podataka

    if (httpCode > 0) { //ako su podaci dohvaceni
        String payload = http.getString(); //spremanje dohvacenih podataka
        Serial.println(httpCode);
        Serial.println(payload);

        int alarm_EPOCH = payload.toInt(); //konverzija String --> int
(EPOCH je dohvacen)
        alarm_EPOCH = (alarm_EPOCH + 3600); //GMT+1, 3600 sekundi = 1 sat

        SPI_array[4] = alarm_EPOCH; //spremanje u matricu za slanje
      }

    else {
      Serial.println("Error on HTTP request");
      prolaz = prolaz + 1;

      if(prolaz<3){
        GETAlarm();
      }else{
        prolaz=0;
        wifiDisconnect();
        wifiClear();
```

```
        wifiConnect();
        GETAlarm();
      }
    }

    http.end(); //Deinicijalizacija
  }else{
      wifiDisconnect();
      wifiClear();
      wifiConnect();
      GETAlarm();
  }
}

////// kraj koda vezanog uz dohvacanje vremena(EPOCH), dohvacanje prognoze
u nastavku \\\\\\

//univerzalna funkcija za dohvacanje podataka s interneta
String httpGETRequest(const char* serverName){
    HTTPClient http;

    // Your IP address with path or Domain name with URL path
    http.begin(serverName);

    // Send HTTP POST request
    int httpResponseCode = http.GET();

    String payload = "{}";

    if (httpResponseCode>0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      payload = http.getString();
    }
    else {
      Serial.print("Error code: ");
      Serial.println(httpResponseCode);

      Serial.print("Reconnecting to WiFI... ");
      wifiConnect();
    }
    // Free resources
    http.end();
    Serial.println(payload);
    return payload;
}

//funkcija za dohvacanje podataka o prognozi
void GETWeatherinfo(){
    check_connection();
    if(ping_res == true) { //Check the current connection status
      String jsonBuffer; //varijabla za RAW JSON odogovor na API call prema
openweather-u
      jsonBuffer = httpGETRequest(poveznica.c_str()); //dohvacanje podataka

      JSONVar myObject = JSON.parse(jsonBuffer); //rastavljanje JSON-a

      // JSON.typeof(jsonVar) can be used to get the type of the var
      if (JSON.typeof(myObject) == "undefined") {
        Serial.println("Parsing input failed!");
        return;
```

```
      }

      //primjer za parsiranje iz {}
      //Serial.println(myObject["main"]["temp"]);

      //primjer za parsiranje iz arraya []
      //JSONVar main_weather=myObject["weather"][0]["main"];

      //TEMPERATURE
      long int temp_morn=myObject["daily"][0]["temp"]["morn"]; //float npr.
"3.52" (izrazeno u °C, bez navodnika)
      SPI_array[0] = temp_morn;
      printf("%d\n\r", temp_morn);

      long int temp_day=myObject["daily"][0]["temp"]["day"]; //int npr. "3
(izrazeno u °C, bez navodnika)
      SPI_array[1] = temp_day;
      printf("%d\n\r", temp_day);

      long int temp_eve=myObject["daily"][0]["temp"]["eve"]; //float npr.
"3.52" (izrazeno u °C, bez navodnika)
      SPI_array[2] = temp_eve;
      printf("%d\n\r", temp_eve);

      //VREMENSKI UVJETI
      long int weather_id=myObject["daily"][0]["weather"][0]["id"]; //npr.
"Clouds" (bez navodnika)
      SPI_array[3] = weather_id;
      printf("%d\n\r", weather_id);


    }else{
          Serial.println("WiFi Disconnected, reconnecting...");
          wifiConnect();
          GETWeatherinfo();
     }

}

////// kraj koda vezanog uz dohvacanje prognoze, opca inicijalizacija u
nastavku \\\\\\

void update_ALL(void){
    GETWeatherinfo(); //funkcija za dohvacanje prognoze i slanje iste putem
SPI-a
    GETAlarm(); //funkcija za dohvacanje posatavljenog alarma i slanje
istog putem SPI-a
    GETLocalTime(); //funckija za dohvacanje trenutnog vremena i slanje
istog putem SPI-a
    SPI_send();
}

void setup(){
    //pocetak serijske komunikacije
    Serial.begin(115200);

    //wifi start
    wifiConnect();

    //inicijalizacija postavki za dohvacanje vremena (sati)
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
```

```
    //SPI inicijalizacija
    vspi = new SPIClass(VSPI);
    vspi->begin(); //SCLK = 18, MISO = 19, MOSI = 23, SS = 5 (standaradni
pinovi)
    pinMode(5, OUTPUT); //VSPI SS

    attachInterrupt(14, update_ALL, FALLING);
}

void loop(){
  GETWeatherinfo(); //funkcija za dohvacanje prognoze
  GETAlarm(); //funkcija za dohvacanje posatavljenog alarma
  GETLocalTime(); //funckija za dohvacanje trenutnog vremena
  SPI_send();
  delay(3600000);
  //}
}
```

Cjelokupni programski kôd internetske stranice namijenjene korisničkom upisu vremena budilice

```
<!DOCTYPE html>
<html>
<head>
	<title>Dodavanje alarma</title>
	<style>
		body {
			text-align: center;
		}
	</style>
</head>

<body>

<h1>Dodavanje novoga alarma</h1>
<h2>Molimo upišite vrijeme novoga alarma:</h2>

<form action="fni02u8FFFF2983B.php" method="post">

Vrijeme: <input type="time" name="vrijeme" /><br><br>
Datum: <input type="date" name="datum" /><br><br>

<input type="submit" />

</form>

</body>
</html>
```

Cjelokupni programski kôd internetske stranice za upis podataka u MySQL bazu

```
<html>
<body>
<?php
$servername = "localhost";
$dbname = "**********";
$username = "**********";
$password = "**********";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql="INSERT INTO Seminarski_alarm (vrijeme, datum)

VALUES ('$_POST[vrijeme]','$_POST[datum]')";

if ($conn->query($sql)) {
  echo "Novi alarm uspješno dodan";
} else {
  echo "Greška: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
</body>
</html>
```

Cjelokupni programski kôd internetske stranice namijenjene dohvaćanju, pretvorbi i ispisu postavljene budilice u EPOCH formatu

```
<?php

$servername = "localhost";
$dbname = "**********";
$username = "**********";
$password = "**********";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT vrijeme, datum FROM Seminarski_alarm WHERE id=(SELECT
max(id) FROM Seminarski_alarm)";

if ($result = $conn->query($sql)) {
    while ($row = $result->fetch_assoc()) {
        $row_vrijeme = $row["vrijeme"];
            $row_datum = $row["datum"];

            $EPOCH_datum = strtotime($row_datum);

            $EPOCH_vrijeme = explode(":", $row_vrijeme);

            $EPOCH_total = $EPOCH_datum + $EPOCH_vrijeme[0]*3600 +
$EPOCH_vrijeme[1]*60 + $EPOCH_vrijeme[0];
            echo $EPOCH_total;
    }
    $result->free();
}
$conn->close();
?>
```