WiFi meteorološka postaja s e-ink zaslonom

Biro, Borna

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

Sveučilišni studij

WIFI METEOROLOŠKA POSTAJA S E-INK ZASLONOM

Diplomski rad

Borna Biro

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 30.10.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Borna Biro
Studij, smjer:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1084, 04.10.2017.
OIB studenta:	71555424583
Mentor:	Izv. prof. dr. sc. Davor Vinko
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Marijan Herceg
Član Povjerenstva 1:	Izv. prof. dr. sc. Davor Vinko
Član Povjerenstva 2:	Izv. prof. dr. sc. Krešimir Grgić
Naslov diplomskog rada:	WiFi meteorološka postaja s e-ink zaslonom
Znanstvena grana rada:	**Elektronika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Zadatak diplomskog rada je razvoj i izrada meteorološke postaje koja prikuplja podatke o temperaturi zraka, vlazi u zraku, tlaku zraka i količini CO2 u zraku. Prikupljene podatke je potrebno spremiti u memoriju iz koje se ti podaci kasnije mogu grafički predočiti. Podatke je potrebno prikazati na eInk ekranu niske potrošnje, rezolucije 800x600. Meteorološka postaja treba imati mogućnost spajanja na internet putem bežične mreže.Za više informacija javiti se mentoru: davor.vinko@ferit.hrTemu rezervirao Borna Biro
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	30.10.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTIRADA

Osijek, 04.11.2021.

Ime i prezime studenta:	Borna Biro
Studij:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1084, 04.10.2017.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: WiFi meteorološka postaja s e-ink zaslonom**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Davor Vinko

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
2. METEOROLOGIJA 2
2.1. Praćenje vremena i vremenska prognoza 2
2.2. Meteorološke stanice 4
3. KOMPONENTE METEOROLOŠKE POSTAJE (STANICE) 7
3.1. Zaslon 7
3.1.1. Odabir EPD panela 11
3.2. Mikroupravljač 13
3.2.1. ESP32 mikroupravljač 16
3.2.2. STM32L073 mikroupravljač 18
3.3. API Servisi 19
3.3.1. OpenWeatherMaps API servis 20
3.4. Panel osjetljiv na dodir (*touchscreen panel*) 21
3.5. Senzori 24
3.6. Bežična komunikacija 27
3.7. Napajanje 29
4. IZRADA METEOROLOŠKE POSTAJE (STANICE) 32
4.1. *Reverse engineering* principa rada *e-paper* zaslona 32
4.2. Izrada unutrašnje jedinice 46
4.2.1. Izrada sklopovlja za panel osjetljiv na dodir 47
4.2.2. Problem s internim *RTC*-om ESP32 mikroupravljača 49
4.3. Izrada vanjske jedinice 50
4.4. Izrada senzora vjetra i sunčevog zračenja 56
4.4.1. Senzor brzine vjetra 56
4.4.2. Senzor smjera vjetra 61
4.4.3. Senzor sunčevog zračenja 65
4.5. Programski kod 67
4.5.1. Unutrašnja jedinica 68
4.5.2. Vanjska jedinica 72

5. REZULTATI 78
5.1. Rukovanje unutrašnjom jedinicom 80
5.2. Rukovanje vanjskom jedinicom 85
6. ZAKLJUČAK 90
LITERATURA 91
SAŽETAK 94
ABSTRACT 95
ŽIVOTOPIS 96
PRILOG 97

1. UVOD

Živimo u svijetu kada se događaju vrlo velike i značajne promjene vremena. Da bi se ljudi mogli bolje i kvalitetnije organizirati, potrebno je znati kakvo nas vrijeme može očekivati, pogotovo kada su u pitanju vanjske aktivnosti. U tome nam uvelike može pomoći vremenska prognoza. Vremenska prognoza nam govori kakvo nas vrijeme očekuje slijedećih par dana. No, vremenska prognoza kakvu većina ljudi ima na mobilnim uređajima nije previše detaljna ili sami podaci nisu vrlo logično prikazani, pa je ponekad vrlo teško odlučiti kakvo nas vrijeme očekuje. Kako je prethodno i rečeno, mobilni uređaji imaju na sebi mogućnost prikaza vremenske prognoze, no ponekad kratki pogled na mobitel može nas udaljiti od trenutnog posla, stoga je bolje imati uređaj koji je samo za prikaz vremenske prognoze. Osim toga, ponekad postoji potreba da ljudi znaju kakvi su uvjeti u prostoru u kojem se trenutno nalaze (poput temperature, vlage, tlaka zraka), kao i za vanjske uvjete (trenutna vanjska temperatura, vlaga, tlak zraka, UV zračenje, itd). Postoje komercijalni uređaji koji obavljaju ovakvu zadaću, no većina njih nemaju mogućnost prikazivanja vremenske prognoze s Interneta, kao i mogućnost praćenja izmjerenih veličina. Ovaj uređaj će to omogućavati.

U prvom dijelu rada ćemo se upoznati s pojmom meteorologije i same vremenske prognoze, kao i sa gotovim uređajima (meteorološkim stanicama) kako za kućnu upotrebu, tako i za profesionalnu upotrebu. Nakon toga, u 3. poglavlju biti će opisana problematika izrade meteorološke postaje, te će biti opisan svaki dio koji se koristi u njoj, kao i koji su zahtjevi postavljeni za određeni dio meteorološke postaje. U narednom poglavlju će biti opisani problemi vezani za programski kod, dijelovi programskog koda koji će se nalaziti u pojedinom mikroupravljaču, te koji su se problemi pojavili na sklopovskoj razini, te kako su riješeni. U zadnjem poglavlju će biti opisan gotov rad, kako se rukuje s njime, te će biti prikazani rezultati, poput potrošnje energije, trajanja baterije, primjer izmjerenih podataka i sl.

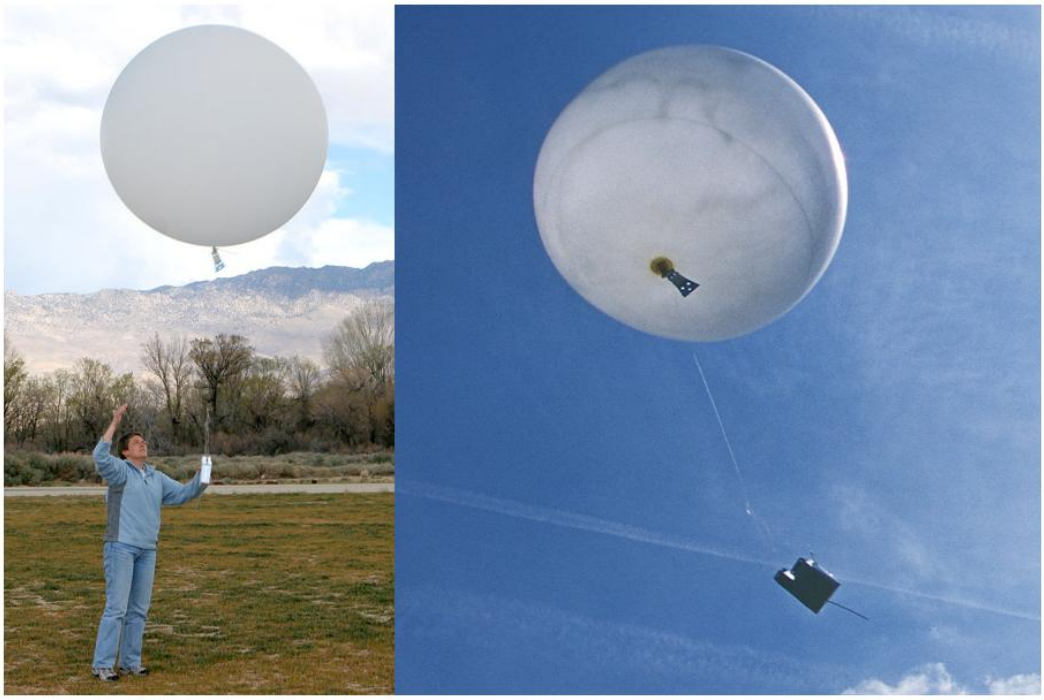
2. METEOROLOGIJA

Meteorologija (grčki *meteoros* – nalazi se, lebdi u zraku, *logos* – riječ, govor, um, učenje) je znanost koja proučava strukturu i sastav Zemljine atmosfere, njezino fizičko stanje, kao i postanak, razvoj i karakter meteoroloških pojava koje se pojavljuju.[1] Meteorologiju najčešće povezujemo s vremenskom prognozom i meteorološkim pojavama (poput kiše, snijega, vjetra, oluje itd).

2.1. Praćenje vremena i vremenska prognoza

Praćenje vremenskih prilika može se vršiti na dva načina; sa zemlje, pomoću raznih meteoroloških postaja (slika 2.1.) koje mjere trenutne vrijednosti temperature, tlaka zraka, vlage zraka, količine čestica u zraku, UV zračenje, brzine i smjera vjetra, radara, te iz zraka pomoću satelitskih snimki i meteoroloških balona (slika 2.2).

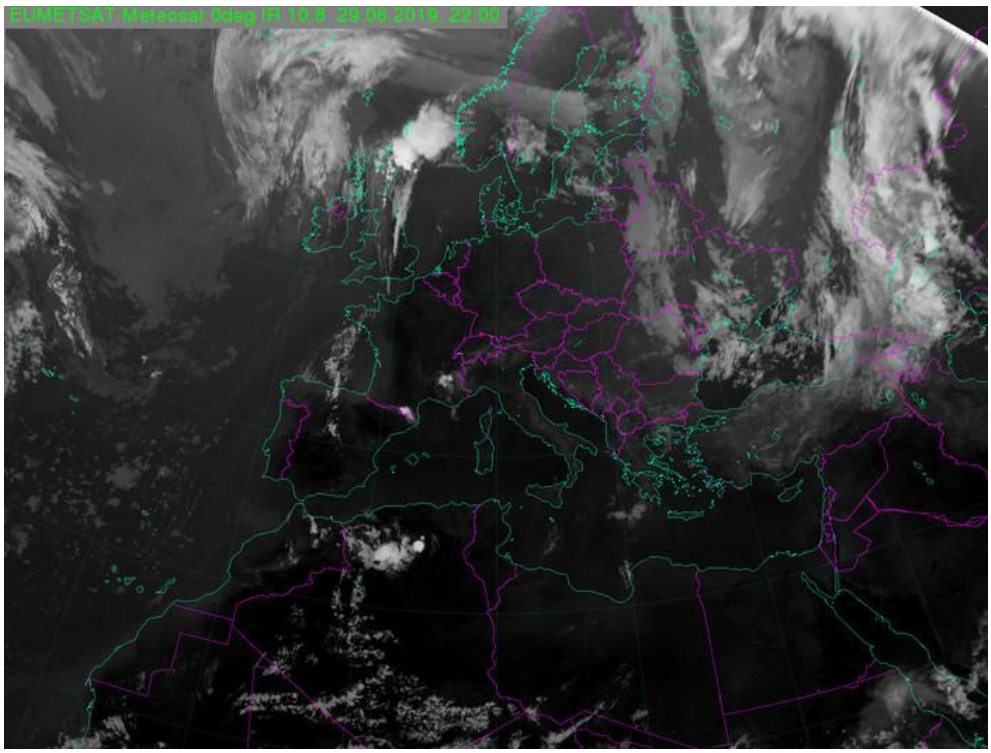
Slika 2.1. Meteorološke postaje [2]



Slika 2.2. Meteorološki balon [3]

Meteorološke postaje se dijele na sinoptičke, zrakoplovne, klimatološke, agrometeorološke, oborinske i specijalne.[4] Vrsta koja se najčešće koristi u dnevnim izvještajima su sinoptičke, gdje se one dalje dijele na površinske i visinske. Površinske se koriste za mjerenja u prizemnom sloju atmosfere i najčešće su viđena od strane ljudi. To su najčešće kučice u kojima se nalazi mjerna oprema ili samo mjerna oprema bez neke trajne zaštite. One dalje mogu biti kopnene ili brodske. One mjere podatke poput temperature zraka, tlaka zraka, vlage zraka, točke orošavanja, količinu oborina, pokrivenost oblacima, brzinu i smjer vjetra, vidljivost i ostale parametre. Postaja ne mora nužno mjeriti sve prethodno navedene podatke, već može samo neke od njih. Osim ovih podataka, mogu mjeriti i količinu peludnih čestica u zraku, kao i količinu raznih kemijskih spojeva u zraku, poput ozona i ugljičnog dioksida. Osim toga, meteorološke postaje mogu biti automatske, gdje su nekom vrstom komunikacijske veze povezane s nekim drugim uređajem (npr. računalo) i gdje meteorološke postaje same periodički šalju podatke o izmjerenim veličinama. Takve stance su vrlo dobre, jer ne zahtijevaju ljude da očitavaju i zapisuju izmjerene podatke. Najčešće se koriste u nekim nepristupačnim područjima.

Satelitske snimke (slika 2.3) su glavni alat kada se želi napraviti vremenska prognoza. One mogu pokazati količinu oblaka u atmosferi kao i smjer gibanja naoblake i visinu na kojoj se oblaci nalaze. Osim toga, pomoću infracrvenog spektra mogu pokazati temperaturu tla.[5]



Slika 2.3. Satelitska snimka Europe [6]

Na kraju, kada se svi ti podaci prikupe, potrebno ih je obraditi, te iz tih obrađenih podataka dobiti vremensku prognozu za naredne dane. Ona se dobiva iz raznih numeričkih atmosferskih prognostičkih modela. Državni hidrometeorološki zavod za Republiku Hrvatsku koristi dva modela; ALADIN za trodnevnu vremensku prognozu i ECMWF model za sedmodnevnu vremensku prognozu. Različiti modeli daju različite rezultate s različitom točnošću vremenske prognoze za različita zemaljska područja. O tome koliko je model dobro napravljen i koliko podataka prati i uzima ih u obzir, ovisi i točnost same vremenske prognoze.

2.2. Meteorološke stanice

Praćenje i analiza meteoroloških podataka može sve vršiti kod kuće pomoću raznih kućnih meteoroloških stanica zvane najčešće „vremenske stanice“. Vremenske stanice (slika 2.4) su uređaji

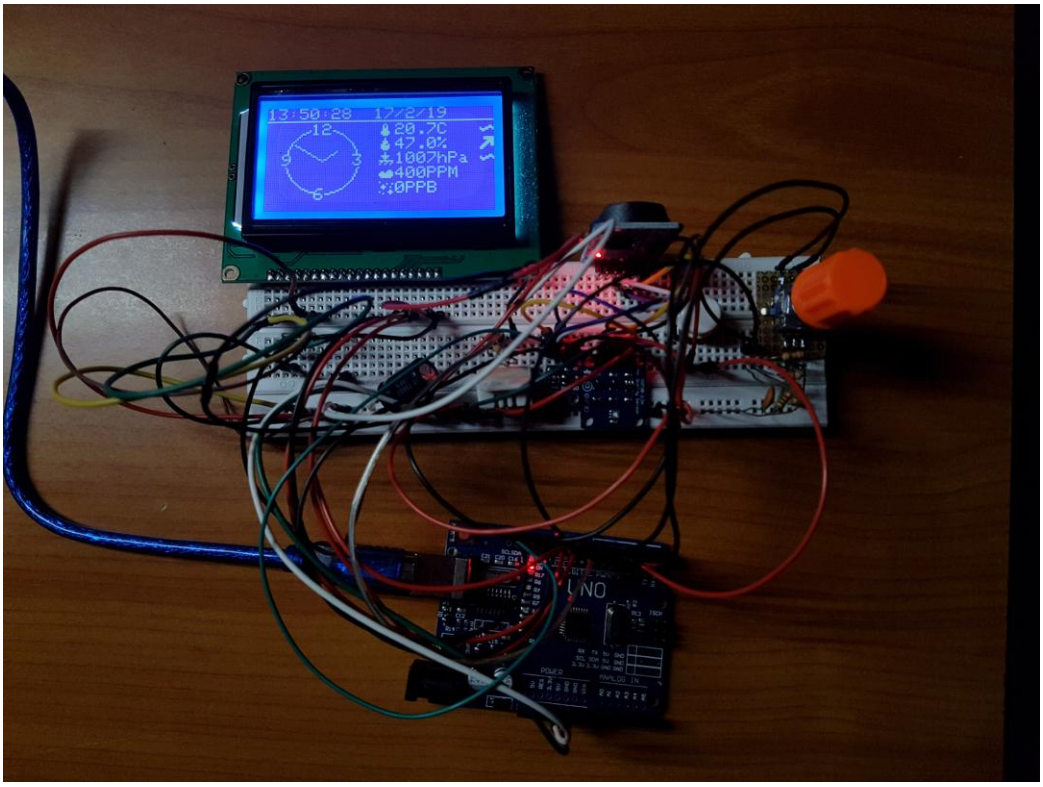
koji se sastoje od unutrašnje jedinice i jedne ili više vanjskih jedinica. Vanjske jedinice imaju različite senzore u sebi koje mjernu veličinu pretvara u električni signal. Jednostavnije vanjske jedinice sadrže samo senzor temperature, no one kompleksnije mogu sadržavati i ostale senzore poput anemometra i senzora za količinu oborina.

Slika 2.4. Vremenska stanica s vanjskom jedinicom [7]

Takve stanice mogu na temelju izmjerenih podataka vršiti vrlo jednostavne vremenske prognoze za par sati unaprijed najčešće na temelju promjene tlaka zraka. Ukoliko je tlak nizak ili tlak pada, tada ide lošije vrijeme, a ukoliko tlak raste ili je visok, tada ide povoljno vrijeme. Osim vanjskih parametara, one mogu pratiti i parametre u prostoru u kojem se nalazi unutrašnja jedinica. Takva jedinica prati temperaturu, tlak i vlagu zraka, no kompleksnije mogu pratiti i količinu CO2 kao i organska onečišćenja u zraku. Unutrašnja jedinica sadrži zaslon na kojem ispisuje izmjerene podatke. Tehnologije zaslona mogu biti različite i variraju od stanice do stanice. Najčešće osim izmjerenih podataka, daju informaciju o trenutnome vremenu (sat) i datumu, kao i mjesečeve mijene.

S porastom popularnosti elektronike kao hobija, tako je i porasla količina vremenskih stanica napravljena od strane ljudi koji se bave elektronikom. Ovakav tip vremenskih stanica (slika 2.5) je vrlo teško grupirati u neke podjele, jer način rada, broj senzor vanjskih jedinica, izgled, parametri

koji se prate i mjere i niz ostalih faktora ovisi o tome što osobu koja pravi tu vremensku stanicu interesira, no isto tako daje jednu vrstu fleksibilnosti, jer ju sami dizajner može napraviti kako njemu odgovara.



Slika 2.5. Digitalna vremenska stanica kućne izrade

Osim što takve vremenske stanice mogu mjeriti, prikazivati i obrađivati podatke, mogu svako mjerenje spremiti na neki mediji poput memorijske kartice ili ih pak postaviti na Internet. Na taj način se podaci mogu pratiti i skladištiti, te ih iskoristiti za daljnju obradu na uređaju s većom računskom moći od same vremenske stanice.

3. KOMPONENTE METEOROLOŠKE POSTAJE (STANICE)

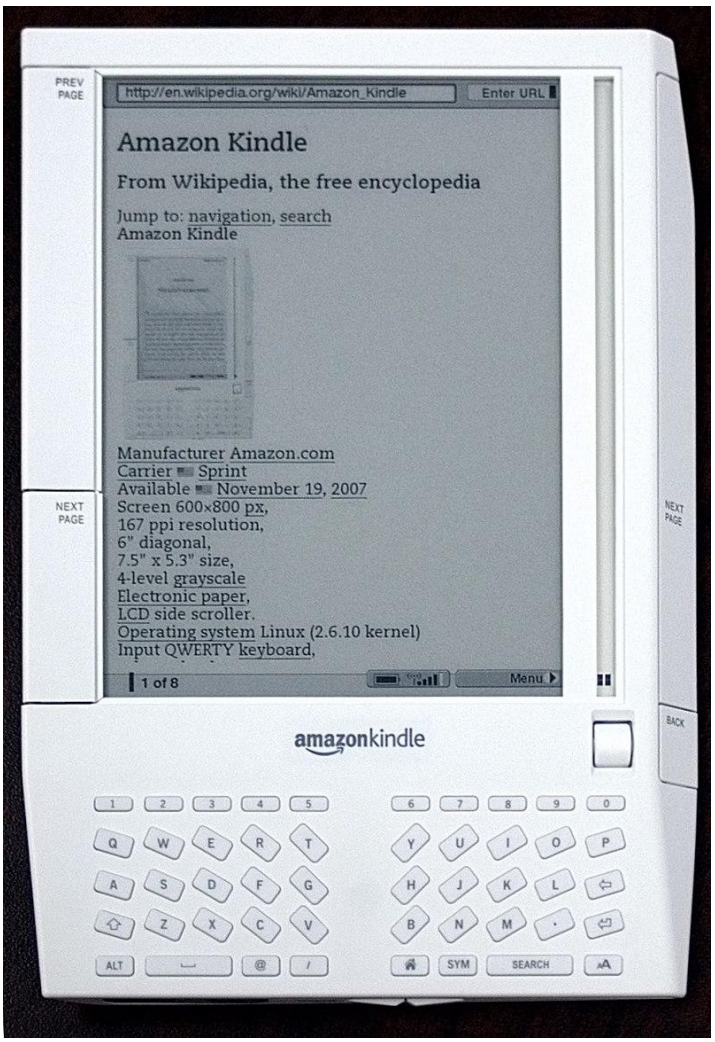
Meteorološka postaja koja se izrađuje u ovome radu se sastoji od više cjelina koje su međusobno povezane. Glavna podjela je na unutrašnju jedinicu i vanjsku jedinicu. Unutrašnja jedinica je zaslužna za spajanje na Internet, mjerenje unutrašnjih parametara poput temperature zraka, relativne vlažnosti zraka, tlak zraka, količine CO2, prikaz podataka i samu interakciju s korisnikom, a vanjska je zaslužna za mjerenje vanjskih parametara poput vanjske temperature zraka, relativne vlažnosti zraka, tlaka zraka, brzine i smjera vjetra, količine UV zračenja, količine sunčevog zračenja, slanje parametara unutrašnjoj jedinici i jednostavan prikaz podataka korisniku. Najvažnije cjeline unutrašnje jedinice su zaslon, zaslužan za prikaz traženih informacija, mikroupravljač koji upravlja cijelim sustavom, omogućava spajanje na bežičnu Internet mrežu (putem WiFi tehnologije), slanje zahtjeva za vremenskom prognozom putem API servisa, obradu podataka, slaganje prikaza, te slanje istog na zaslon, čitanje unosa podataka od strane korisnika pomoću zaslona osjetljivog na dodoir (engl. *Touchscreen*), radijskog primopredajnika za komunikaciju s vanjskom jedinicom i na kraju senzor koji mjerene podatke (temperatura, vlaga i tlak zraka) pretvara u neku digitalnu riječ.

Cijela meteorološka postaja mora biti izvedena na taj način (programski i sklopovski) da troši što je moguće manje energije, te da se spaja na bežičnu Internet mrežu, osvježava zaslon i vrši radijsku komunikaciju samo kada je to zbilja potrebno. Sve ostalo vrijeme, sklopovlje treba biti u režimu rada s vrlo malo potrošnjom energije (engl. *Low Power Mode*). Razlog tomu je taj da se omogući što dulji vijek trajanja baterije, bez da ju je potrebno često puniti.

3.1. Zaslon

Zaslon (engl. *display*) koji se koristi u meteorološkoj postaji je EPD (engl. *Electronic Paper Display*, ostali nazivi su *e-ink, e-paper display*), odnosno bistabilni zaslon. Bistabilni zasloni su vrsta zaslona kod koje zapis na njemu ostaje trajno upisan, sve dok se ponovno ne zada promjena prikaza. Za razliku od klasičnih LCD (engl. *Liquid Crystal Display*), OLED (engl. *Organic Light Emitting Diode*), LED (engl. *Light Emitting Diode*), plazma, VFD (engl. *Vacuum Flourescent Display*) zaslona, ovakvi zasloni ne zahtijevaju stalno napajanje da bi održali prikaz na sebi, već samo zahtijevaju napajanje (energiju) kada se mijenja prikaz. Ovakvi zasloni se sve više koriste u elektroničkim uređajima radi ekstremno niske potrošnje energije (satovi, mobiteli, razni *gadeti*), a najveću popularnost je stekao kada se implementirao u elektronički čitač knjiga (slika 3.1.). Jedan

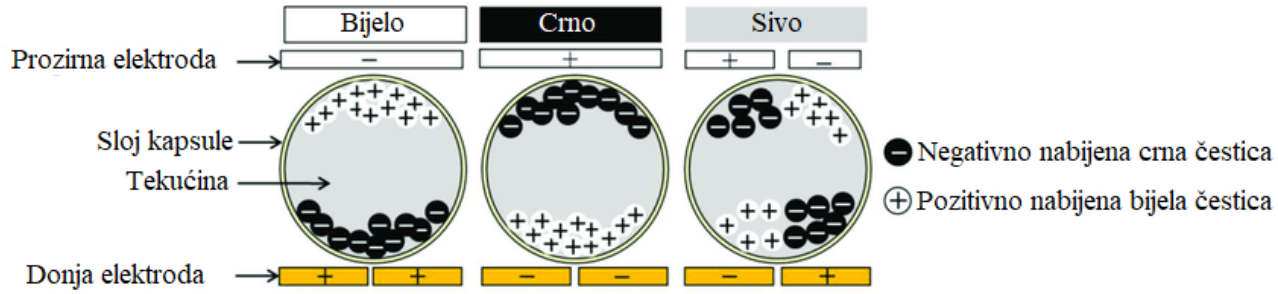
takav uređaj je razvila tvrtka Amazon, a uređaj se zvao Kindle. Prvi takav uređaj je bio Sony Libre i Sony Reader, pa zatim Amazon Kindle. Takvi uređaji su mogli da naprave 7500 promjena stranica, prije nego se baterija isprazni.



Slika 3.1. Amazon Kindle [8]

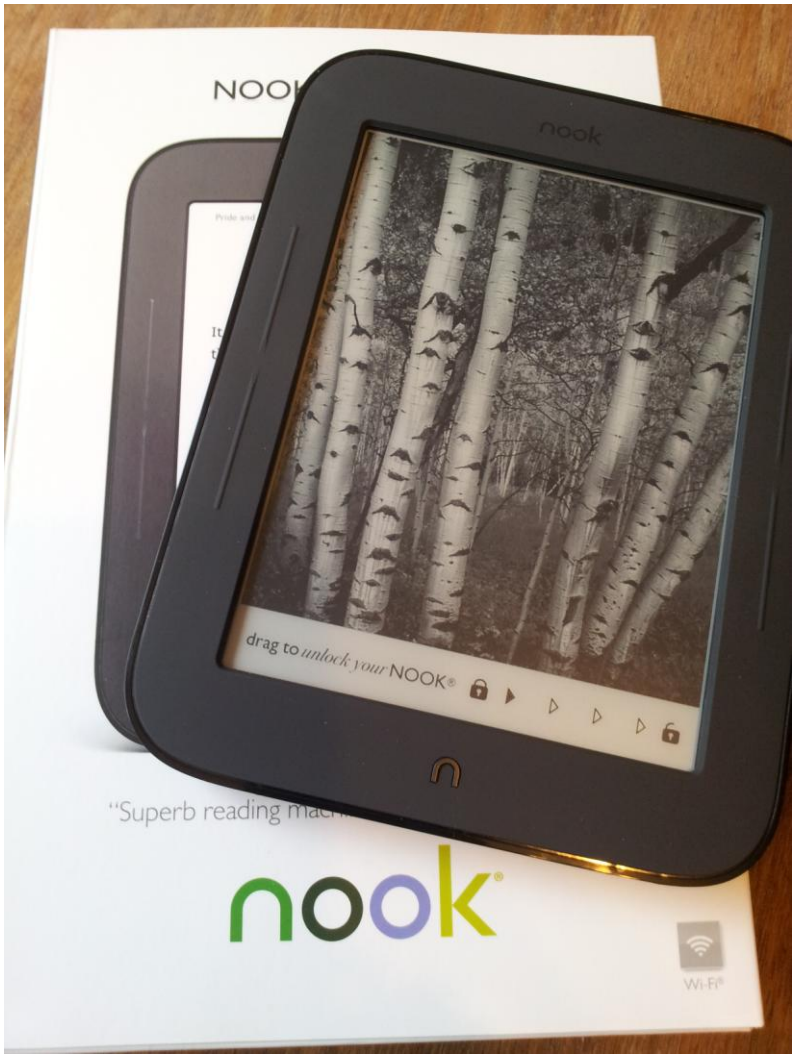
Postoji više tehnologija rada e-paper zaslona. Trenutno najčešće korištena jest izvedba s mikrokapsulama (engl. *Microencapsulated electrophoretic display*). Tehnologija rada se bazira na principu da se između dvije elektrode, koje se nalaze na gornjem i donjem staklu, nalaze male, mikroskopske kapsule koje su iznutra popunjene s vrlo viskoznom tekućinom. U njima se nalaze dvije vrste čestica, crne i bijele. Bijele reflektiraju svijetlo prema osobi koja gleda u panel, crna čestica apsorbira svijetlost, čime se to mjesto čini crno na zaslonu. Čestice imaju različite električne

naboje na sebi, pa su tako crne čestice negativno nabijene, dok su bijele pozitivno nabijene (slika 3.2).



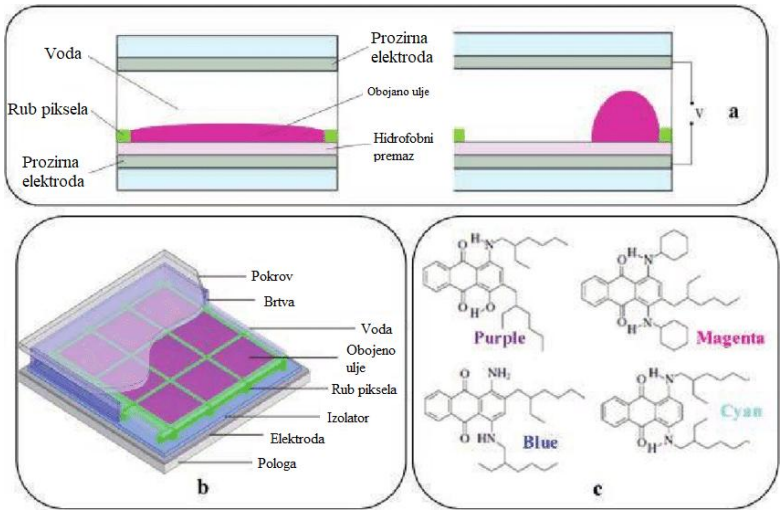
Slika 3.2. Princip rada EPD-a s mikrokapsulama [9]

Ukoliko bi željeli da određeni piksel bude u crnoj boji, potrebno je gornju (prozirnu elektrodu) postaviti na pozitivni potencijal, a donju na negativni. Tada će crne čestice isplivati prema gore, te tako biti vidljive, a bijele će potonuti dolje i time ostati manje vidljive korisniku. S ovakvim zaslonima je moguće i stvarati prikaze s nijansama sive. Ovdje postoje dva načina, s tim da je drugi način puno više korišten u praksi. Prvi način je taj da se postavi više elektroda ispod jedne kapsule, te da se jedan dio kapsule nabije pozitivnim, a druga negativnim nabojem. Rezultat toga je da gornja polovica kapsule sadrži i crne i bijele čestice, što za rezultat daje sivu boju. S ovom metodom moguće je izvesti samo crnu, sivu i bijelu boju, što je ujedno i mana ove metode. Moguće je povećati broj elektroda, pa i samim time povećati broj nijansi sive, no radi dimenzija mikrokapsula od promjera od svega 40 μm, to bi bilo vrlo teško izvedivo. Druga metoda se bazira na sljedećem principu: ukoliko se želi sivi piksel, omogućimo da bijela čestica potone dolje do kraja, te zatim crnu stavimo da bude negdje u sredini. Na taj način crna čestica neće moći u potpunosti apsorbirati svu svjetlost koja je usmjerena na nju, već samo dio, te će taj piksel izgledati sivo (slika 3.3.).



Slika 3.3. Ispis slike s nijansama sive na *e-paperu* [10]

Druga tehnologija *e-papera* koja je još uvijek u razvoju jest *Electrowetting* tehnologija. Zasniva se na tome da se između dvije elektrode, od kojih jedna ima hidrofobni premaz na sebi nalazi voda i obojeno ulje. Kada se na elektrode dovede napon, ulje se pomakne u kut, te bude manje vidljivo, što je objašnjeno na slici 3.4. Kada se ukloni napon s elektroda, ulje se razlije, te oboja piksel, time piksel u boji ulja bude vidljiv.



Slika 3.4. Princip rada *Electrowetting E-Paper* zaslona [11]

Ovakvi zasloni mogu biti u boji, te imaju dosta velik *refresh rate*, što im omogućava da se koriste u uređajima koji *e-paper* zasloni s mikrokapsulama nisu mogli (reklamni izlozi, ekrani na mobitelima, monitori, itd). Treba napomenuti da i jedna i druga tehnologija ne emitira nikakvu svijetlost, već se bazira na reflektiranju postojeće svjetlosti koja se nalazi u prostoru u kojem se koriste. Ovo je vrlo velika prednost na spram klasičnih zaslona, jer omogućavaju da se smanji naprezanje očiju kod ljudi, a time i da se smanji mogućnost bolesti očiju. No, to im je ujedno i mana, jer ne mogu se koristiti gdje nema svijetlosti.

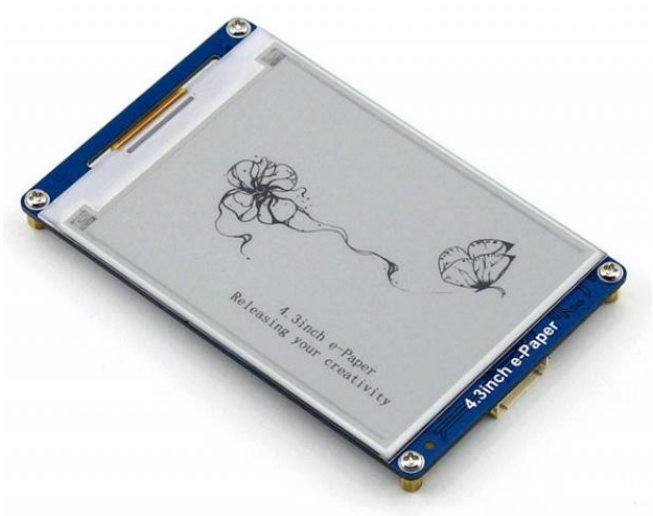
3.1.1. Odabir EPD panela

Postoje više vrsta i modela *e-paper* panela, stoga je potrebno odabrati onaj koji bi karakteristikama što bolje odgovarao ovom radu. Odabir se radio na temelju slijedećih karakteristika:

- Veličina zaslona (dijagonala)
- Rezolucija zaslona
- Vrsta komunikacije s zaslonom
- Kontrast
- Cijena
- Potrebno sklopovlje za upravljanje s zaslonom

Moguće je odabrati dvije rute; onu s gotovim komercijalnim rješenjima ili onu gdje će se uzeti sami zaslon, te će se naknadno stvoriti sklopovlje koje će s njime upravljati, kao i programska

podrška. Kod odabira nekih od komercijalnih rješenja, problem je bila vrlo mala dimenzija (kao kod Waveshare 4.3 Inch E-Paper Display, slika 3.5), iako je rezolucija bila zadovoljavajuća (800x600 piksela), kao i vrsta komunikacije (Serijska komunikacija), te sklopovlje (zaslon je u obliku modula).



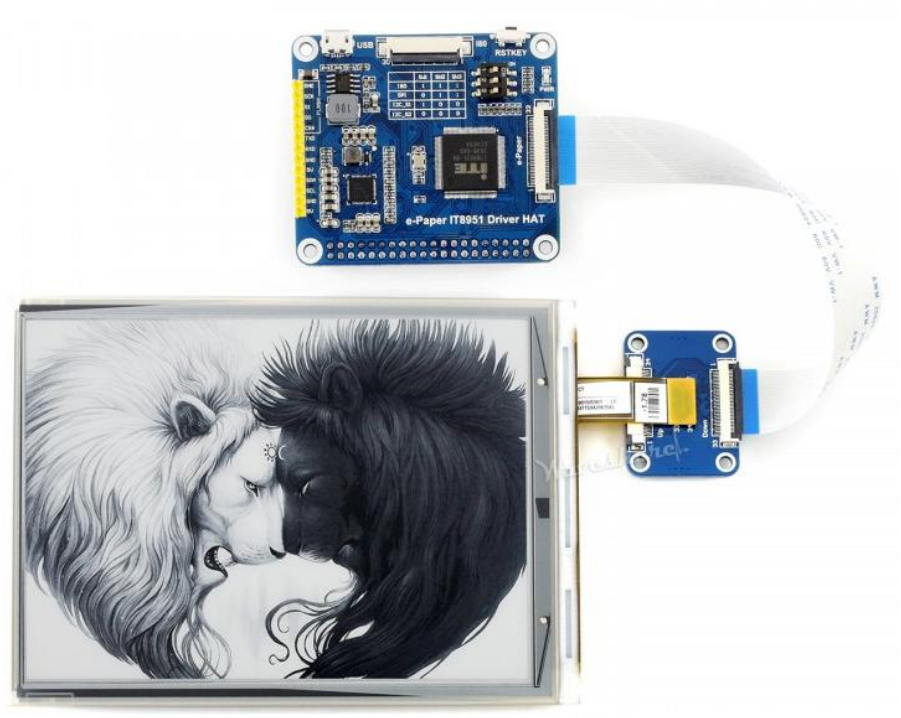
Slika 3.5. Waveshare 4.3 Inch E-Paper Display [12]

Od iste firme postoje zasloni i većih dimenzija (7.5inch e-Paper HAT, slika 3.6), no zato imaju manju rezoluciju (640x384 piksela) nego prethodno navedeni.

Slika 3.6. Waveshare 7.5inch e-Paper HAT [13]

Također, postoje zasloni i većih dimenzija (dijagonale 6 inča) i većih rezolucija (800x600) kao što je prikazan na slici 3.7, no, imaju vrlo veliko i kompleksno sklopovlje koje bi teško bilo ugraditi u

meteorološku stanicu, a da pri tome potrošnja bude mala i da jednostavnost upravljanja bude što veća.



Slika 3.7. Waveshare 6inch e-Paper HAT [14]

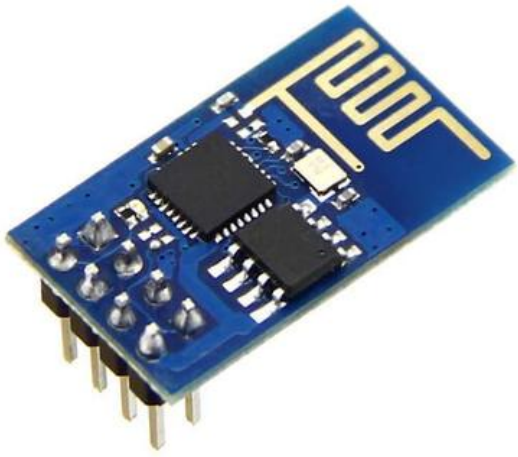
3.2. Mikroupravljač

Mikroupravljač je glavni dio ovoga rada. On upravlja s cijelim sustavom; od komunikacije s senzorima, do komunikacije s Internetom, radijske komunikacije, obrade primljenih podataka s Interneta, osvježavanje zaslona, detekcija unosa podataka od strane korisnika putem zaslona osjetljivog na dodir (engl *Touchscreen*), itd. Radi različitih zahtjeva, mikroupravljači unutrašnje i vanjske jedinice mogu biti različiti, stoga se od mikroupravljača za unutrašnju jedinicu zahtjeva da bude dovoljno brz (ponajviše za što brži i kvalitetniji ispis slike na zaslonu), da ima dovoljno RAM memorije (za skladištenje cijele trenutne slike za zaslonu, engl. *Frame buffer* i za skladištenje i obradu primljenih podataka s interneta), da ima dovoljno ulaza i izlaza, te da ima dovoljno programske memorije (FLASH memorije), dok se za mikroupravljač za vanjsku jedinicu zahtjeva da ima vrlo malu potrošnju, bogatu periferiju (što se tiče komunikacije, analognih ulaza), dovoljan broj ulaza i izlaza. Neki od mogućih kandidata se nalaze u obliku razvojne pločice ili u obliku modula. To su sklopovi koji imaju na sebi ugrađen programibilni mikroupravljač, kao i programsku podršku

(razvojno okruženje). Neki od poznatih mikroupravljača koji dolaze u obzir za unutrašnju jedinicu su:

- ATMEGA328P (Arduino Uno i Arduino Nano razvojna pločica)
- ATMEGA2560 (Arduino Mega R3 razvojna pločica)
- STM32F103C8T6 (Bluepill razvojna pločica)
- STM32F407VGT6
- STM32H743ZIT6
- ESP8266 (ESP8266-12E Modul)
- ESP32 (WROOM32 i WROVER moduli)

Svaki od prethodno navedenih mikroupravljača ima svoje prednosti i mane. ATMEGA328P je dobar izbor što se tiče cijene, dostupnosti i broja ulaza i izlaza, kao i vrlo dobre programske podrške kroz Arduino IDE razvojno okruženje, no problemi su vrlo mala programska memorija (32 kB), vrlo mala RAM memorija (2 kB), te relativno spor radni takt (16 – 20 MHz). ATMEGA2560 ima nešto veću programsku memoriju od 256 kB i 8 kB RAM memorije, no to i dalje nije dovoljno da bude korišten u ovoj meteorološkoj postaji. Treba najmanje 128 kB RAM memorije za skladištenje trenutnoga prikaza, 60 kB za obradu i skladištenje podataka s interneta, te najmanje 100 kB za razne ikonice koje bi trebale biti spremljene u programskoj memoriji i korištene u prikazu na zaslonu. STM32F103C8T6 bi bio dobar kandidat za ovaj rad, no problem je ponovno mala RAM memorija koja bi se mogla povećati korištenjem vanjske RAM memorije u obliku integriranog kruga, no tada bi se izgubio značajan broj ulaza i izlaza. Taj problem bi se mogao zaobići korištenjem STM32F407VGT6 ili STM32H743ZIT6, jer imaju dovoljno slobodnih ulaza i izlaza čak i kada bi se koristila vanjska RAM memorija, no ono što svi prethodno navedeni mikroupravljači nemaju je mogućnost spajanja s Internetom. Rješenje bi bilo korištenje bežičnog WiFi modula (slika 3.12) ili Ethernet modula (slika 3.13), no to bi povećalo kompleksnost, broj komponenata, a na kraju otežalo upravljanje, te dovelo do veće potrošnje električne energije.



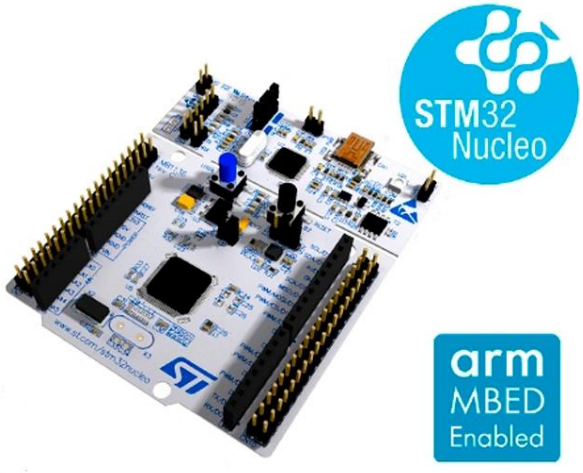
Slika 3.12. WiFi modul [15]

Slika 3.13. Ethernet modul [16]

ESP8266 i ESP32 mikroupravljači imaju integrirane bežične Internet module (koristeći WiFi tehnologiju), imaju dovoljno RAM memorije, kao i programske memorije, no ono što je problem kod ESP8266 mikroupravljača je taj što nema dovoljno ulaza i izlaza za kontrolu *e-paper* zaslona. Zaslon zahtjeva 15 ulaza i izlaza s mikroupravljača za slanje podataka, te još 4 za kontrolu napajanja, što ukupno daje 19 ulaza i izlaza. Povećanje broja izlaza može se postići putem posmačnih registara (engl. *Shift Registers*) koji bi serijsku SPI (engl. *Serial Peripheral Interface*) komunikaciju pretvarao u paralelnu, no tada bi se značajno izgubilo na brzini. Stoga, jedini

mikroupravljač koji bi karakteristikama odgovarao ovome radu jest ESP32. Njega je moguće programirati kroz Arduino IDE razvojno okruženje uz instaliranje njegovog dodatka.

Što se tiče izbora mikroupravljača za vanjsku jedinicu svi prethodno navedeni mikroupravljači dolaze u izbor (osim ESP8266 zbog izuzetno malog broja ulaza i izlaza), no puno bolji izbor je STM32L0 serija mikroupravljača, čija je glavna karakteristika izuzetno mala potrošnja energije, jer je namijenjen za režim rada s niskom potrošnjom (engl. *Low Power Mode*). Najbolji odabir su STM32L053 i STM32L073 zbog toga što za navedene modele postoji razvojna pločica (vidljiva na slici 3.14) pomoću koje je razvoj vanjske jedinice olakšan (ne treba se odmah razvijati prototip na tiskanoj pločici) i zbog toga što postoji Arduino programska podrška za oba modela, kao i službena podrška u obliku programskog paketa CubeMX i Atollic TrueSTUDIO for STM32.



Slika 3.14. Nucleo-64 razvojna pločica [17]

3.2.1. ESP32 mikroupravljač

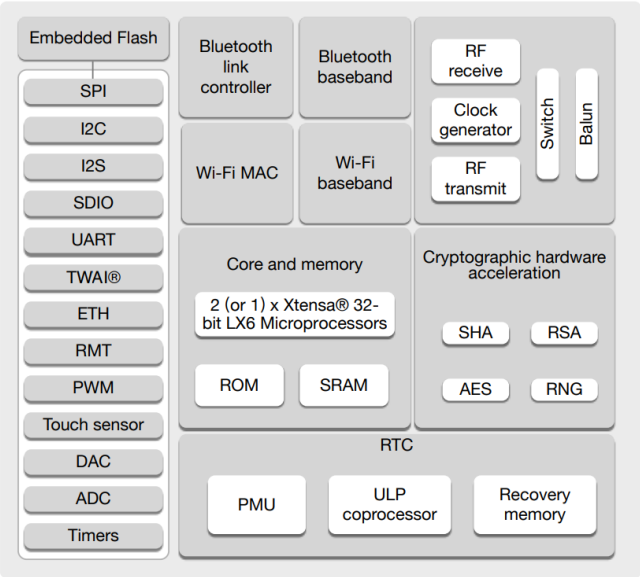
Odabrani mikroupravljač je ESP32 u obliku modula koji ima oznaku ESP32-WROVER (vidljiv na slici 3.15). Taj modul osim što sadrži ESP32 mikroupravljač, sadrži i FLASH programsku memoriju od 4 MB, te PSRAM od 8 MB što je i više nego dovoljno za *Frame Buffer e-paper* zaslona. Također sadrži i integriranu WiFi antenu s dodatnim IPX priključkom na koji se može spojiti vanjska WiFi antena radi boljeg prijama (s tim da se prethodno *jumper* prebaci u poziciju da se koristi vanjska antena). Mana ovoga mikroupravljača su brzina PSRAMa koja je teoretski ograničena na 40 MB/s, a u stvarnosti je puno niža, reda oko 13 MB/s što je dovoljno za ispis slike u crno-bijeloj boji, te 4 ili 8 nijansi sive boje i kako je prije navedeno mali broj

upravljačkih ulaza i izlaza (GPIO). Spajanje na Internet je također olakšano jer ESP32 mikroupravljač u sebi sadrži WiFi, koji ima programsku podršku kroz Arduino WiFi biblioteke.

Slika 3.15. ESP32 WROVER modul [18]

Modul je također vrlo pogodan za ovaj rad jer ima režim rada s niskom potrošnjom energije (engl. *Low power mode*). Na ovaj način, mikroupravljač će biti većinu vremena u režimu rada s niskom potrošnjom, a raditi će tek kada postoji neka interaktivnost od strane korisnika, dok se osvježavaju podaci ili dok se spaja na WiFi mrežu. Ovakav način rada nam omogućava da smanjimo potrošnju energije na najmanju moguću mjeru. Kada se mikroupravljač nalazi u režimu niske potrošnje energije, moguće ga je prebaciti u normalan režim rada dovođenjem *Interrupt* signala na njegove ulaze ili putem sata realnog vremena (engl. *RTC – Real Time Clock*) koji će stvoriti alarmni signal i prebaciti mikroupravljač iz režima niske potrošnje u režim normalnoga rada. Prva mogućnost je korisna kada korisnik zahtjeva interaktivnost s uređajem. Druga mogućnost je korisna kada sam uređaj treba obnoviti podatke o vremenu putem interneta. Upravljanje s *e-paperom* se vrši pomoću tzv. *bit-banging*-a gdje se komunikacijski protokol pokušava emulirati preko programskoga koda. Problem ove metode je da je relativno spora, tj. najveći *clock rate* koji se može postići jest 6.67 MHz što je dvostruko manje od onoga što zahtjeva panel (12 MHz). Duga mogućnost bi bila da se koristi paralelna I2S komunikacija putem DMA modula. Ovo omogućava da *clock rate* bude i do 40 MHz, te omogućava da se vrši neki drugi dio koda dok DMA modul izbacuje van podatke putem I2S komunikacije. Mana je što je na ovome mikroupravljaču dokumentacija za taj dio jako loša i

vrlo je teško napraviti da to radi stabilno i pouzdano. Kompletna unutrašnja struktura s kompletnom periferijom je nalazi na slici 3.16.



Slika 3.16. Unutrašnji blok-dijagram ESP32 mikroupravljača [19]

Ostala periferija koja može biti korisna jest UART komunikacija (primarno za svrhe *debug*-a), I2C komunikacija, SPI komunikacija, RTC (engl. *Real Time Clock*) i njegova memorija, ADC (engl. *Analog to Digital Converter*).

3.2.2. STM32L073 mikroupravljač

Mikroupravljač koji će se koristiti za vanjsku jedinicu jest STM32L073CZT3, a razvoj će se raditi na STM32L053R8T6 koji se nalazi na NUCLEO-L053 razvojnoj pločici. Razlog ovomu je nedostupnost NUCLEO-L073 razvojne pločice i STM32L053R8T6 mikroupravljača u trenutku izrade ovoga rada. Pošto su razlike minimalne (glavna razlika je samo u količini memorije i broju dostupnih ulaza i izlaza, gdje STM32L073CZT3 ima 192 kByte-a programske FLASH memorije / 20 kBytea RAM-a i 37 ulaza i izlaza, a STM32L053R8T6 ima 64 kByte-a programske FLASH memorije / 8 kByte-a RAM-a i 51 ulaz i izlaz, a ostale, manje bitne razlike su količina identičnih periferijskih sklopovlja), razvoj se radi jednostavnosti može raditi na jednome, a zatim se na završnome radu može koristiti drugi, uz manje prepravke koda. Neke od bitnih karakteristike su:

- Dva 12 bitni analogno digitalni pretvornik
- Tri I2C komunikacijska sučelja

- Četiri USART / UART komunikacijska sučelja
- Četiri SPI komunikacijska sučelja
- Pet izvora glavnoga takta
- Radni napon od 1.65 VDC do 3.6 VDC
- Integrirani RTC s mogućnošću „buđenja“ mikroupravljača

Ostatak periferije i sami blok dijagram mikroupravljača se može pronaći u prilogu (prilog 1) [20]

3.3. API Servisi

Da bi meteorološka postaja mogla prikazivati vremensku prognozu potrebna je mogućnost spajanja na internet odakle bi se vremenska prognoza i mogla preuzeti. No, problem je što sam mikroupravljač nema dovoljno resursa (brzine i memorije) da učita cijelu Web stranicu napisanu u HTML-u ili nekom drugom programskome jeziku, obradi je i prikaže nam bitne podatke. K tome, sam mikroupravljač nema nikakvu programsku podršku za ijedan web programski jezik, već sve podatke s interneta gleda kao tekstualne podatke (ASCII kod). Stoga je potrebno pronaći način da se cijela vremenska prognoza može dostaviti kao čisti tekstualni podatak. Ovakav problem rješavaju API servisi. API servis (engl. *Application Programming Interface*) omogućava interakciju s Web poslužiteljem putem vrlo jednostavnih tekstualnih poruka na razini ASCII koda. Zahtjev se najčešće radi na radini samoga linka koji je ubačen u HTTP GET zahtjev ili pak može biti dio HTTP POST zahtjeva gdje podaci mogu biti ubačeni i sami podatkovni dio HTTP-a u slučaju da je poruka izuzetno duga ili ukoliko se želi napraviti neka jednostavna zaštita podataka. Odgovor može biti u čistome tekstualnome obliku ili pak kako to bude češće, u obliku JSON zapisa (engl. *JavaScript Object Notation*). Primjer izgleda JSON zapisa vidljiv je na slici 3.17. se može vidjeti Pomoću raznih programskih biblioteka, mikroupravljač može vrlo jednostavno obraditi podatke i prikazati vremensku prognozu uz korištenje vrlo malo memorije i resursa.

```json
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}
```

Slika 3.17. JSON zapis [21]

3.3.1. OpenWeatherMaps API servis

API servis nije obavezna stvar za svaku web stranicu (ili web poslužitelja), stoga je potrebno pronaći poslužitelja za vremensku prognozu koji ima tu opciju. Najčešće se takva opcija dodatno naplaćuje, pa je također jedan od kriterija da poslužitelj ima mogućnost besplatnoga korištenja API servisa (čak i ako ima smanjenje mogućnosti u odnosu na verziju s plaćanjem). Neki od popularnijih servisa za vremensku prognozu s podrškom za API servisom su:

- Open Weather Maps
- The Weather Channel
- ACCU Weather
- The Dark Sky
- Weatherbit

Svaki od njih ima svoje prednosti i mane, te je potrebno odabrati što najviše odgovara ovome radu. Open Weather Maps je jedan od najčešće korištenih API servisa za vremensku prognozu. Ima mogućnost besplatnoga plana koji ima veliki broj opcija (trenutno vrijeme, prognoza za 5 dana svaka tri sata, upozorenja na iznimno loše vrijeme, tzv. engl. *Weather alerts*, vrijeme za 48 sati unaprijed s podacima za svaki sat, a ponekad i za svaku minutu unutar jednoga sata, ovisno o lokaciji). Servis ima ograničenje na 60 poziva na jednu minutu, odnosno 1000000 poziva mjesečno, što je i više nego dovoljno za ovaj rad. Problem ovoga servisa je da dosta često za biti nedostupan i da je točnost loša, jer ponekad nema podatke za zadano mjesto, pa da podatke od većeg mjesta koje je u blizini. The Weather channel je besplatan samo za probu, što znači da nakon određenoga

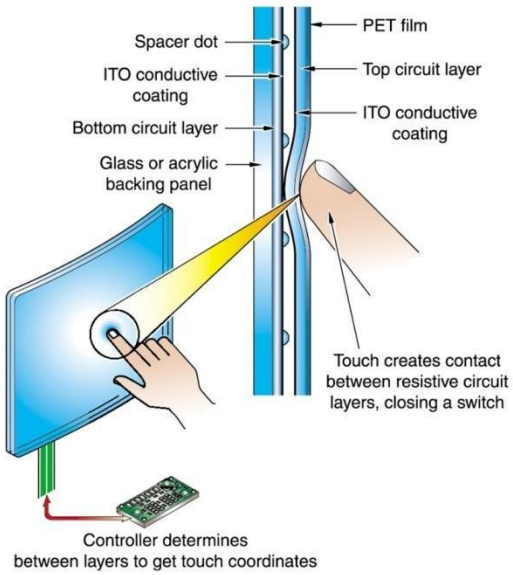
perioda se gubi mogućnost da se nastavi koristi taj servis. K tome, dokumentacija je vrlo teško dostupna i sam servis nije namijenjen za manje korisnike. ACCU Weather ima besplatnu inačicu koja je vrlo ograničena na samo 50 API poziva na dan i ima samo trenutne podatke o vremenu, prognozu 5 dana unaprijed i detaljniju prognozu 12 sati unaprijed, no za razliku od Open Weather Maps servisa ima puno bolju točnost i manje ispade s radom. The Dark Sky je bio vrlo dobar alternativa Open Weather Maps servis sve do 1.8.2020. godine kada je otkupljen od tvrtke Apple Inc. te je postao nedostupan za ostale korisnike. Weatherbit također radi vrlo velikih ograničenja od 500 poziva na dan i malog broja mogućnosti ne dolazi kao jedna od opcija. Stoga servis koji će biti korišten na radu jest Open Weather Maps bez obzira na njegove mane.

3.4. Panel osjetljiv na dodir (*touchscreen panel*)

Da bi korisnik mogao imati nekakvu interakciju s uređajem (unos podataka, podešavanje postavki, pregled prognoze, itd), potrebno je da postoji nekakav ulazni uređaj. Postoje razne metode koje omogućuju da korisnik ima interakciju s uređajem, neke od njih su tipke (ili tipkovnica), računalo ili mobitel, razni senzori pokreta i gesti, itd. U ovom radu odlučeno je da će se koristiti panel osjetljiv na dodir (koji će u kombinaciji s zaslonom stvarati zaslon osjetljiv na dodir). Panel osjetljiv na dodir je transparenta ploha (najčešće napravljena od transparentne plastike ili stakla) koja se postavlja na zaslon i koja prevodi dodir u električni signal. Razlog da se koristi *touchscren* kao ulazni uređaj proizlazi iz više stvari; prvi razlog je da smo danas okruženi velikim brojem uređaja koji kao glavni ulazni uređaj imaju zaslon osjetljiv na dodir (npr. pametni telefon), te nam je „prirodan“ način interakcije s uređajem putem njega, drugi je da nam on omogućava stvaranje grafičkog korisničkog sučelja (engl. *GUI – Graphical User Interface,* slika 3.18), gdje se postiže puno veća i jednostavnija interakcija i veći broj mogućnosti, nego kada se koriste tipke i tipkovnice, treći je da se smanjuje broj ulaza na mikroupravljaču potrebnih da bi se omogućio rad panela (npr. kod tipaka ili tipkovnice, broj ulaza na mikroupravljaču je jednak broju tipaka ukoliko se ne koristi *multiplexing* ili nekakva slična metoda koja smanjuje broj ulaza, no povećava ili programsku ili sklopovsku kompleksnost, što ovdje nije moguće radi režima rada s niskom potrošnjom energije).

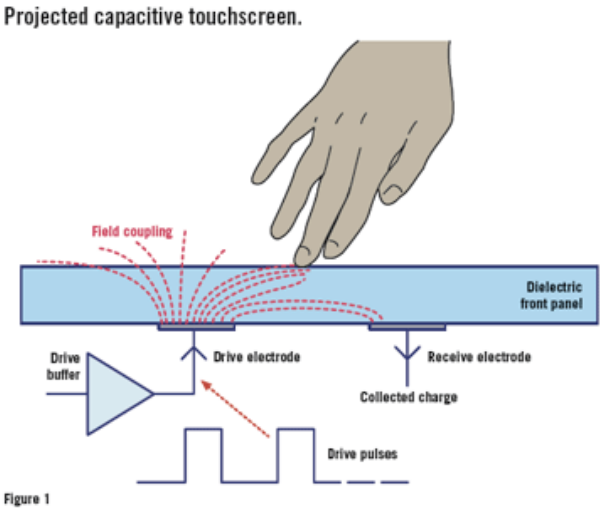
Slika 3.18. GUI sučelje s zaslonom osjetljivim na dodir na Arduino platformi [22]

Treba napomenuti da zaslon i sami panel koji je osjetljiv na dodir ne moraju nužno biti u cjelini, već mogu biti odvojeni kao u ovome slučaju. Kod izbora panela (ili zaslona) osjetljivih na dodir, postoje više tehnologija: Rezistivni paneli osjetljivi na dodir, kapacitivni paneli osjetljivi na dodir, infracrveni paneli osjetljiv na dodir i paneli osjetljivi na dodir s akustičnim površinskim valom. Ovisno o odabranoj tehnologiji, svaki od njih ima svoje prednosti i mane. Rezistivni (engl. *Resistive Touchscreen*) je panel koji se sastoji od dvije transparentne ili semi-transparente plohe koje imaju vodljivi sloj na sebi. Na jednoj od njih se nalaze male „točkice“ koje razdvajaju ta dva sloja da budu u kontaktu samo kada se dogodi dodir. Strana na kojoj se stvara dodir mora biti fleksibilna, stoga se je načinjena od PET materijala ili neke slične fleksibilne plastike, dok druga elektroda mora biti načinjena od stakla ili tvrde plastike (slika 3.19). Na krajevima plohi se nalaze kontakti, no, kontakti moraju biti tako postavljeni da ne budu na istoj strani na obje plohe (ako su na jednoj postavljeni vertikalno, na drugoj moraju biti postavljeni horizontalno) Dodir se prevodi u jedinicu otpora, gdje se, ukoliko se znaju parametri panela (otpor obje plohe), može odrediti točno mjesto dodira. Pretvorba otpora se najčešće prevodi u napon, gdje se onda preko analogno-digitalno pretvornika prevodi u digitalnu riječ. Prednost ovoga panela je jednostavnost u radu, mala cijena. Mala potrošnja relativno dobra točnost, no mala je da je moguće detektirati samo jedan dodir (ukoliko se koristi 4 izvoda s panela), relativno loša optička svojstva (prvenstveno se ovdje misli na količinu svjetlosti koji propušta kod sebe koja iznosi oko 75% [23]) i temperaturno je ovisan).



Slika 3.19. Princip rada rezistivnog panela na dodir [24]

Kapacitivni panel osjetljiv na dodir je nešto kompleksniji. Postoje više tehnologija rada kapacitivnog panela osjetljivog na dodir, no princip je svugdje isti, a to je da se prati promjena naboja na površini elektroda panela, a time i kapacitet. Kada čovjek dodirne panel, poremeti se količina naboja, a time i kapacitet pojedine elektrode (slika 3.20). Signali svih elektroda se šalju u integrirani krug dediciran za kapacitivni panel osjetljiv na dodir koji obrađuje te signale i detektira dodir ili dodire. I u tome leži kompleksnost ove tehnologije, da bi se pravilno odredio dodir, potrebno je imati specijalni sklop koji generira i obrađuje signale s samoga panela. Bez toga vrlo je teško izvesti pretvorbu dodira u električni signal ili digitalnu riječ. Još jedna mana ove tehnologije je veća potrošnja energije, dok se čeka dodir i dostupnost takvih panela povezanih zajedno s sklopom za upravljanje s njime.



Slika 3.20 Princip rada kapacitivnog panela osjetljivog na dodir [25]

Ostale dvije tehnologije koje postoje su ili vrlo skupe ili nedostupne za manje projekte (za nabavu u manjim količinama s manjim dimenzijama panela). Stoga te dvije tehnologije niti ne ulaze u izbor. Zbog vrlo male potrošnje, vrlo jednostavnoga upravljanja, lake dostupnosti, u radu će se koristi rezistivni panel.

3.5. Senzori

Senzori su elektroničke komponente koje pretvaraju neku fizikalnu veličinu u električnu (napon, struja, frekvencija, otpor,…) ili digitalnu riječ. Senzori u ovome radi su vrlo bitni jer nam oni omogućavaju da kvalitativno odredimo parametre vezane za meteorologiju. U ovom slučaju ono što se želi pratiti su podaci o temperaturi zraka, relativnoj vlazi u zraku, atmosferski tlak zraka, količina CO2 (ili barem ekvivalentna količina CO2) u zraku, a poželjno bi bilo pratiti i neke od vanjskih parametara poput smjer vjetra, brzina vjetra (anemometar), količina oborina, količina UV zračenja i sunčevog zračenja. Neki od senzora su potpuno elektronički, bez mehaničkih komponenata (ili su mehaničke komponente već integrirane u sami senzor, poput senzora atmosferskog tlaka zraka), no neki su potpuno mehanički poput senzora vjetra i oborina (slika 3.21).

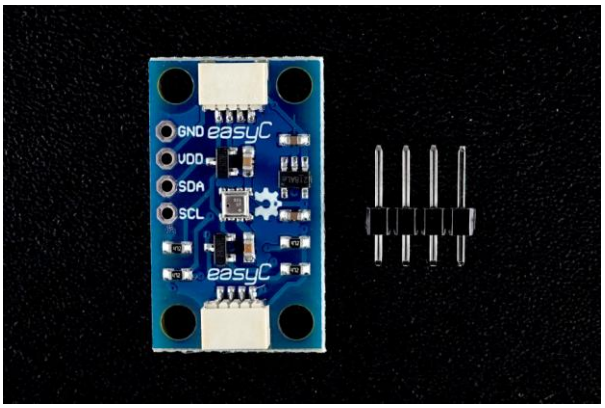
Slika 3.21. Anemometar i senzor smjera vjetra[26]

Kod izbora senzora treba obratiti na slijedeće karakteristike:

- Mala potrošnja senzora (svega nekoliko mA dok radi, svega nekoliko μA u stanju čekanja
- Male dimenzije senzora (samo za senzore temperature, vlage, tlaka, svjetla, UV-a)
- Dovoljno velika točnost i preciznost
- Radi napon (od 3 VDC do 5 VDC)
- Izlazna veličina (poželjno da bude digitalna riječ u obliku nekog popularnog komunikacijskog protokola, kao što je I2C, no može i napon u rasponu od 3 VDC do 5 VDC, struja u opsegu do 50 mA ili frekvencija do 1 kHz)

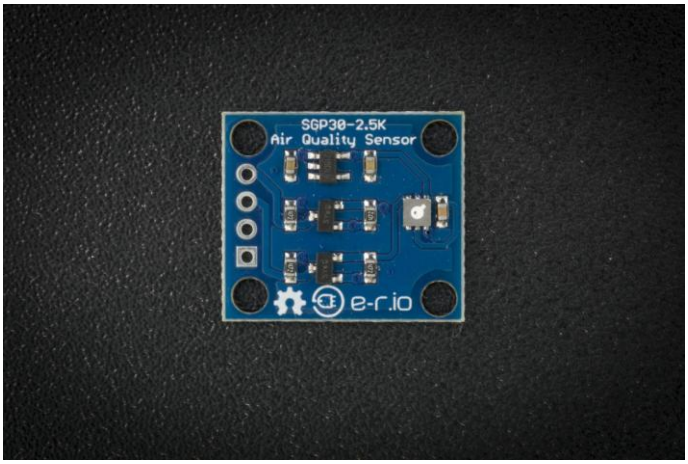
Zbog velikog izbora senzora temperature zraka, relativne vlažnosti zraka, atmosferskog tlaka zraka i svjetlosti koji su dostupni, senzore takvih karakteristika neće biti problem naći, no senzore za razinu sunčevog zračenja, UV zračenja, anemometar i senzor smjera vjetra je već teže naći. Senzore za vjetar je moguće napraviti pomoću raznih metoda, no točnost će im biti manja nego kod komercijalnih rješenja. Radi teške dobavljivosti, odlučeno je da će se senzori za vjetar napraviti putem procesa 3D printanja, te da će se njihova točnost naknadno utvrditi. Što se tiče senzora sunčevog zračenja, koristiti će se solarna ćelija s shunt otpornikom za mjerenje struje KS-a ćelije koja je proporcionalna količini sunčevoga zračenja. Ostali senzori su u obliku elektroničkih komponenti koji se mogu lako nabaviti i za koje proizvođači garantiraju zadanu točnost i preciznost. Tlak i temperatura unutrašnje jedinice će biti BME280 (slika 3.22) kojega je proizvela firma Bosch.

Senzor mjeri temperaturu zraka, relativnu vlažnost zraka i atmosferski tlak zraka, vrši pretvorbu mjernih veličina u digitalnu riječ koju šalje mikroupravljaču putem I2C komunikacijskog protokola.



Slika 3.22. BME280 Senzor temperature, vlage i tlaka zraka [27]

Senzor za CO2 (ili u ovome slučaju eCO2 – *equivalent* CO2) je SGP30-2.5K (slika 3.23), koji mjeri eCO2 i TVOC (engl. *Total Volatile Organic Compounds*), te kao i BME280 sadrži kompletno sklopovlje za pretvorbu i komunikaciju. Za vanjsku jedinicu, koristi se SHT21 za temperaturu zraka i relativnu vlažnost zraka, a BMP180 za atmosferski tlak zraka, a za UV i količinu svjetlosti Si1147. Svi navedeni senzori imaju sklopovlje za pretvorbu mjerne veličine i komunikaciju već integriranu u sebi. Također, imaju i automatsko prebacivanje iz režima niske potrošnje kod ne mjere u režim normalne potrošnje dok vrše pretvorbu čime se pojednostavljuje sklopovlje i sam programski kod.



Slika 3.23. SGP30-2.5K Senzor eCO2 i TVOC-a [28]

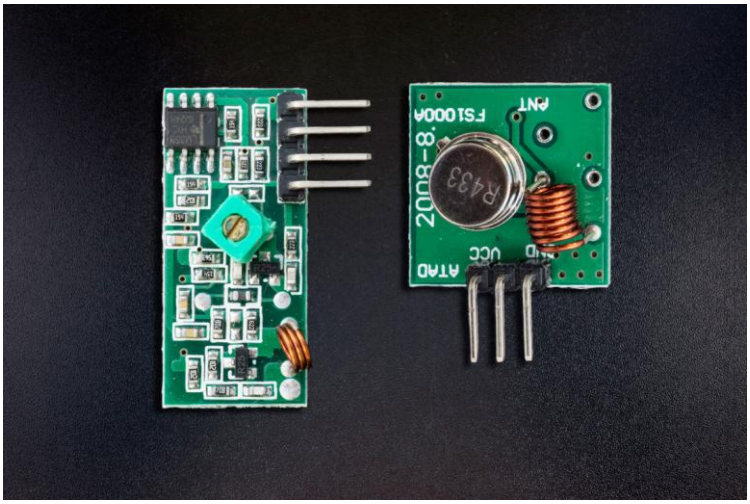
3.6. Bežična komunikacija

Da bi vanjska jedinica mogla slati podatke unutrašnjoj, gdje ih korisnik može vidjeti, spremiti i obraditi, potrebno koristiti nekakvu komunikaciju između ta dva uređaja. Jedno od najjednostavnijih rješenja ovoga problema bi bilo koristiti žičnu komunikaciju. Ovakva komunikacija je vrlo jednostavna za implementirati, izuzetno je robusna na smetnje i vrlo je jeftina, no problem može nastupiti ukoliko je vanjska jedinica dosta udaljena od unutrašnje ili ukoliko kabel estetski narušava izgled i funkcionalnost nekih stvari ili pak nije moguće na jednostavan način provesti kabel u unutrašnjost prostora gdje se nalazi unutrašnja jedinica. U tom slučaju jedina opcija je koristiti bežičnu komunikaciju, gdje se pritom misli na bežičnu komunikaciju koja koristi radiovalove za prijenos informacija. Bežična komunikacija nije bez kompromisa jer je značajno kompliciranija, zahtjeva više energije za rad, ima manju prijenosnu brzinu podataka i osjetljivija je na šum i smetnje. Zbog kompleksnosti bežične komunikacije, posebno RF dijela, koristit će se već neka gotova rješenja u obliku modula ili pak integriranoga kruga.

Zahtjevi koji se stavljaju na bežičnu vezu su:

- Frekvencijski opseg rada da je u IMS opsegu
- Domet od barem 20 m kroz dva zida
- Prijenos digitalne riječi
- Postoji zaštita od pogrešaka (pouzdanost komunikacije)
- Programska podrška
- Napon napajanja u opsegu od 3 VDC do 6 VDC
- Dvosmjerna komunikacija

Postoje par gotovih komercijalnih rješenja koja mogu riješiti problem komunikacije, a neki od njih su: WiFi, NRF24l01(+) radijski modul, LoRa komunikacija, 433 MHz radio modul. WiFi iako se na prvu čini kao jako dobro rješenje jer sami mikropupravljač već ima ugrađen WiFi modul u sebi, stoga je potrebno dodati WiFi na vanjsku jedinicu, stvara problem na prvu jer WiFi komunikacija nema veliki domet pogotovo kada se ubroji gušenje kroz zidove (15 m je tipičan domet kroz zidove), stoga bi komunikacija morala ići preko posrednika poput WiFi routera, gdje ponovno može nastati problem ukoliko je on nedostupan (kvar na routeru, nestanak električne energije,…). Drugi logičan odabir jest koristiti 433 MHz radijske module (slika 3.24) kakvi se koriste u sličim meteorološkom postajama za kućnu upotrebu.



Slika 3.24. 433 MHz radijski modul [29]

Ovisno o modulu, domet im može varirati, no većina modula uz veću antenu može stvorit domet od 20 m. Problem je što zahtijevaju dodatno sklopovlje koje će vršiti provjeru pogrešaka ili pak to mora biti odrađeno u samome programskome kodu. Treba nadodati da takvu moduli imaju samo jednosmjernu komunikaciju, te da bi se izvela dvostruka komunikacija, potrebno je koristiti dva različita modula sa svake strane komunikacije, gdje može postojati veliki problem interferencije pošto rade na istim frekvencijama. Radi loše pouzdanosti prijenosa podataka i kompleksnosti, ovi moduli ne zadovoljavanju kriterije. LoRa komunikacija je poznata po svojem velikom dometu, vrlo jednostavnom korištenju i vrlo maloj potrošnji radi korištenja modernih modulacijskih protokola. LoRa ima vrlo malu brzinu prijenosa podataka i ograničena je na slanje podataka vrlo rijetko i na vrlo male pakete (tekst i ASCII podaci nisu dozvoljeni) [30]. Radi limitacije čestosti slanja podataka i količine podataka, ova komunikacija ne zadovoljava kriterije korištenja u ovome radu, iako su ostali kriteriji valjani. Ukoliko bi se princip rada meteorološke stanice promijenio, ovo može biti jako dobar izbor. Zadnji odabir jest nRF24l01(+) komunikacijski modul. Karakteristike su slične LoRa komunikaciji, no nRF omogućava puno veći prijenos podataka (do max 2 Mbit/s), nema ograničenja na slanje podataka (jer koristi 2.4 GHz IMS frekvencijski opseg), no ima manji domet nego LoRa i nešto veću potrošnju. Također, komunikacija je dvosmjerna, gdje se čak i omogućava potvrda primitka podatka uz dodatan *payload* (gdje je *payload* ponovno podatak koji se žali poslati nazad, no ovoga puta ne postoji mogućnost potvrde primitka). Sami domet se može značajno povećati korištenjem modula s izlaznim pojačalom i LNA prijamnim pojačalom snage (slika 3.25) i spuštanjem brzine na najnižu moguću od 250 kBit/s što je i više nego dovoljno za ovu primjenu.

Iako i ova komunikacija ima mana kao što je nezadovoljavajući domet usporedno s LoRa komunikacijom, zbog svih ostalih parametara prikladna je za korištenje u ovome radu.

Slika 3.25. nRF24l01 RF modul s pojačalom i antenom [31]

3.7. Napajanje

Kao i svakom elektroničkom uređaju, potrebno je napajanje. Izbor napajanja u ovome radu je vrlo bitan, pošto uređaj radi većinu vremena u režimu niske potrošnje energije, a kada radi, može trošiti velike količine energije u vrlo kratkom vremenskome intervalu (pogotovo tijekom spajanja na WiFi mrežu ili prilikom osvježavanja *e-paper* zaslona. Povrh toga, poželjno je da se ovaj uređaj može napajati s sunčevom energijom putem solarnog panela. Pošto ne postoji situacija da solarni panel može proizvoditi električnu energiju konstantno bez prekida, potrebno je višak energije skladištiti, da bi se za vrijeme manjka sunčeve energije uređaj mogao koristiti. To je svrha baterije. No, postoje različite baterijske tehnologije, stoga je potrebno pažljivo odabrati odgovarajuću baterijsku tehnologiju za ovaj rad. Da bi se izrada rada pojednostavila, vanjska i unutrašnja jedinica će imati iste baterije i gotovo identične zahtjeve na istu.

Baterija mora imati slijedeće karakteristike:

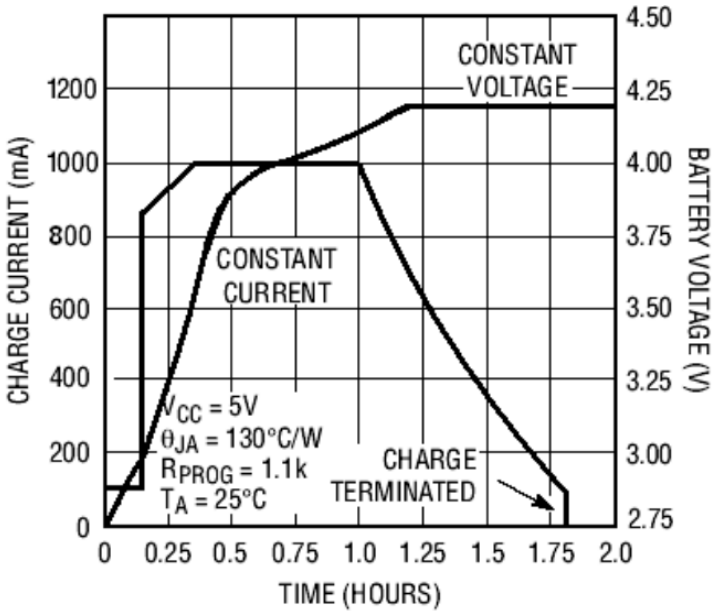
- Radni napon u rasponu od 3 VDC do 6 VDC
- Kapacitet reda 1 Ah i više
- Može proizvoditi barem 1 A struje (potrebno kod osvježavanja zaslona, WiFi komunikacije i RF komunikacije)

- Mogućnost punjenja (sa i bez dodatnog sklopovlja)
- Jako malu struju samopražnjenja
- Veliki temperaturni raspon (-10 °C do 40 °C)
- Vrlo dostupna cijenom i dobavljivošću

Jedine baterijske tehnologije koje dobrim dijelom imaju takve karakteristike su NiMH baterije i litijeve baterije (slika 3.26). Problem kod NiMH baterija je taj što imaju relativno visok unutrašnji otpor [32] u odnosu na litijeve baterije [33], jer se pri većim iznosima struje može dogoditi značajan pad napona i nestabilnost uređaja u radu kao i nepotrebna disipacija energije (gubitak energije). Također, NiMH baterije imaju i veliku struju samo pražnjenja, što znači da baterija gubi energiju iako na nju nije ništa priključeno. Problem je uz to što ovakve baterije pate od tzv. *memory effect-a*, gdje ukoliko se baterija svaki puta ne isprazni do kraja i ponovno napuni, gubi nazivni kapacitet. S druge strane, NiMH mogu ispravno raditi na puno širem temperaturnome opsegu (od -20 °C do 45 °C) za razliku od litijevih koji imaju opseg od 0 °C do 35 °C.

Slika 3.26. 18650 litijeva baterija [34]

Ono što može predstaviti dodatan problem jest taj što litijeve baterije traže specijalan način punjenja s tzv. CC/CV metodom (slika 3.27) (engl. *CC - Constant Current, CV – Constant Voltage*) inače može doći do trajnog uništenja baterije ili pak materijalne štete ukoliko dođe do zapaljenja baterije. Srećom, postoje gotovi integrirani punjači litijevih baterija koji omogućavaju punjenje baterije na pravilan način poput TP4056 integriranoga kruga.



Slika 3.27. CC/CV metoda punjenja [35]

Bez obzira na uži temperaturni opseg nego NiMH baterije, odabrano je da će se u ovome radu koristiti litijeve baterije zbog radnoga napona od 3 V do 4.2 V, velike izlazne struje do 3 A kod nekih modela baterija, velikog kapaciteta (jedna ćelija može imati i do 2.5 Ah) i jako male struje samo pražnjenja.

4. IZRADA METEOROLOŠKE POSTAJE (STANICE)

4.1. *Reverse engineering* principa rada *e-paper* zaslona

Pošto niti jedan od zaslona navedenih u odjeljku 3.1. ne dogovara po svojim karakteristikama ovome radu, odlučeno je da će se koristiti sami zaslon (panel) koji će se kontrolirati preko mikroupravljača. Takvi zasloni se obično koriste u e-book čitačima (kao npr. Amazon Kindle) kao zamjenski zasloni u slučaju kvara na originalnome. Takvi zasloni su rezolucije 800x600 piksela, omogućavaju nam prikaz slike u nijansama sive (uz pomoć adekvatne programske i sklopovske podrške), imaju vrlo malu potrošnju, relativno veliku dimenziju i relativno jednostavno sklopovlje (potrebno je napraviti napajanje koje će generirati potrebne napone napajanja za zaslon). I ovdje postoji izbor zaslona, no najviše se ovdje bazira na to za koji zaslon ima najviše dostupnih podataka (najbitniji je *datasheet* zaslona). Zasloni za koje je moguće naći *datasheet* su

- ED060SC4
- ED060SC4 H1
- ED060SC4 H2
- ED060SC7 (slika 3.8)
- ED097OC1
- ED060XH7

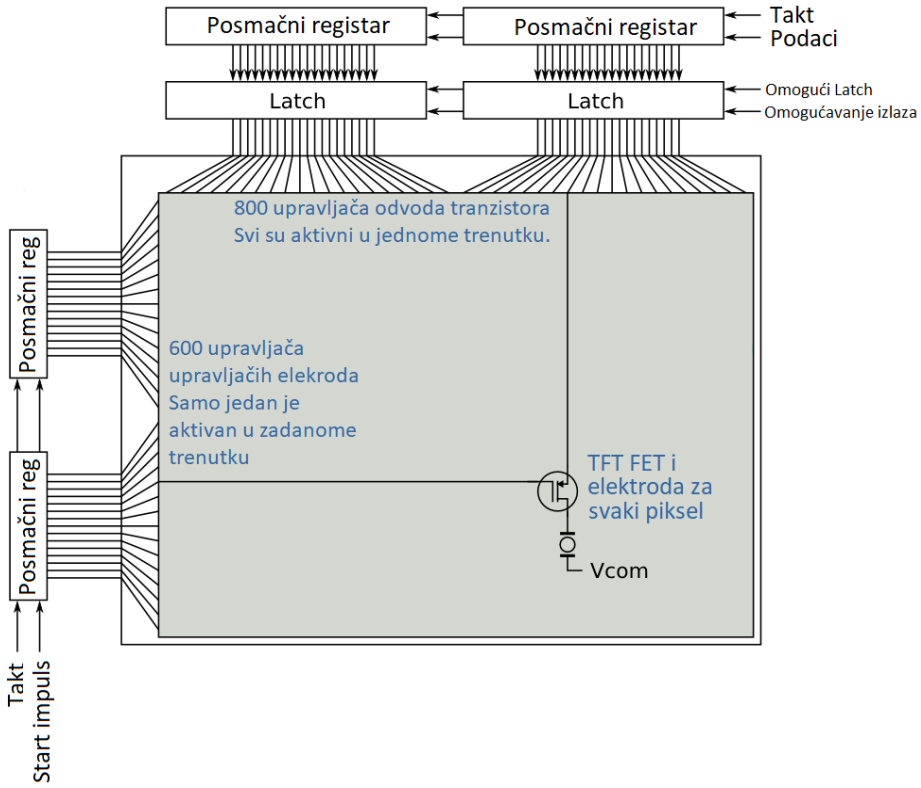
Prva tri su isti model, pa imaju i isti *datasheet*, no razlika je u kontrastu i kvaliteti slike koju proizvode. Bez H oznake ima vrlo dobru sliku, no to je prva generacija takvih zaslona i vrlo ih je teško dobaviti. H1 zaslon je vrlo dobavljiv, no kontrast je relativno loš, crna je više sivkasta, bijela je tamnija nego što treba biti. H2 zaslon ima odličan kontrast, no, kao i prva generacija tog modela zaslona (bez H oznake), teško je dobavljiv.

ED060SC7 je zaslon novije generacije, kod kojeg je moguće dobaviti *datasheet*, pristupačan je cijenom i ima izvrstan kontrast (crna je vrlo tamna), no problem je u cijeni konektora koja može dostići i 3 € za jedan. No, usprkos tomu, ovaj model zaslona je odabran da se koristi u meteorološkoj postaji.

Slika 4.1. ED060SC7 zaslon

Zaslon se kontrolira pomoću paralelne komunikacije. Zaslon u sebi nema nikakav mikropravljač za kontrolu, već ima niz posmačnih registra (engl. *Shift Registers*), upravljačkih sklopova za upravljačku elektrodu i uvod tranzistora za svaki piksel. Stoga, da bi ga se upravljalo, potrebno je znati sekvencu slanja određenih impulsa. Ti podaci nisu dostupni za javnost, stoga je potrebno izvesti postupak zvan *reverse engineering*. To je postupak kada se pomoću raznih alata (osciloskop, multimetar, logički analizator, itd), logike i vlastitog iskustva pokušava saznati kako neki uređaj ili dio uređaja radi metodom pokušaja i promašaja, te samim istraživanjem. Ovakvi poduhvati su već načinjeni od strane raznih ljudi za ovu vrstu zaslona, stoga je potrebno istražiti rezultate mjerenja slanja impulsa i podataka prema zaslonu.

Kao što je već prethodno rečeno, zasloni imaju niz posmačnih registara za upis podataka u sami zaslon (panel). Posmačni registri (slika 3.9, gornji posmačni registri) su postavljeni tako da omogućavaju odabir piksela samo za jedan red i to onaj koji je trenutno odabran od strane posmačnog registra (slika 4.2, posmačni registri s lijeve strane) koji omogućava da se samo jedan izlaz postavi u stanje logičke jedinice, a time i odabere da se pikseli postave u željeno stanje.



Slika 4.2. Unutrašnja struktura e-paper zaslona [36]

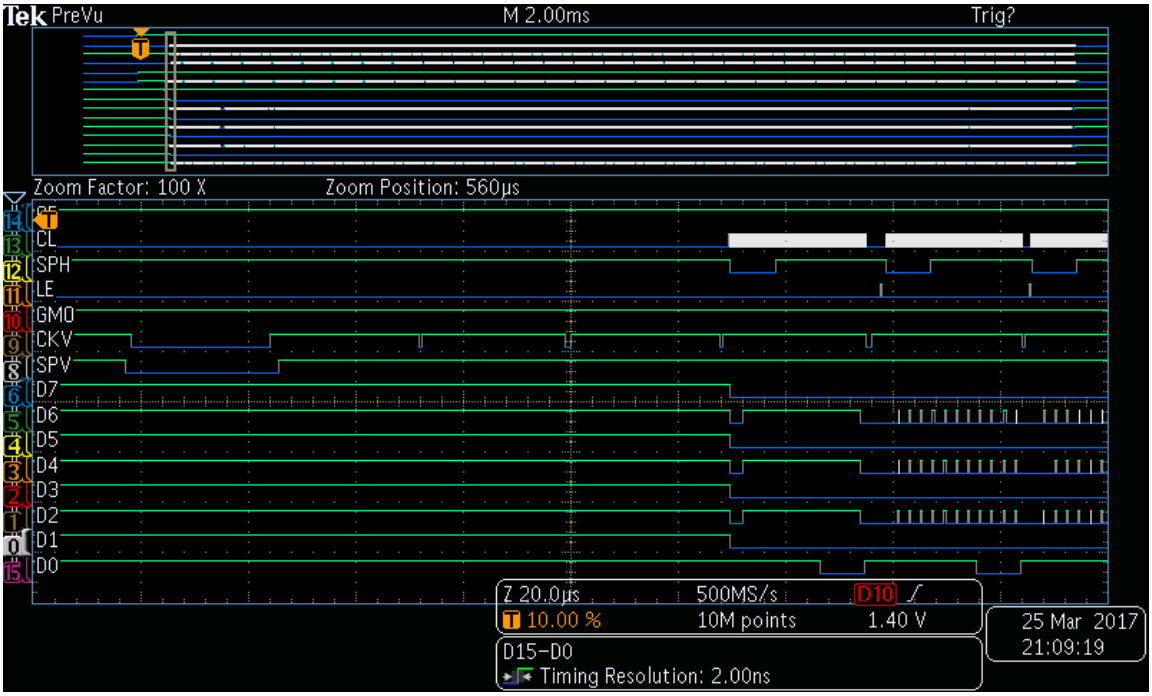
Gornji posmačni registar je napravljen da primi 1600 bita, gdje dva bita reprezentiraju jedan piksel. To znači da jedan bajt (koji se šalje u gornji posmačni registar, jer sam posmačni registar radi na principu paralelne komunikacije), kontrolira stanja 4 piksela. U tablici 4.1 moguće je vidjeti tablicu istine za ta dva bita po pikselu.

Tablica 4.1. Tablica istine za jedan piksel u bajtu podataka

Stanje bitova	Rezultat
00	Pražnjenje panela
01	Crni piksel
10	Bijeli piksel
11	Stanje visoke impedancije izlaza (HiZ state)

Kompletan popis izvoda zaslona, moguće je naći u prilogu, prilog 2 [37]. Naime, zaslon ima nekoliko bitnih izvoda za kontrolu: SPV (*Star Pulse Vertical*), GMODE, SPH (*Start Pulse Horizontal*), OE (*Output Enable*), LE (*Latch Enable*), CKV (*Clock Vertical*), CL (*Clock*), Data. SPV

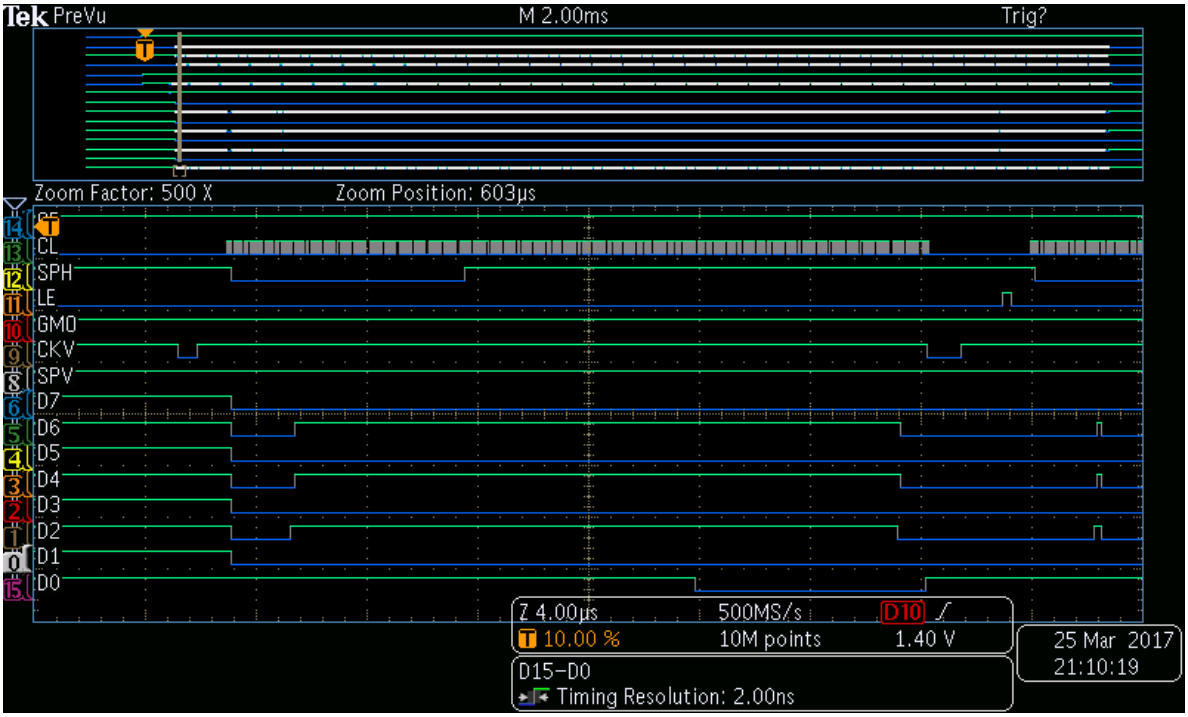
i SPH moraju biti u stanju logičke jedinice, zatim se OE postavi u stanje logičke jedinice, nakon čega GMODE i CKV također se postave u stanje logičke jedinice. Nakon toga je potrebno napraviti pauzu od barem 500 µS, nakon čega se SPV i CKV postavljaju u stanje logičke nule na približno 30 µS, nakon čega se ponovno postavljaju u logičku jedinicu. Ova sekvenca restartira zaslon da bude spreman za primanje podataka. Na nekim zaslonima je potrebno preskočiti tri reda da bi se odabrao prvi red, stoga se CKV okine tri puta zaredom. Sve prethodno objašnjeno, vidljivo je na slici 4.3.



Slika 4.3. Sekvenca inicijaliziranja zaslona [38]

Upis podataka na zaslon vrši se red po red, stoga, nakon što je zaslon restartiran, počinje proces odabira koji će pikseli biti postavljeni na crnu boju, a koji na bijelu. Proces se vidi na slici 3.11, a u nastavku će biti detaljnije objašnjen. Naime, da bi se započeo unos u posmačni registar za uvode tranzistora (engl. *Source driver*), potrebno je SPH postaviti u stanje logičke nule. Nakon toga, na podatkovnu sabirnicu zaslona šalju se podaci (svaki bajt reprezentira 4 piksela), nakon čega se CL postavi u stanje logičke jedinice i odmah nakon toga u stanje logičke nule (pošalje se impuls, koji ne smije biti brži od 20 MHz). To se izvede 200 puta, da bi se upisao svaki piksel. Sada su pikseli u posmačnom registru i potrebno ih je poslati na zaslon. Prvo je potrebno poslati impuls na CKV koji ne smije biti kraći od 1 µS ni za vrijeme logičke jedinice, niti za vrijeme logičke nule. Nakon toga se pošalje impuls na LE (*Latch*) koji ponovno ne smije biti brži od 20 MHz. To je potrebno ponavljati, dok se svi redovi ne upišu. Nakon toga, postupak slanja okvira slike (engl.

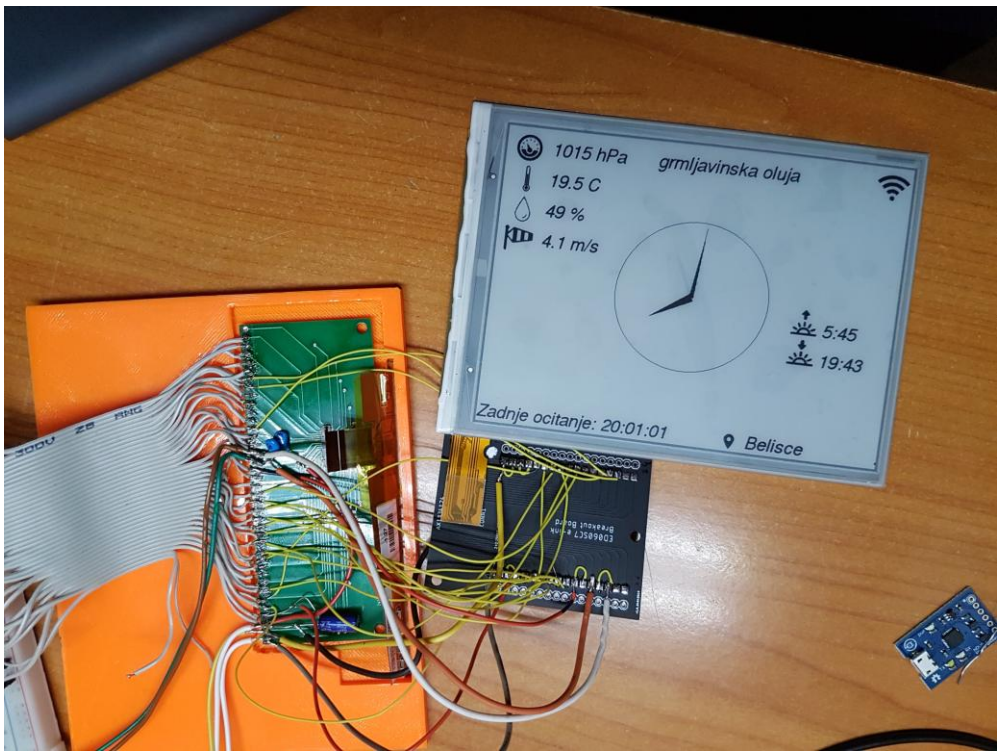
Frame) se ponavlja određeni broj puta da bi se prethodna slika očistila i da bi se dobila zadovoljavajuća kvaliteta novonastale slike. Zatim, zapis se zaključava tako što se CKV, LE i CL postave u logičku nulu, zatim se, nakon nešto manje od 2 mS GMODE postavi u logičku nulu, kao i OE, te je poželjno da svi izvodi prema zaslonu budu u stanje visoke impedancije jer će se nakon toga, ugasiti napajanje zaslonu.



Slika 4.4. Sekvenca slanja podataka (piksela) na zaslon [38]

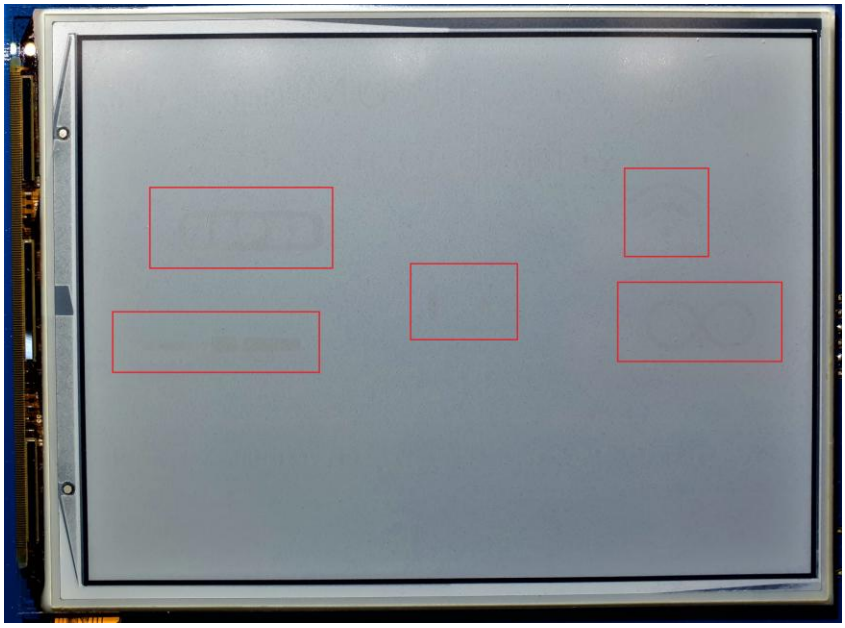
Dovođenje i gašenje napajanja također treba uraditi točnom sekvencom, inače može doći do lošije kvalitete ispisa na zaslonu, kraćeg trajanja zadržavanja slike na zaslonu (slika s vremenom izblijedi), te u konačnici do uništenja elektronike *e-paper* zaslona. Kod pokretanja, smatrajući da je zaslon uvijek spojen s masom, prvo se dovodi digitalno napajanje od 3.3 VDC, te pričeka. Nakon toga potrebno je uključiti negativno napajanje, pričekati 1 mS, te nakon toga, uključiti i pozitivno napajanje. Kada se ta dva napajanja stabiliziraju (minimum 100 µS), može se dovesti VCOM napon, pričekati najmanje 100 µS. Sekvenca gašenja je slična sekvenci pokretanja, samo što ide obrnuto. Vremenski dijagrami pokteranja i gašenja mogu se pronaći u prilogu, prilog 3 [37] i prilog 4 [37]. Nakon istraživanja, potrebno je bilo stvoriti prvi prototip. Prototip je sastavljen od ESP32-WROOM mikroupravljačkoga modula, modula za napajanje *e-paper* panela koje proizvodi +22 VDC, +15 VDC, -20 VDC, -15 VDC i VCOM napon u rasponu od 0 VDC do -5 VDC. Da bi se panel mogao

spojiti na mikroupravljač, potrebno je napraviti adapter koji će nam omogućiti spajanje, što se izvelo na način da se napravila tiskana pločica s konektorom za *e-paper* panel čiji su kontakti bili izvučeni na rub tiskane pločice. Cijela postava (bez mikroupravljača) je vidljiva na slici 4.5.



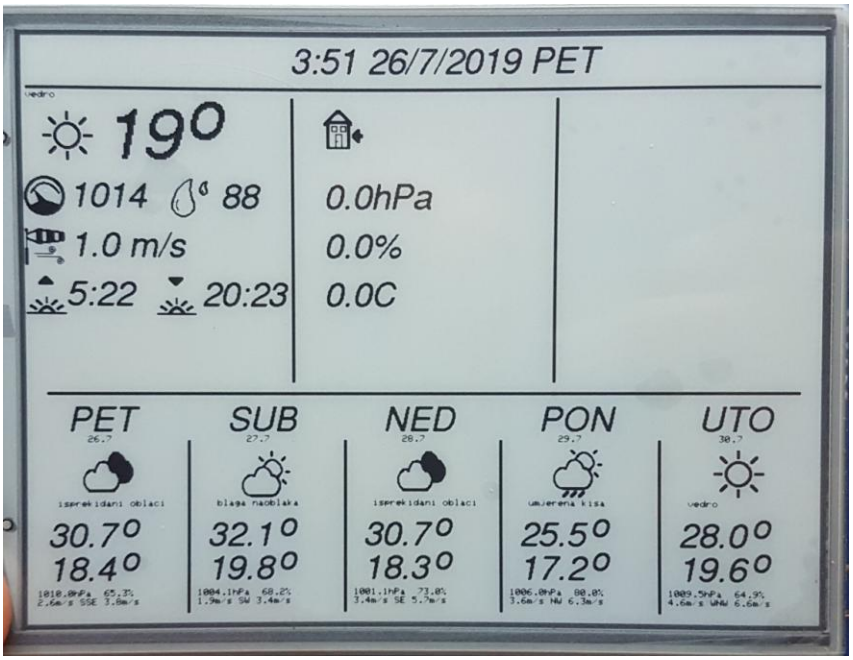
Slika 4.5. Prototip za pokretanje *e-paper* panela

Rezultat je da se panel uspio pokrenuti i ispisati sliku, ali slika nije zadovoljavajuće kvalitete, niti je brzina osvježavanja dovoljno velika. Na panelu su vidljivi tragovi prethodne slike (tzv. engl. *Ghosting*) što umanjuje kvalitetu prikaza, no ujedno nije ni dobro za panel, jer ukoliko se slika na nekome mjestu ne mijenja nakon niza osvježavanja, na tome mjestu se može dogoditi tzv. engl. *Burn-In* gdje se slika „upeče“ u sami panel i vrlo ju je teško ukloniti čak i nakon više čišćenja. Ukoliko se taj problem ne adresira, može doći to trajnog oštećenja *e-paper* panela što je vidljivo na slici 4.6.

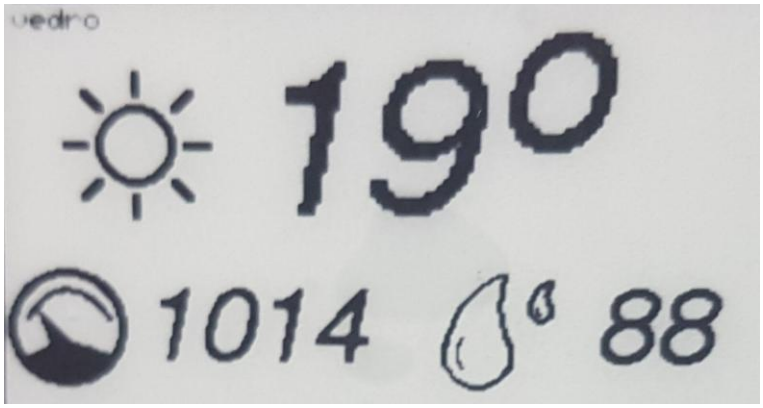


Slika 4.6. Trajno oštećenje panela radi lošeg brisanja

Nakon detaljnije analize je otkriveno da napajanje za *e-paper* ne može proizvoditi dovoljnu količinu struje što rezultira u padu napona i lošoj kvaliteti slike. Rješenje je koristiti jače napajanje ili koristiti već komercijalno rješenje koje je specifično namijenjeno kao napajanje za *e-paper* panele. Odabrano je drugo rješenje u obliku TPS65186 integriranoga kruga, jer je jednostavnije i pouzdanije rješenje. Brzina osvježavanja je povećana tako da se povećao glavni takt koji ide na *e-paper* panel na 6.67 MHz što je ujedno i limit za dotični mikroupravljač, no da bi se to postiglo moralo se koristiti kašnjenje na liniji takta *e-papera* pomoću kondenzatora, inače podaci i takt dođu u isto vrijeme, te je rezultat toga da panel ne registrira nikakve podatke. Sve prethodno navedene stvari rezultirale su u nešto boljoj kvaliteti slike, no sada se pojavio problem da slika više nije dovoljno oštra (slike 4.7 i 4.8).



Slika 4.7. Više nema zaostale prethodne slike, no ujedno je izgubljena i oštrina slike

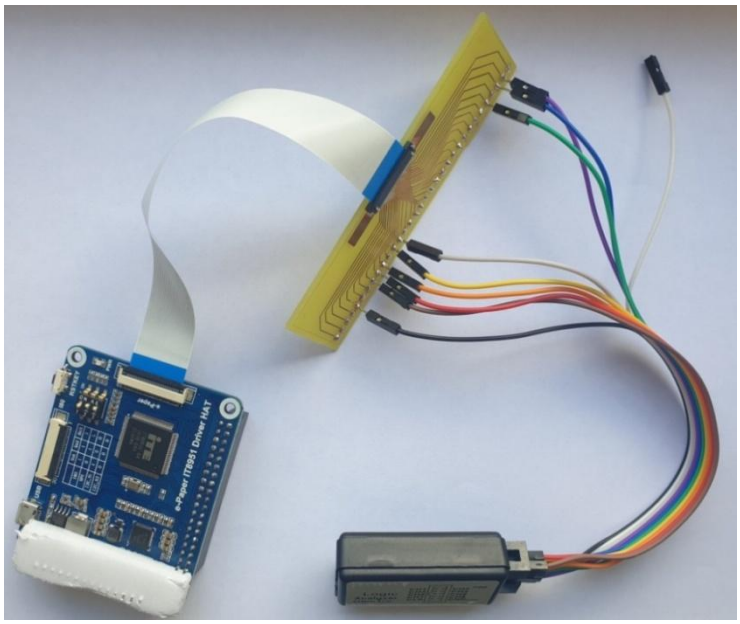


Slika 4.8. Uvećani detalj oštrine slike (gubitak oštrine je definitivno vidljiv na riječi „vedro“)

Na slici 4.8 je izrazito vidljiv gubitak oštrine na slici „vedro“, gdje se više ne vide pojedinačni pikseli oštro, već su rubovi samih piksela zamućeni, a poneki pikseli su čak i slabo vidljivi. Isti efekt se također vidi na simbolu pored broja 1014. Također, na nekim panelima istog modela pojavila se situacija da slika bude pomaknuta (slika 4.9) za 4 piksela (odnosno jedan bajt), iako je korišten isti model panela.

Slika 4.9. Slika pomaknuta za 4 piksela

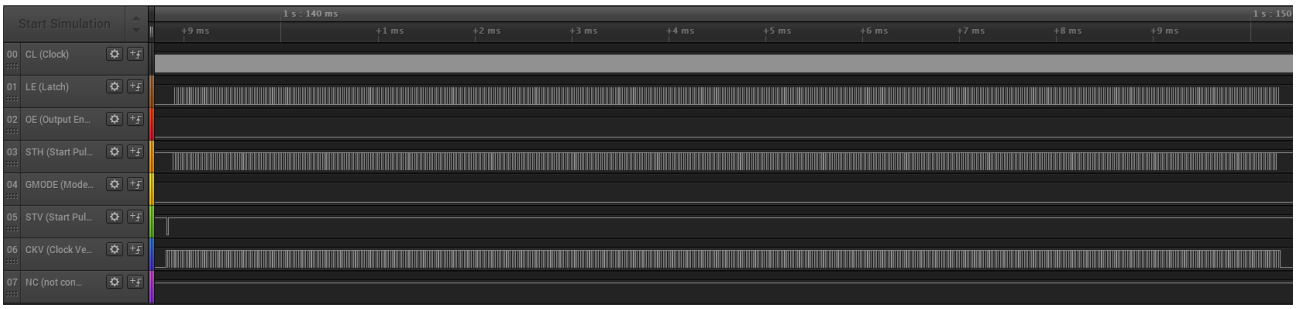
Kako korištenje prve navedene sekvence impulsa koja je pronađena na Internetu nije stvarala zadovoljavajuću kvalitetu slike i neke dodatne probleme, potrebno je bilo uzeti komercijalan sustav za upravljanje *e-paper* i provjeriti izgled i redoslijed impulsa na njemu. Za to je potrebno imati logički analizator i tiskanu pločicu s konektorom za *e-paper flat cable*. Korišteni logički analizator ima 8 digitalnih kanala, te brzinu uzorkovanja od 24 MS/s, što je dovoljno za dohvaćanje signala (iako se ponekad izgubi signal glavnoga takta *e-paper* panela, jer je jednak polovici frekvencije uzorkovanja). Uz njega je korišten i Salea Logic programski paket za prikaz snimljenih podataka. Tiskana pločica s konektorom predstavlja adapter za spajanje logičkog analizatora na komercijalan sustav za upravljanje *e-paper* zaslonom (slika 4.10). Komercijalan sustav za upravljanje *e-paper* zaslonom je bazirano na IT8951 integriranome krugu za slanje sekvence impulsa i komunikaciju s računalom, a TPS65185 integrirani krug je zadužen za napajanje *e-paper* zaslona.



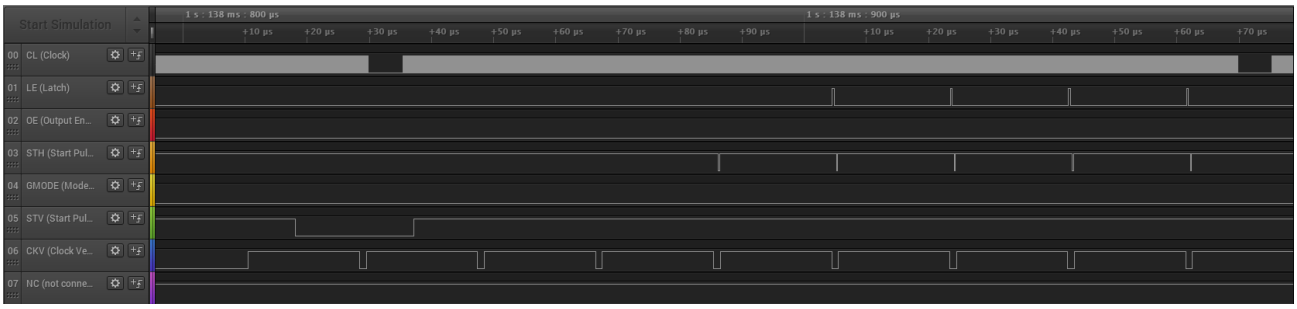
Slika 4.10. IT8951 upravljač spojen na logički analizator signala

Nakon snimanja signala i analize signala, uvidjela se da postoji razlika u sekvenci koja se dobiva na izlazu iz mikroupravljača, sekvenci koja je pronađena na internetu i sekvenca koja je

snimljena s IT8951 integriranoga kruga. Ispravna snimljena sekvenca impulsa je vidljiva na narednim slikama (slike od 4.11 do 4.17).

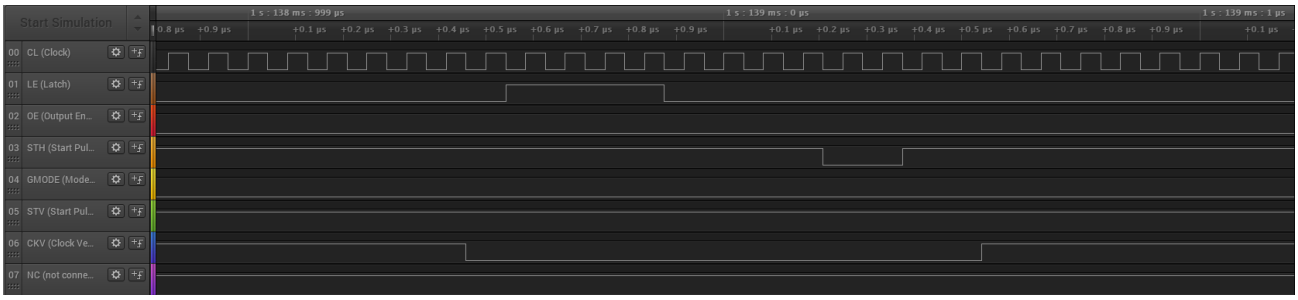


Slika 4.11. Ispravna sekvenca impulsa – čišćenje panela, upis jednog okvira



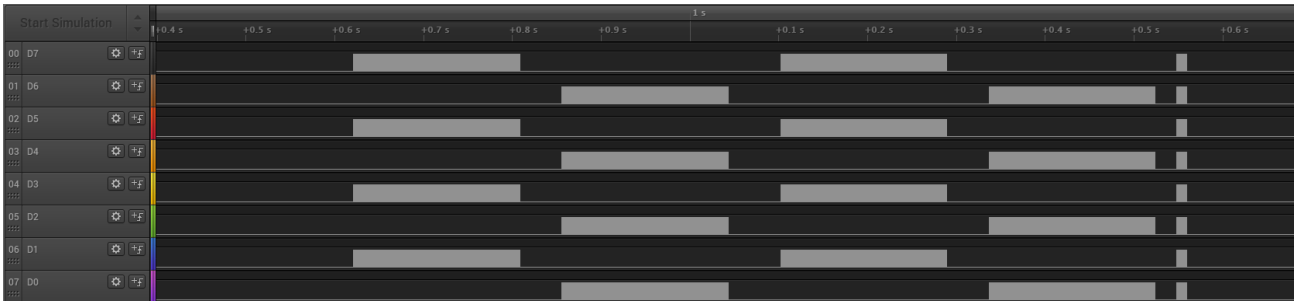
Slika 4.12. Ispravna sekvenca impulsa – čišćenje panela, početak upisa novog okvira

Slika 4.13. Ispravna sekvenca impulsa – čišćenje panela, upis jednog reda i početak upisa novog reda



Slika 4.14. Ispravna sekvenca impulsa – čišćenje panela, uvećani detalj upisa jednog reda

Slika 4.15. Ispravna sekvenca impulsa – čišćenje panela, završetak upisa zadnjeg reda i početak upisa novog okvira

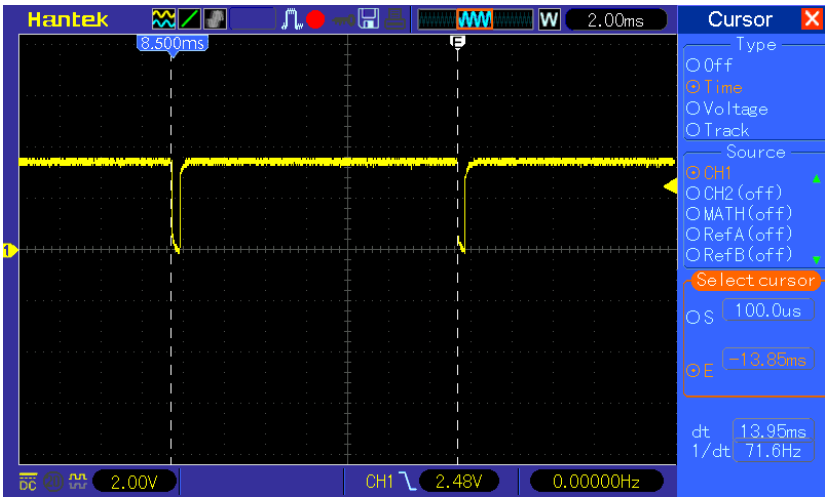


Slika 4.16. Ispravna sekvenca impulsa – čišćenje panela, podatkovna sabirnica

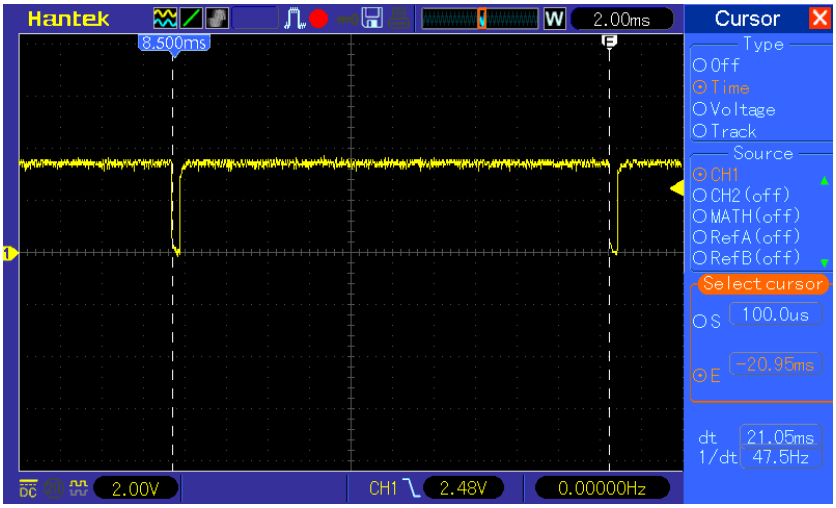
Slika 4.17. Ispravna sekvenca impulsa – čišćenje panela, podatkovna sabirnica, broj okvira jedne boje tijekom čišćenja

Kontrola IT8951 upravljačem se vršila putem računala preko USB veze. Na računalu je postojala programska podrška u obliku programa zvanog E Ink TCon Demo AP koji je omogućavao slanje jednostavnih komandi prema IT8951 kao i slanje slika u .bmp i .jpg formatu na IT8951 upravljač koji bi tu sliku prikazao na *e-paper* panelu. Iz snimljenih sekvenci sastavljen je programski kod koji pokušava replicirati pravu sekvencu, no zbog tehničke limitacije mikroupravljača, kao što je brzina i broj izvoda, nije bilo moguće napraviti identičnu sekvencu. Prvi problem s brzinom je bio što CL linija daje impulse brzine nešto više 12 MHz, dok je limit na mikrouravljaču bio 6.67 MHz s GPIO. Moguće je koristiti paralelni I2S, no zbog loše dokumentacije to bi uzelo previše vremena za razvoj.

Tu postoji i problem s brzinom RAM memorije. Da bi *e-paper* radio ispravno, potrebno je skladištiti kompletnu trenutnu sliku s panela u RAM memoriji, stoga je potrebna vrlo velika memorija. Za taj slučaj se koristi vanjska PSRAM memorija ugrađena u ESP2-WROVER modul, no njezina stvarna brzina od 12.3 MB/s je premala da bi se izvelo 85 okvira u jednoj sekundi (što je definirano po tehničkoj dokumentaciji korištenog *e-paper* panela), stoga se uz sve optimizacije koda poput korištenja LUT tablica i korištenje što više logičkih operacija umjesto matematičkih nije postigao veći broj okvira od 71 okvir u sekundi (slika 4.18) tijekom čišćenja panela, te 47 okvira u sekundi tijekom upisa crno-bijele slike (slika 4.19).

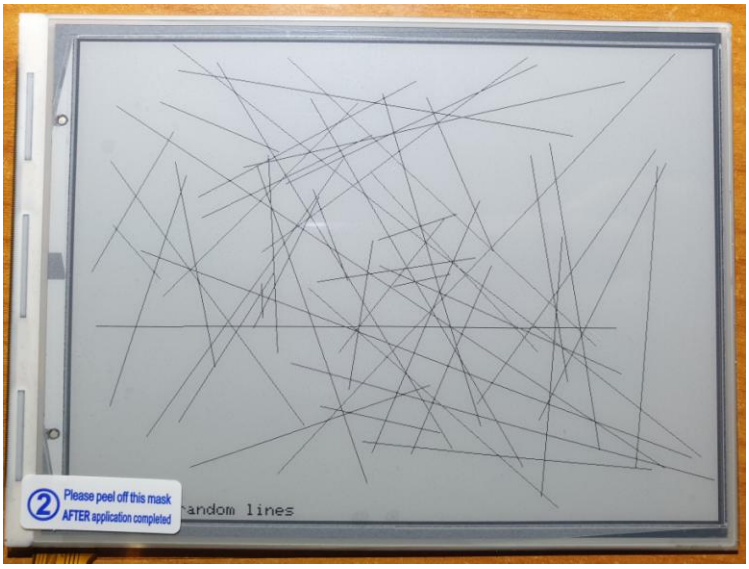


Slika 4.18. Trajanje upisa jednog okvira u *e-paper* panel tijekom čišćenja (SPV linija)

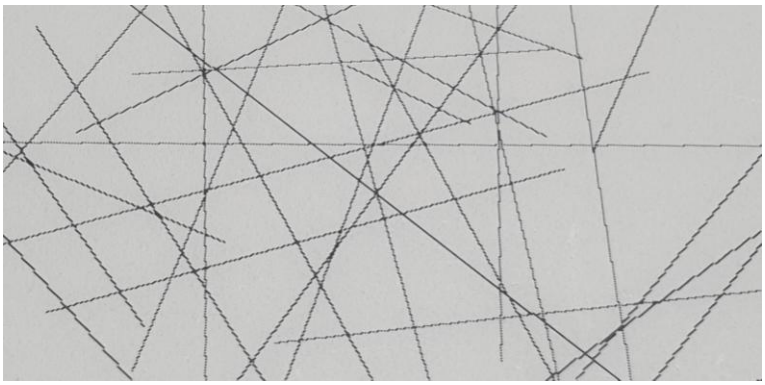


Slika 4.19. Trajanje upisa jednog okvira u *e-paper* panel tijekom slanja crno-bijele slike (SPV linija)

Nakon što se napravio novi programski kod za upravljanje s panelom, kvaliteta slike, kao i brzina osvježavanja se poboljšala, više nije bilo problema zaostale slike i smanjene oštrine, što se vidi na slikama 4.20 i 4.21, te više nije bilo problema da pojedini paneli odbijaju raditi ili imaju pomak slike.



Slika 4.20. Slika na panelu nakon prepravke sekvence i programskog koda



Slika 4.21. na panelu nakon prepravke sekvence i programskog koda (oštrina slike i pojedinih piksela)

Za svaku i najmanju promjenu slike na panelu potrebno je izvršiti proces čišćenja panela koji traje oko jednu sekundu. Da bi se proces ubrzao, može se koristiti tzv. djelomično osvježavanje (engl. *Partial Update*). Ono radi na principu da se u memoriji stvore dva memorijska spremnika (engl. *Frame Buffer*), jedan koji pamti ono što je trenutno na panelu i jedan u koji se sprema što se novo želi ispisati na panelu. Prilikom osvježavanja, šalje se samo razlika na panel za svaki piksel. Na ovaj

način se može drastično ubrzati ispis, po cijenu kvalitete slike, jer se sa svakim novim osvježavanjem gubi kvaliteta slike (slika 4.22) i nakon određenog broja ovakvih osvježavanja, koji ovisi o faktorima poput temperature, kvalitete samog panela, sadržaja na zaslonu, mora se sve očistiti s zaslona i napraviti potpuno osvježavanje panela s čišćenjem.

Slika 4.22. Gubitak kvalitete slike korištenjem djelomičnog osvježavanja

Na primjeru slike 4.22 vidljivo je da je tekst koji je bio statičan blijed i da se vide koji dijelovi brojeva su djelomično osvježavani, a koji nisu. Ovakav efekt je vidljiv nakon 15 uzastopnih djelomičnih osvježavanja, gdje se napajanje panela nije gasilo između, no ukoliko bi se napajanje svaki puta palilo i gasilo, negativan efekt ovakvoga načina osvježavanja bi bio puno više vidljiv. No treba napomenuti da se taj negativan efekt može razlikovati od panela do panela, te da je jako temperaturno ovisan, no isto tako da se putem prilagodbe programskoga koda može ublažiti (no ne i potpuno ukloniti). Još jedna od mogućnosti *e-papera* jest stvaranje slika u nijansi sive boje. To se radi na način da se jedan piksel, nakon procesa čišćenja panela, ne osvježava samo jednom bojom (npr. crnom, kombinacija bitova 01) kroz sve okvire, već da se u različitim okvirima mijenjaju kombinacije crnih i bijelih piksela da bi se na kraju odgovarajuća nijansa sive boje. Ovo je moguće jer 01 kombinacija bitova ustvari ne označava stvaranje crnog pikesla, već zatamnjenje trenutnoga. Takva kombinacija slanja određenih stanja piksela u trenutnome okviru se zove engl. *Waveform* i on se može razlikovati od panela do panela. Taj podatak je obično nedostupan za javnost, stoga se mora ručno tražiti kombinacija za svaku pojedinačnu nijansu sive boje. Ukoliko se podaci i okviri šalju

brzo (barem 85 okvira u sekundi), može se dobiti slika s čak 16 nijansi sive boje. Radi tehničke limitacije mkroupravljača, moguće je dobiti samo 8 nijansi sive (slika 4.23). Najveći razlog tomu je sama brzina RAM memorije, zbog koje je moguće proizvesti samo 30 okvira u sekundi.

Slika 4.23. Slika u nijansama sive boje koristeći ESP32 mikroupravljač (8 razina sive boje) (slika preuzeta s [39])

Ono što treba napomenuti da je i sama ovakva temperaturno ovisna, kao što je i jako ovisna o panelu, stoga ukoliko kvaliteta slike nije zadovoljavajuća, *waveform* je potrebno prilagoditi.

4.2. Izrada unutrašnje jedinice

Unutrašnja jedinica služi da bi se korisniku ispisali željeni podaci poput vremenske prognoze, kao i izmjereni podaci unutar prostora, te vani u okolišu. Stoga, sama unutrašnja jedinica se mora sastojati od zaslona (*e-paper* panela), senzora, radijskog modula za komunikaciju s vanjskom jedinicom, baterije i solarnog panela. Radi jednostavnosti će se koristiti gotova pločica koja na sebi sadrži ESP32 mikroupravljač, TPS65186 napajanje, integrirani krug za povećanje broja digitalnih ulaza i izlaza i sami panel. Pločica je vidljiva na slici 4.24. Ostale komponente poput upravljača panela osjetljivog na dodir, senzora, RTC integriranog kruga će se nalaziti na zasebnoj tiskanoj pločici koja će se spajati s pločicom na kojoj se nalazi zaslon. Programska podrška za

programiranje mikroupravljača je u obliku integriranog razvojnog okruženja programskog paketa Arduino IDE uz dodatak programskog dodatka (engl. *Core*) za ESP32 seriju mikroupravljača. Razlog korištenja Arduino IDE jest velika jednostavnost korištenja, te veliki broj programskih biblioteka za različite senzore, kao i za WiFi podršku. Korišteni senzori će biti spojeni putem I2C protokola, a napajanje će biti u obliku litijeve baterije koju će puniti solarni panel. Solarni panel je nazivne snage 2 W i napona otvorenog kruga od 6 V.

4.2.1. Izrada sklopovlja za panel osjetljiv na dodir

Da bi se korisniku omogućilo upravljanje unutrašnjom jedinicom, potrebno je koristiti panel osjetljiv na dodir. Koristi se rezistivni panel osjetljiv na dodir, što znači da trenutna pozicija dodira se može odrediti mjerenjem otpora. Pošto mikroupravljač nema mogućnost mjerenja otpora izravno, potrebno je izvršiti pretvorbu otpora u neku drugu mjernu veličinu. Mikroupravljač ima analogno digitalni pretvornik koji pretvara analogan napon u 12 bitnu digitalnu riječ. Ukoliko bi se vršila pretvorba otpora u napon, onda bi se moglo odrediti točna pozicija dodira pomoću izmjerenoga napona. Kako *touch* panel ima četiri izvoda (dva izvoda za svaku transparentnu plohu), jedna od mogućnosti jest da se jedna od ploha spoji između napajanja, a zatim jedna od preostale dvije da mjeri napon. U tome slučaju panel osjetljiv na dodri se ponaša kao potenciometar gdje napon ovisi o mjestu dodira jedne osi. Zatim se napon sada postavi na druge dvije elektrode, a sa trećom se mjeri napon i točka dodira za drugu os. Još uvijek nemamo točan podatak gdje se dodir dogodio, no ukoliko se stavi u omjer s naponom napajanja, tada se može točno odrediti mjesto dodira. U nastavku je opisan primjer:

U_{DD} = 3.3 V
U_X = 1.78 V

$$X_{dodira} = \frac{U_X}{U_{DD}} \cdot 100\% = \frac{1.78\,V}{3.3\,V} \cdot 100\% = 53.94\,\% \qquad (4\text{-}1)$$

To znači da je dodir u X osi na 54 % dodirnog dijela X osi. No, to i dalje ne daje nikakav podatak što je točno dodirnuto na zaslonu, pogotovo jer je panel osjetljiv na dodir veći od samoga zaslona. Stoga je potrebno izvršiti kalibraciju panela osjetljiv na dodir na način da se ispisuju podaci za svaki dodir na panelu osjetljivom na dodir i da se pri tome dodirnu dvije točke. Prva je mjesto gdje se na zaslonu ispisuje piksel s koordinatama (0,0), a druga je gdje se na zaslonu ispisuje zadnji piksel u x i y osi kojem su koordinate (799, 599). Prva nam prva nam točka daje kalibracijske podatke za

X_MIN i Y_MIN, a druga točka nam daje za X_MAX i Y_MAX. Nakon toga, ti brojevi se upišu trajno u kod i koriste se prilikom svakoga dodira (slika 4.25).

```
#define TS_XMIN 2767
#define TS_XMAX 591
#define TS_YMIN 2559
#define TS_YMAX 384

tsX = map(tsX, TS_XMIN, TS_XMAX, 0, 799);
tsY = map(tsY, TS_YMIN, TS_YMAX, 0, 599);
```

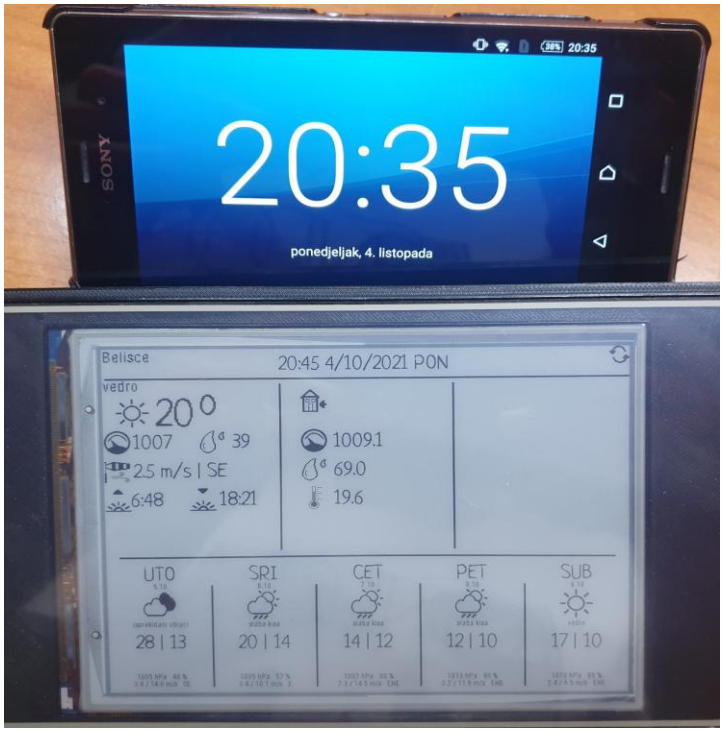
Slika 4.25. Programski kod za pretvaranje dodira za X i Y koordinatu na zaslonu

Programska funkcija map() je integrirana Arduino funkcija koja varijablu jednog opsega vrijednosti pretvara u varijablu s drugim opsegom vrijednosti. Prilikom testiranja ovog načina određivanja dodira, utvrđeno jest da se točnost vrlo jako mijenja s promjenom temperature i napona napajanja. Pretpostavlja se da je razlog sami analogno digitalni pretvornik unutar mikroupravljača razlog tomu, stoga se mora naći drugo rješenje, a to je korištenje zasebnog sklopa koji bi omogućio određivanje dodira i pretvorbu dodira u digitalnu riječ. Postoji veliki broj gotovih komercijalnih rješenja u obliku integriranoga kruga, od kojih je odabrano tri moguća za rješenje ovoga problema: TSC2046EIPW, BU1029MUV-E2 i AR1020T-I/SO. Svi se razlikuju po potrošnji, mogućnostima, točnosti, rezoluciji analogno-digitalnog pretvornika i naponu napajanja. Nakon testiranja svakog integriranog kruga posebno (za što je bilo potrebno i stvaranje jednostavne programske podrške za upravljanje i komunikaciju) određeno je da je TSC2046EIPW najbolji izbor zbog jednostavne kontrole, mogućnosti mjerenja napona baterije bez korištenja naponskog dijelila (koje bi narušilo potrošnju u režimu rada s niskom potrošnjom), male potrošnje kod se čeka na dodir (prošnja je manja od 1 μA) i napona napajanja. TSC2046EIPW koristi SPI komunikacijski protokol za komunikaciju uz još poneke dodatne izvode. Pa tako jedan od bitnih jest PENIRQ koji stvara prekidni zahtjev (engl. *Interrupt*) prema mikroupravljaču. To znači da mikroupravljač može biti stalno u režimu niske potrošnje, te se samo mora prebaciti u normalan režim rada kada se detektira dodir, odnosno kada se dobije impuls na PENIRQ pinu TSC2046EIPW integriranoga kruga. Pošto EPS32 nema više slobodnih ulaza, signal PENIRQ će biti poslan na MCP23017 integrirani krug na *e-paper* pločici (koji služi povećanju dostupnih ulaza i izlaza), te će zatim biti poslan ESP32 mikroupravljaču putem INTB izvoda na MCP23017 integriranome krugu. Na taj način se dobila vrlo velika točnost mjesta dodira koja se ne mijenja ovisno o temperaturi i naponu napajanja (jer TSC2046EIPW ima svoj

fiksan referentni napon koji ne ovisi o naponu napajanja, vidljivo na blok dijagramu u prilogu 5 [40]).

4.2.2. Problem s internim *RTC*-om ESP32 mikroupravljača

Da bi se znalo kada se mora napraviti novo mjerenje podataka, spajanje na Internet i izvršilo dohvaćanje nove vremenske prognoze ili pak dočekivanje podataka s vanjske jedinice, potreban je vrlo točan tajming koji bi se izvršavao pomoću sata. Sat se koristi umjesto tajmera u mikroupravljaču jer je točnost izvršavanja različitih događaja na razini jedne sekunde. Da bi se iskoristila jednostavnost i vrlo dobra integriranost prvi pokušaj bilo je korištenje integriranog RTC (engl. *Real Time Clock*) modula unutar ESP32 mikroupravljača. To nam omogućava da se vrlo jednostavno upravlja s satom pomoću već integriranih funkcija i vrlo jednostavno zada termini „buđenja“ mikroupravljača (prebacivanje režima rada). Za samo postavljanje sata se koristi NTP protokol (engl. *Network Time Protocol*) gdje se vremenska stanica spoji na Internet putem WiFi mreže, dohvati UNIX / POSIX vrijeme i podesi svoj sat. Na taj način imamo što je moguće točnije vrijeme, bez da ga sam korisnik mora podešavati. No, nakon testiranja uočen je problem s integriranim satom. Sat je nakon deset sati rada bio izuzetno netočan, gdje je griješio čak 10 minuta (slika 4.26). Razlog tomu jest što se kao izvor takta za sat nije koristio kvarcni kristal, koji ima veliku točnost, već se koristio RC oscilator koji ima malu točnost i vrlo je temperaturno ovisan. Sam integrirani sat ima mogućnost da se koristi vanjski kvarcni kristal od 32.768 kHz, no ti izvodi su već iskorišteni za upravljanje s *e-paper* panelom. Postoji mogućnost da se nakon nekog vremena sat ponovno sinkronizira putem NTP protokola, no i dalje ostaje problem gubitka sinkronizacije s vanjskom jedinicom.



Slika 4.26. Gubitak točnosti sata kod ESP32 mikroupravljača

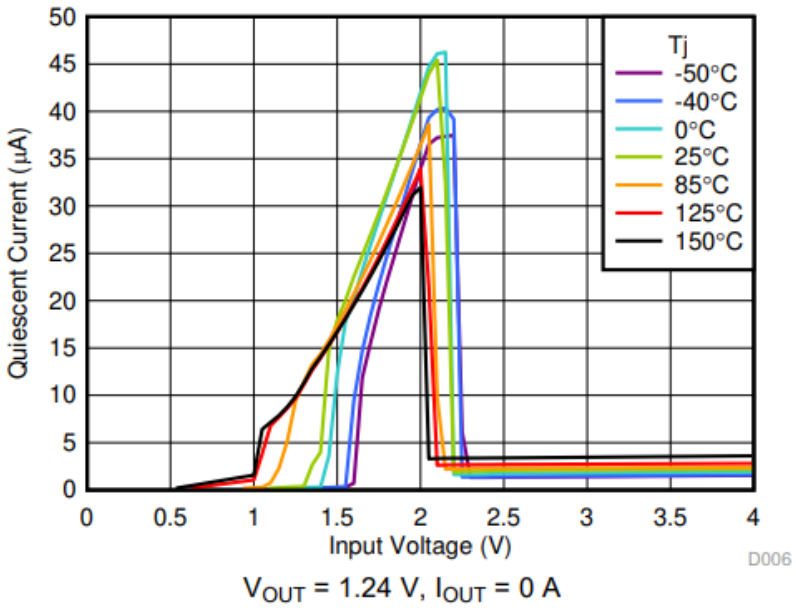
Drugi problem je što ne postoji programska podrška u Arduino IDE da se omogući prebacivanje izvora takta sata na neki drugi. Stoga je za tu svrhu korišten vanjski sat u obliku integriranoga kruga pod oznakom PCF85063. Korišteni sat ima vrlo malu potrošnju, ima vanjski izvor takta u obliku kvarcnog kristala od 32.768 kHz, mogućnost tajmera i alarma, te I2C komunikaciju. Nakon dodavanja vanjskoga sata, nije se primijetio ikakav gubitak točnosti sata.

4.3. Izrada vanjske jedinice

Vanjska jedinica se sastoji od mikroupravljača, senzora za temperaturu zraka, relativnu vlažnost zraka, atmosferskog tlaka zraka, senzora UV zračenja, senzora količine svjetlosti, senzora vjetra, senzora sunčevog zračenja, segmentnog LCD zaslona s tipkama, litijeve baterije i solarnog panela, radijskog modula, punjača litijeve baterije i naponskoga regulatora. Senzor za mjerenje temperature zraka jest SHT21 radi svoje velike točnost, male potrošnje i I2C komunikacije, za senzor tlaka zraka se koristi BMP180, UV zračenje i svjetlost Si1147 (trenutno jedini dobavljivi senzor za mjerenje UV zračenja), dok se za mjerenje brzine vjetra i smjera vjetra, te količine sunčevog zračenja koriste senzori vlastite izrade. Kompletna shema vanjske jedinice se može

pronaći u prilogu, prilog 6 i 7, dok se shema dodatne pločice za unutrašnju jedinicu nalazi u prilogu 8, 9 i 10.

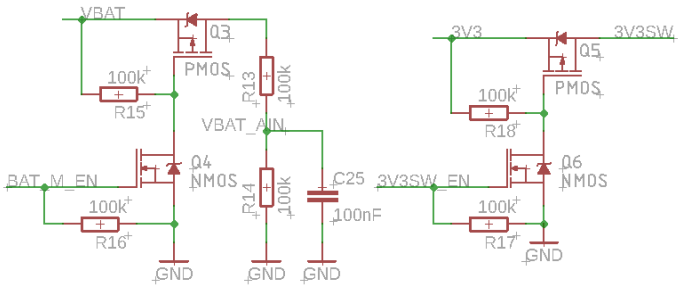
Primarna napajanja vanjske jedinice su punjiva baterija i solarni panel. Koristi se litijeva baterija od 3.6 V / 700 mAh koja ima već integriranu zaštitu. Baterije koje nemaju zaštitu nisu dozvoljene da se koriste na vanjskoj jedinici. Solarni panel koji je korišten ima nazivnu snagu od 1 W i napon otvorenog kruga od 6 V. Taj se napon šalje na TP4056 punjač koji puni litijevu bateriju. Napon s baterije se zatim šalje na linearni LDO naponski regulator (engl. *Linear Low-Dropout regulator*) koji spušta napon litijeve baterije koji može dosegnuti i 4.2 V na stabilnih 3.3 V potrebnih za rad ostatka sklopova. Pošto je riječ o LDO regulatoru napona, ulazni napon može biti gotovo jednak izlaznom naponu iz regulatora, a da pritom regulator uspije stabilno regulirati napon. Treba pripaziti da regulator može proizvoditi 500 mA struje, zbog nRF24l01 radijskog modula, a ujedno da ima malu struju mirovanja. Naponski regulator koji se koristi jest TPS7A2633DRVR koji može proizvoditi do najviše 970 mA, a da se pri tome izlazni napon smanji za 10 % nazivnog napona. Također, struja mirovanja mu je ne veća od 5 µA (slika 4.27 [41]) ukoliko je ulazni napon veći od 2.5 V.



Slika 4.27. Struja mirovanja TPS7A2633DRVR naponskog regulatora [41]

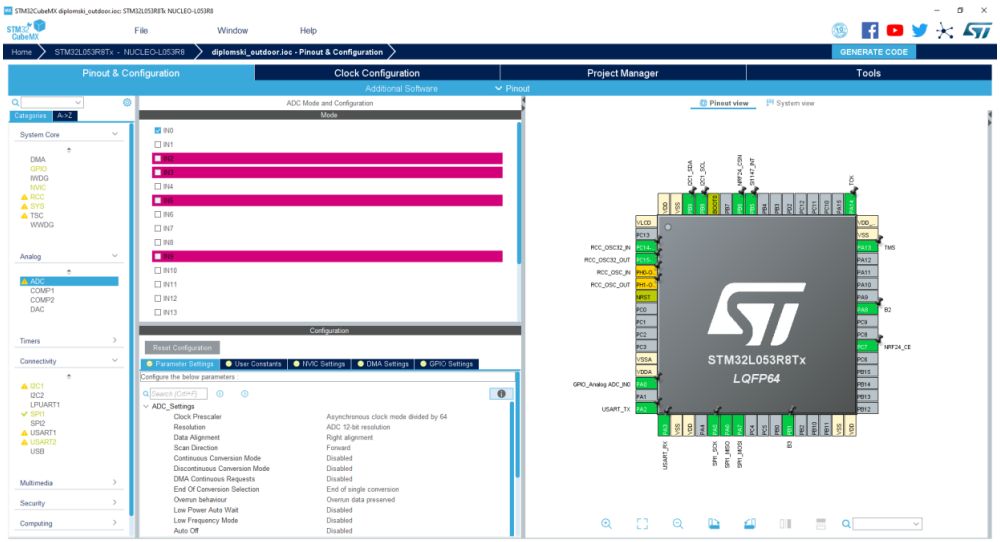
Prilikom dizajna tiskane pločice treba obratiti na pozornost da kondenzatori za filtraciju (engl. *decoupling capacitors*) budu što bliže njemu. To bi trebalo biti pravilo za sve integrirane krugove

koji se koriste na vanjskoj jedinici uključujući senzore, mikroupravljač, pojačalo itd. Nakon što je napon reguliran šalje se senzorima, izlazim priključnicama, LCD zaslonu, mikroupravljaču. Poneki dijelovi nemaju mogućnost režima rada niske potrošnje poput senzora sunčevog zračenja, te senzora za vjetar, stoga im je potrebno prekinuti napajanje kada se ne koriste. To se radi pomoću sklopa na slici 4.28. Sličan sklop se koristi i za aktivaciju naponskog dijelila za mjerenje napona baterije (da bi se smanjila potrošnja kada se mjerenje ne koristi).



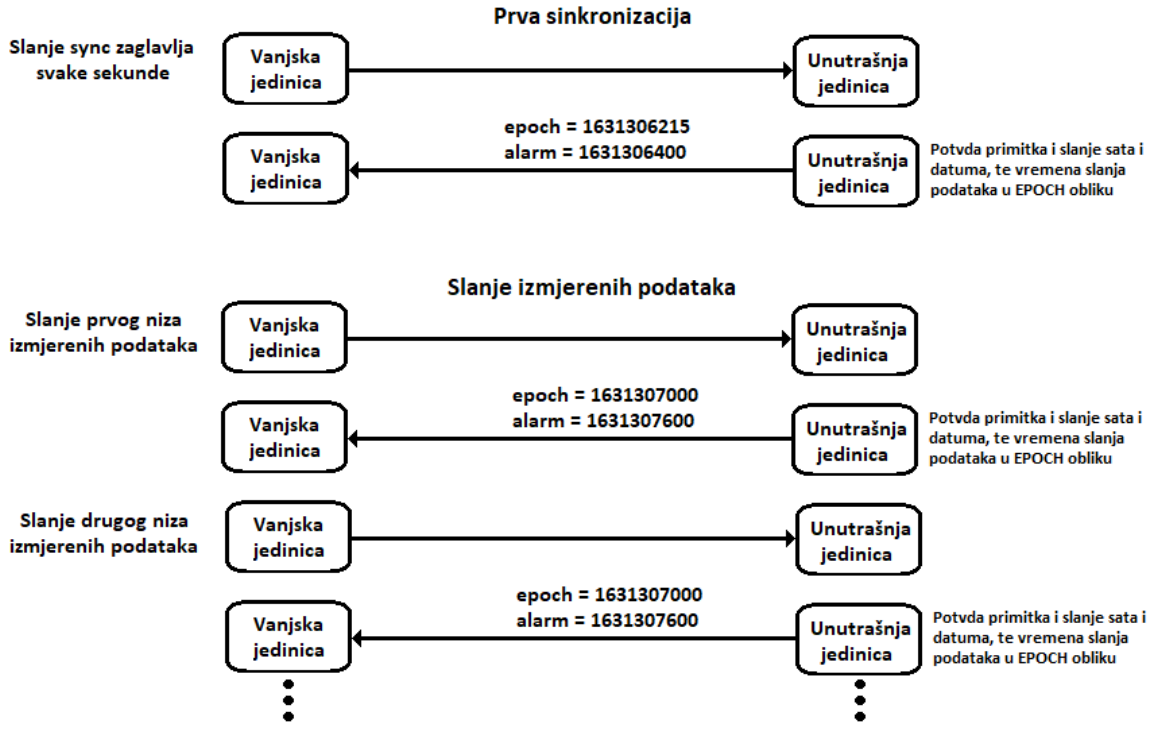
Slika 4.28. Sklop za mjerenje napona baterije (lijevo) i za deaktivaciju napajanja (desno)

Mikroupravljač je zaslužan za svu kontrolu ostatka sklopovlja. Da bi mikroupravljač ispravno radio, potrebno mu je uz napajanje i programski kod, kao i izvor takta. Mikroupravljač je programiran kroz Atollic TRUEStudio For STM32 razvojno sučelje, te je pisan u C programskome jeziku. Sve potrebne programske biblioteke za periferiju i komunikaciju su proizašle iz STM32CubeMX programskog paketa, gdje je moguće omogućiti pojedini dio periferije mikroupravljača i podesiti joj određene parametre (slika 4.29).



Slika 4.29. Izgled STM32CubeMX programa – Podešavanje analogno-digitalnog pretvornika

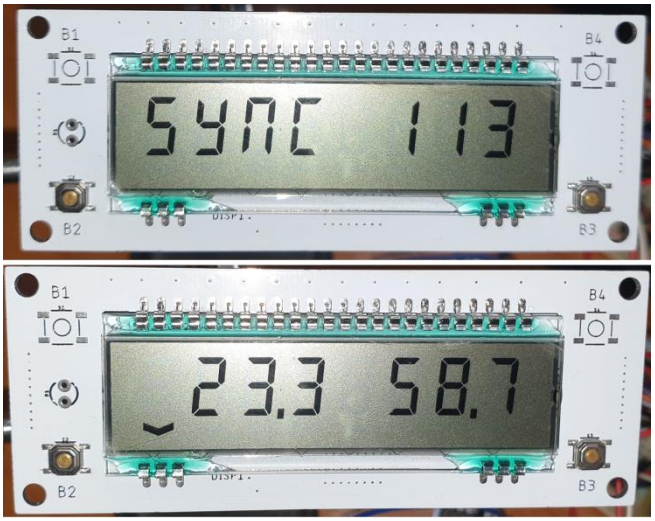
Mikroupravljač prikuplja podatke sa senzora u vremenskim intervalima od 10 minuta. Sve ostalo vrijeme, mikroupravljač s nalazi u režimu niske potrošnje energije. Da bi znalo kada mikroupravljač treba slati podatke, potreban mu je nekakvo praćenje vremena i to je izvedeno u obliku RTC-a. Podešavanje vremena se izvodi prilikom procesa sinkronizacije s unutrašnjom jedinicom. Naime da bi se omogućio režim Niske potrošnje energije na oba kraja, mora se znati točno vrijeme slanja podataka, a da bi uređaji bili spremni za slanje podataka, odnosno prijam podataka kod unutrašnje jedinice, njihovi satovi moraju biti usklađeni u sekundu. No, to samo po sebi nije dovoljno, jer će se nakon nekog vremena rada pojaviti drift između ta dva sata, stoga ih je potrebno redovito usklađivati, pa se prema tome usklađivanje vrši prilikom svakog slanja podataka unutrašnjoj jedinici u obliku odgovorna na primitak podatka (princip rada prikaza na slici 4.30).



Slika 4.30. Princip sinkronizacije unutrašnje i vanjske jedinice

Nakon primljene sinkronizacije, čeka se uvjet buđenja mikroupravljača, što može biti alarm na RTC-u ili da je korisnik pritisnuo tipku. Ukoliko je RTC alarm probudio mikroupravljač, tada se mjere mjerne veličine, spremaju u memoriju, slažu se paketi za slanje i podaci se šalju unutrašnjoj jedinici 10 puta u razmaku od jedne sekunde. Ukoliko veza s unutrašnjom jedinicom nije ostvarena, podaci su izgubljeni, a vrijeme buđenja je razlika vremena od prošle sinkronizacije i vremena buđenja od prošle sinkronizacije dodano na trenutno vrijeme oduzeto za 10 sekundi. Ukoliko je

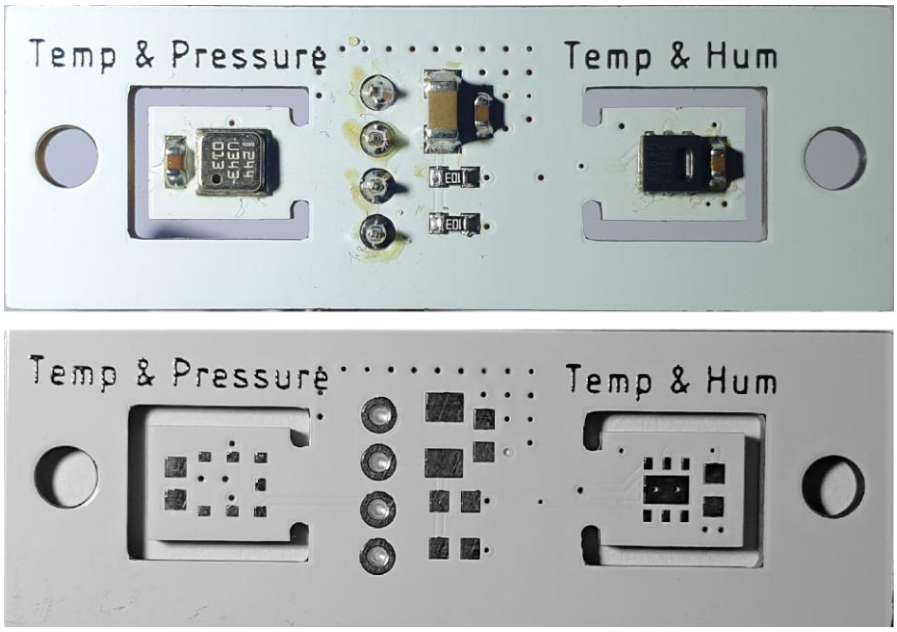
korisnik pritisnuo tipku, postoje dvije mogućnosti; ili korisnik odabire koje podatke želi vidjeti na LCD zaslonu od zadnjeg mjerenja ili pak želi da se odmah očitaju novi podaci mjernih veličina trenutno prikazanih na LCD zaslonu, no ti podaci neće biti poslani unutrašnjoj jedinici. LCD zaslon korišten na vanjskoj jedinici je 7 segmentni monokromatski reflektivni LCD zaslon radnog napona 3 V s 8 znamenki, bez ikakvog unutrašnjeg upravljača oznake DE133-RS-30/6.35-3. LCD je napravljen da prihvaća signal od 1/3 Bias-a i 1/3 multipleksa. Razlog odabira ovog zaslona jest izuzetno mala potrošnja (svega 25 μA dok je sadržaj prikazana na zaslonu uz PCF85176T/1,118), jako dobra vidljivost čak i u slučajevima s vrlo malo svijetla, čitljivost, jednostavno upravljanje, male dimenzije i vrlo širok temperaturni radni opseg (od -40 °C do +90 °C). Svako od ostalih rješenja (16x2 LCD, 128x64 grafički LCD baziran na ST7920 LCD upravljaču, 128x64 OLED, Nokia 5110 84x48 grafički LCD, 7 segmentni LED zaslon, *e-paper*) je imalo neku manu koja je ometala ispravan rad vanjske jedinice (dimenzije, potrošnja, komplicirano upravljanje, loša vidljivost, uzak temperaturni radni opseg, loša prilagodba na vanjske utjecaje). Slika 4.31 prikazuje LCD zaslon zajedno s tipkama i LCD upravljačem na poleđini.



Slika 4.31. LCD zaslon na pločici (gore: sinkronizacija, dolje: temperatura i vlaga zraka)

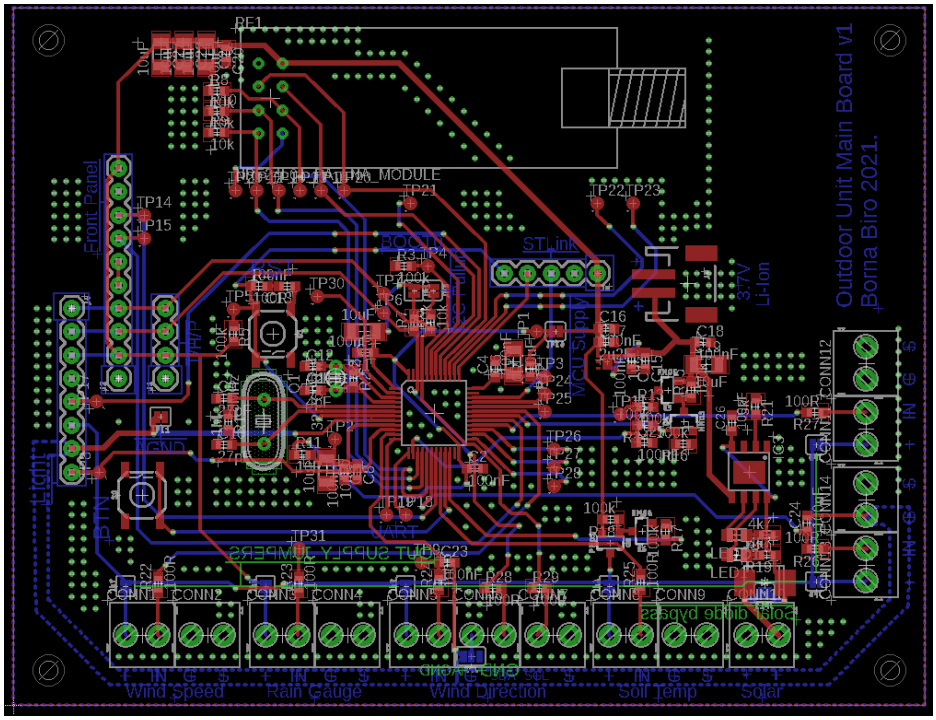
Također na slici 4.31. je vidljiva strelica u donjem desnome kutu LCD zaslona. Postoje 8 strelica (jedna ispod svake znamenke) koje prikazuju koji tip podatka se trenutno prikazuje na LCD zaslonu. Još jedna detalj treba primijetiti, a to je boja tiskane pločice koja je bijela, a razlog tomu jest da u slučaju da bude u istome kućištu kao i senzori i da postoji šansa da sunce dopre do nje, da se ne zagrijava veće da što više reflektira sunčevu toplinu, da rezultati mjerenja budu što točniji. Ista stvar je i sa pločicama za senzor svjetla i za senzor temperature, vlage i tlaka zraka. One imaju još jedan

detalj za primijetiti, a to su prorezi na tiskanoj pločici, da se omogući što manji prijenos temperature s pločice na senzor (slika 4.32)



Slika 4.32. Slika tiskanih pločica senzora

Jedini problem koji se pojavio jest nakon što je dodan nRF24l01 radijski modul i to u trenucima kada je cijela vanjska jedinica ušla u režim niske potrošnje energije. Radijski modul je počeo trošiti više struje nego što je očekivano. Razlog tomu je bio što su SPI prijenosne linije bile u stajnu visoke impedancije, time hvatajući šum i smetnje iz zraka aktivirajući modul i samim time povećanu potrošnju. Problem je riješen dodavanjem *pull down* otpornika. Slika 4.33 prikazuje dizajn kompletne glavne tiskane pločice vanjske jedinice. Pločica je dizajnirana u programu Autodesk Eagle.



Slika 4.33. Izgled glavne tiskane pločice vanjske jedinice (*ground pour* je uklonjen radi preglednosti)

Tiskana pločica je napravljena od FR4 materijala i dvostrana je, debljine 1.6 mm. Vidljivo je na slici da se nalazi veliki broj *via* koje spajaju dva sloja mase na pločici. To se radi da bi se smanjila strujna petlja (engl. *current loop*) i time smanjile smetnje u analognom dijelu kruga i stvorila veća stabilnost na RF krugu.

4.4. Izrada senzora vjetra i sunčevog zračenja

Zbog visoke cijene i relativno teške dostupnosti, senzori za određivanje brzine vjetra, smjera vjetra i senzor sunčevog zračenja biti će napravljeni u kućnoj izradi. Zbog toga što nisu komercijalni senzori, točnost će im biti manja, te će biti potreban proces ukalibravanja senzora, no takva izrada nam omogućava da se senzori naprave s točno željenim karakteristikama i da u slučaju kvara (pošto postoje mehaničke komponente senzora) budu vrlo lako popravljivi.

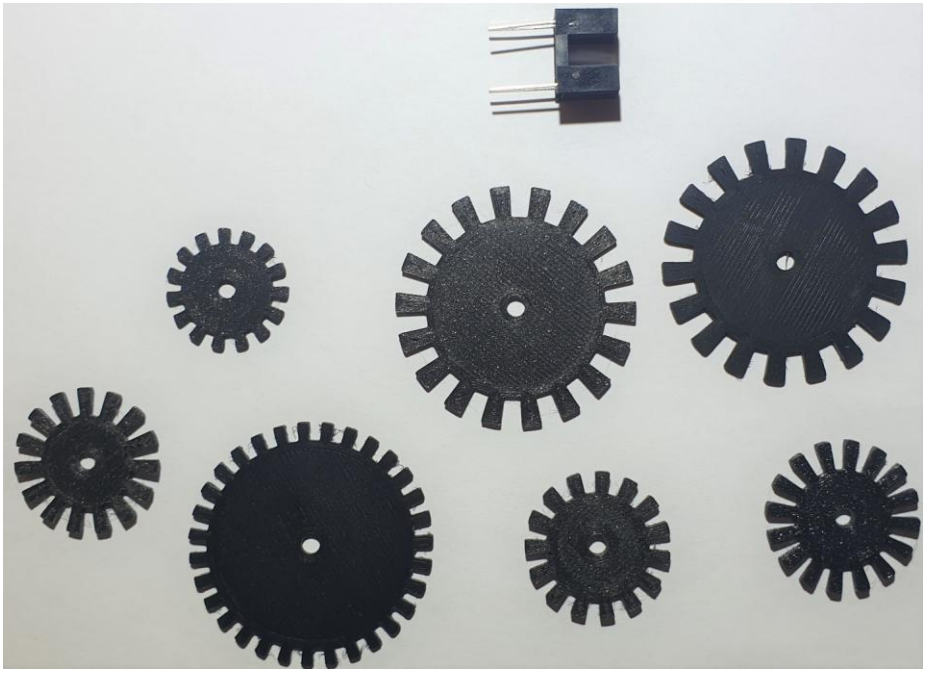
4.4.1. Senzor brzine vjetra

Senzor brzine vjetra (anemometar) radi na principu da brzinu vjetra pretvori u neki oblik električnog signala. Postoje dvije glavne podjele anemometra s obzirom na princip rada; anemometar s lopaticama i ultrazvučni anemometar (slika 4.34).

Slika 4.34. Ultrazvučni anemometar [42]

Ultrazvučni anemometar nema nikakvih mehaničkih dijelova, jer mu se princip rada bazira na pretpostavci da se brzina zvuka mijenja s količinom vjetra. Pošto se zna točna udaljenost između ultrazvučnog prijamnika i predajnika, mjerenjem vremenskog perioda između slanja i primitka ultrazvučnog signala može se dobiti podatak o brzini zvuka iz kojega se naknadno može odrediti brzina vjetra. Također, koristeći niz mjerenja s više senzora unutar jednoga anemometra, moguće je i izračunati smjer vjetra. Iz opisa principa rada vidi se da je riječ od dosta kompliciranom senzoru za napraviti, stoga je odluka da se izradi mehanički anemometar s lopaticama. Njegov princip rada se bazira na tome da vjetar pokreće lopatice koje su spojene na osovinu na kojoj se nalazi senzor koji rotaciju pretvara u napon, struju ili neki električni impuls. Napon i struju čiji iznosi ovise o brzini vrtnje moguće je napraviti pomoću nekog malog generatora električne energije. Ukoliko se koristi ovaj način pretvorbe brzine vrtnje u električni signal, može postojati problem s radnim opsegom. Naime, da bi se detektirale male brzine vrtnje, potreban je generator koji na malom broju okretaja proizvodi dovoljan visok napon da bi bio detektiran od strane analogno-digitalno pretvornika u mikroupravljaču, no tada se može dogoditi situacija da se pri većim brzinama vjetra stvara previsok napon koji može oštetiti sklopovlje. Također, tu opstoji problem i održavanja samoga generatora, jer je riječ o još jednoj mehaničkoj komponenti. Druga opcija je da se nađe senzor koji će brzinu vrtnje pretvoriti u broj impulsa (veća brzina vrtnje bi tada rezultirala u većem broju impulsa). Takav senzor je moguće izvesti pomoću magneta i engl. *reed switch* komponente. *Reed switch* je ustvari prekidač koji se nalazi unutar cjevaste staklene ovojnice i osjetljiv je na magnetsko polje. Ukoliko se nalazi u magnetskome polju, kontakti prekidača se spoje. Ugrađujući magnet na osovinu anemometra i

postavljajući *reed* prekidač u neposrednu blizinu osovine, može se detektirati svaki puni okret lopatica anemometra. No, problem je što za vrlo male brzine vjetra, period impulsa može biti dugačak čak i po nekoliko sekundi, što definitivno nije pogodno za režim rada niske potrošnje kod mikroupravljača. Bilo bi dobro da se dobije više impulsa za jedan okret, no to je gotovo nemoguće izvesti pouzdano pomoću *reed* prekidača. Stoga je odluka pala da se koristi foto-optički prekidnik s diskom (slika 4.35) koji ima zubiće postavljene na njegov rub. Sami disk bi bio tada ugrađen na osovinu anemometra, gdje bi za puni okret omogućavao više kraćih impulsa. Tada bi se i pri manjim brzinama vjetra detektirali impulsi, što bi skratilo vrijeme mjerenja brzine vjetra, a time i vrijeme koliko je dugo mikroupravljač u režimu normalnog rada veće potrošnje energije.

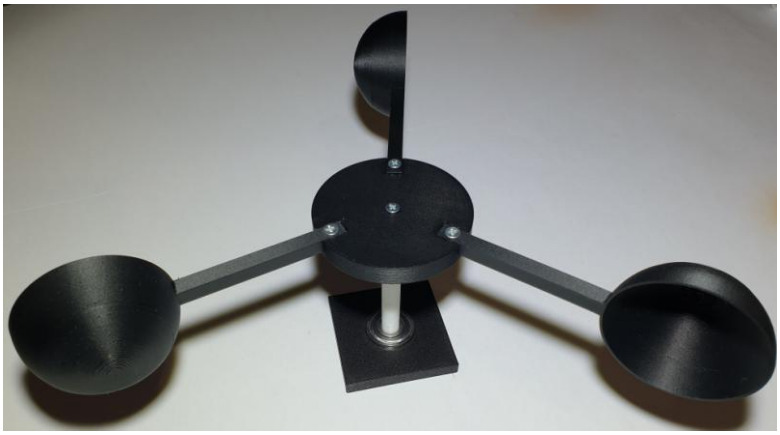


Slika 4.35. Foto-optički prekidnik s različitim dizajnima diskova

Diskovi su izrađeni od PETG plastike na 3D printeru, a dizajnirane su u programu FreeCAD. Disko koji je koristi u senzoru polumjera je 12.7 mm i ima 16 zuba. Foto-optički prekidnik koji je korišten jest ITR9608. Sastoji se od infracrvene LED diode i foto tranzistora. Struja kroz LED diodu je podešena da bude iznosa 5 mA na naponu napajanja od 3.3 V. Ako je U_{FLED} = 1.2 V, tada se može izračunati otpor serijskog otpornika za LED diodu:

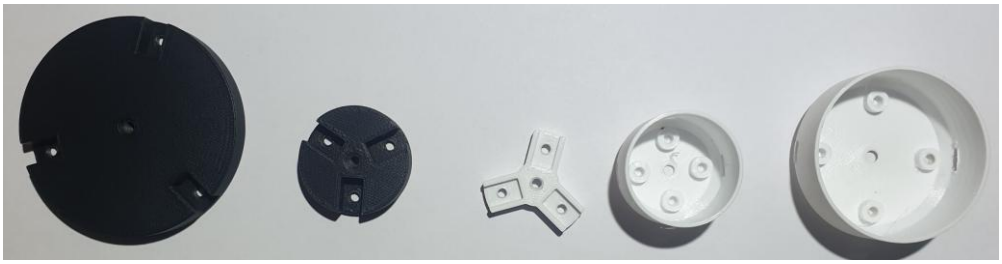
$$R_{LED} = \frac{U_{DD} - U_{FLED}}{I_{LED}} = \frac{3.3 - 1.2}{5\ mA} = 420\ \Omega \qquad (4\text{-}2)$$

Kako najbliži otpornici su 470 Ω i 330 Ω, te je odabran otpornik od 330 Ω. Struja tada iznosi 6.3 mA. Nadalje, potrebno je dizajnirati lopatice, osovinu i kućište samoga senzora. Ti dijelovi će također biti dizajnirani u FreeCAD programu i napravljeni na 3D printeru s PETG materijalom. No, prije toga je potrebno osmisliti dizajn samih lopatica, ponajviše njihovu veličinu, način učvršćivanja na osovinu i dužinu nosača lopatice. Svi ti parametri mogu odigrati važnu ulogu kada je riječ o samoj osjetljivosti senzora na male iznose vjetra kao i o izdržljivosti na velike udare vjetra.



Slika 4.36. Izgled prvoga prototipa

Prvi prototip je vidljiv na slici 4.36. Ovaj prototip se nije nabolje pokazao u praksi, jer uopće nije reagirao na male iznose vjetra, a na većima bi se tek nešto malo zaokrenuo. Stoga je bilo potrebno smanjiti dio koji lopatice spaja s osovinom (slika 4.37).

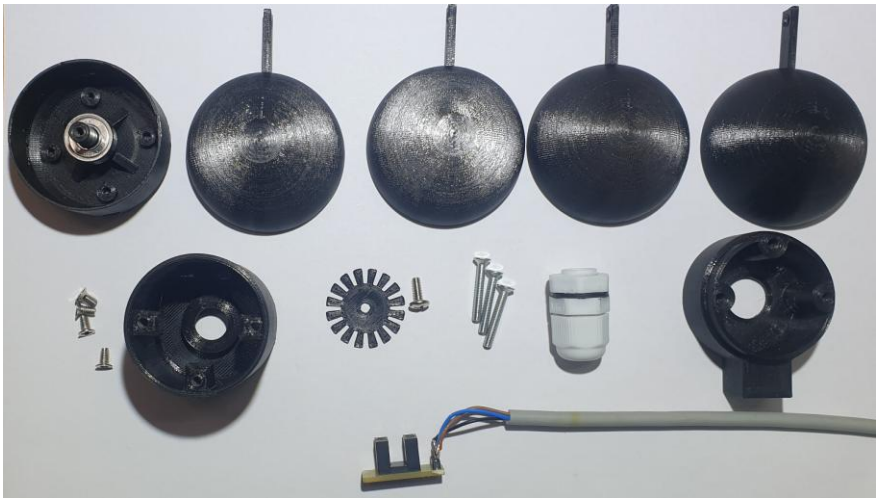


Slika 4.37. Različiti dizajni nosača lopatica (desni je najbliži finalnome dizajnu)

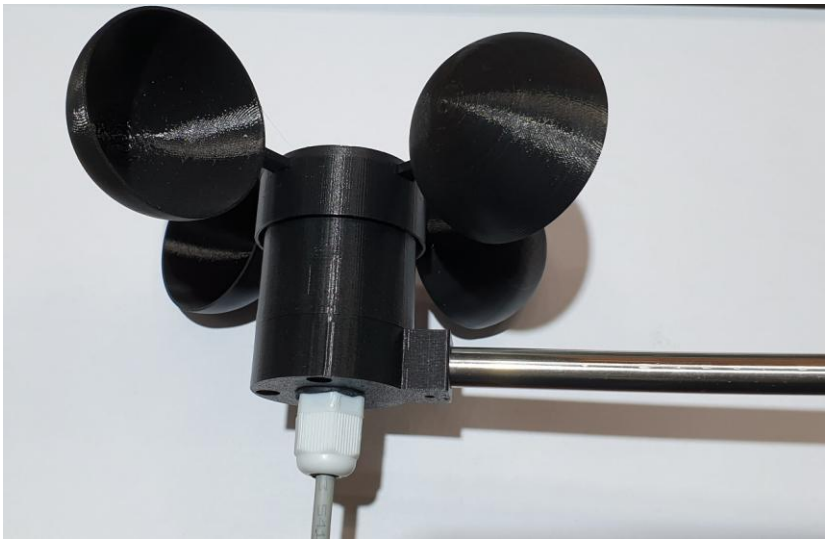
Poboljšanje je postojalo, no i dalje nije bilo nikakvoga odziva na male vjetrove. Nakon toga se pokušalo mijenjati parametre samih lopatica (slika 4.38), iako je bilo poboljšanja prilikom smanjivanja mase lopatice i dalje je odziv senzora na male vjetrove bio vrlo slab.

Slika 4.38. Različiti dizajni lopatica (u donjem desnome kutu je finalni dizajn)

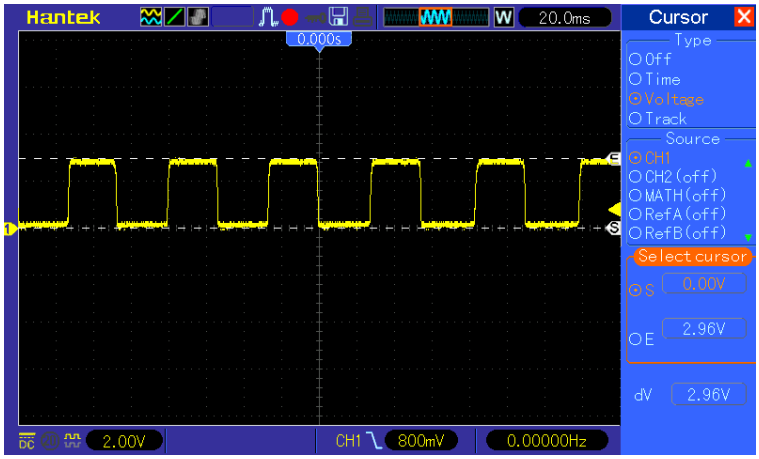
Najveće poboljšanje je pridobilo promjenom ležaja. Prvobitno korišteni ležaj (608-ZZ) na osovini anemometra je imao vrlo veliko trenje, stoga je zamijenjen s manjim ležajem 619/6 ZZ. Nakon toga je postojao odziv i na manje vjetrove. Završno, potrebo je bilo napraviti kućište. Kućište mora biti tamno kako vanjska svjetlost ne bi ometala ispravan rad foto-optičkog prekidinika, te ujedno mora biti vodootporno i mora postojati mogućnost pričvršćenja na nosač. Slika 4.39 pokazuje sve dijelove senzora, a slika 4.40 pokazuje kompletno sklopljeni senzor. Izlazni signal iz senzora vidljiv je na slici 4.41.



Slika 4.39. Dijelovi senzora brzine vjetra (anemometra)



Slika 4.40. Slika sklopljenog senzora

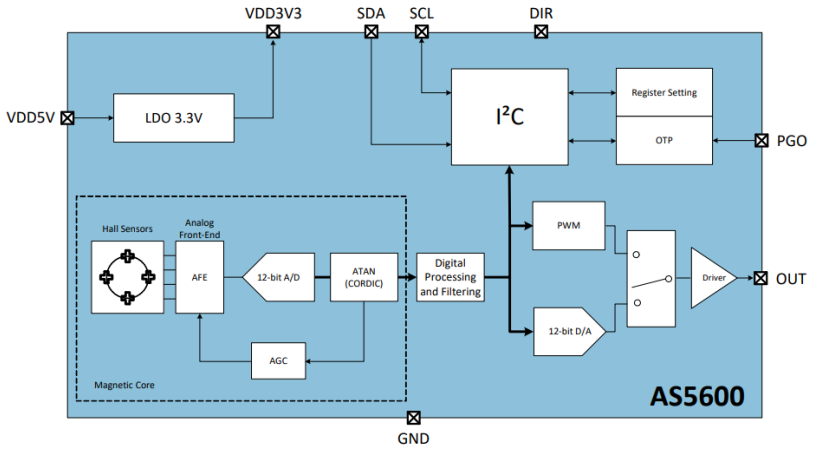


Slika 4.41. Valni oblik izlaznog signala iz senzora

4.4.2. Senzor smjera vjetra

Senzor smjera vjetra radi na principu da smjer odnosno poziciju mehaničkog pokaznika vjetra pretvara u neku vrstu električnog signala. Električni signal tada može biti napon, struja, impuls ili pak digitalna riječ (koliko se koristi neki komunikacijski protokol). Načini na koji se može pretvoriti pozicija u neki električni signal su različiti, ali pri tome treba paziti da način omogućava kontinuiranu detekciju pozicije za puni krug i to neograničen broj okretaja, te da ne ograničava ispravan rad senzora (npr. velikim trenjem). Neki od načina izvedbe pretvorbe pozicije u neki električan signal mogu biti izvedeni pomoću *reed* prekidača i magneta, foto-optičkog prekidnika i apsolutnog rotirajućeg enkodera (engl. *Absolute Rotary Encoder*) koji mogu biti mehanički, optički, kapacitivni i magnetski. Kod *reed* prekidača, postoji problem točnosti pošto niti jedan *reed* prekidač

nije identičan, stoga kut aktivacije prekidača za svaki prekidač neće biti isti. Osim toga, magnet zajedno s nekoliko *reed* prekidača (4 ili 8, za svaku stanu svijeta) može stvoriti kočenje mehaničkog pokaznika, a time i netočnu detekciju. Foto-optički prekidnik bi se mogao koristiti na način sličan onome za *reed* prekidač, gdje ne bi postojali problemi točnosti i kočenja, ali je detekcija i dalje ograničena na 4 ili 8 strana svijeta. Korištenje apsolutnog rotirajućeg enkodera je ekonomski neisplativo rješenje i postoji problem velikih dimenzija i osjetljivosti na vanjske uvjete. Stoga se pretvorba pozicije vrši magnetski pomoću integriranog kruga AS5600 (blok dijagram integriranoga kruga vidljiv na slici 4.42) koji može detektirati poziciju magneta i zatim taj podatak poslati u obliku analognog napona (0 V predstavlja kut od 0 °, a napon napajanja predstavlja kut od 359 °) ili digitalne riječi putem I2C protokola (iz digitalne riječi se može direktno očitati kut uz skaliranje).

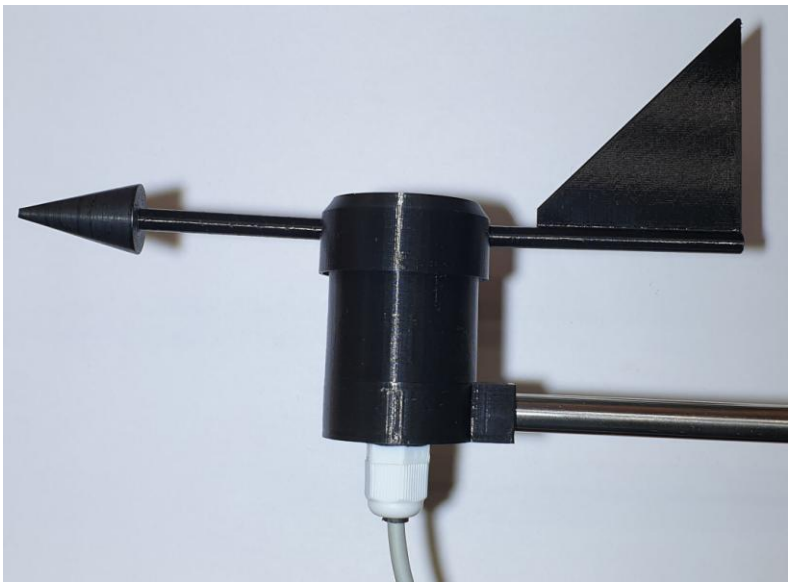


Slika 4.42. Blok dijagram AS5600 integriranog kruga [43]

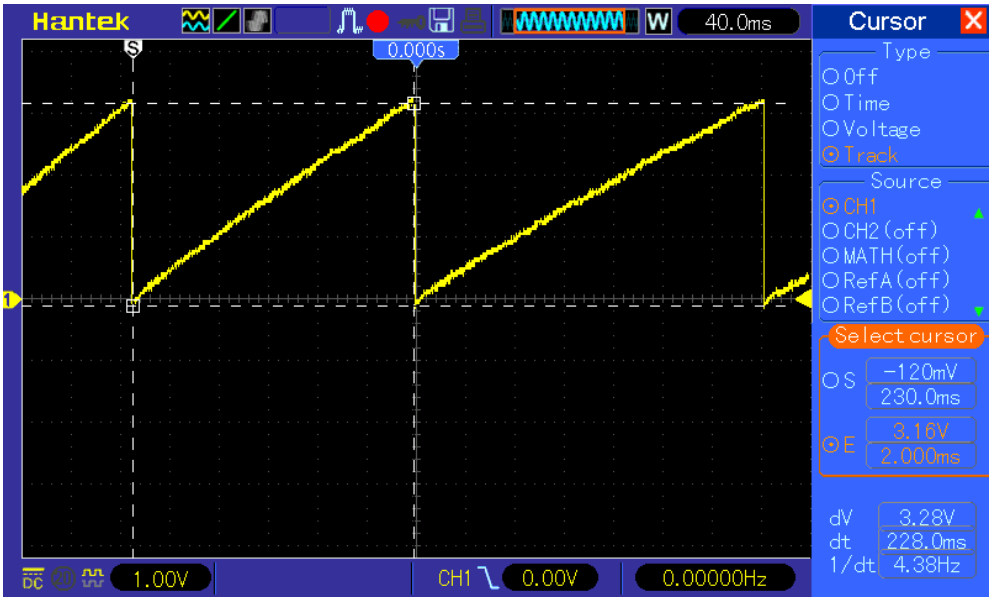
Kao i kod senzora za detekciju brizne vjetra, potrebno je napraviti pokaznik smjera vjetra, nosač pokaznika vjetra (ujedno i nosač magneta) i samo kućište. Ovdje je također kućište dizajnirano u FreeCAD programu, a zatim realizirano na 3D printeru. Najzahtjevniji dio za napraviti jest dio senzora koji se usmjerava prema smjeru puhanja vjetra. Da bi se to postiglo, na prednjem dijelu se mora nalaziti šiljak, a na zadnjem „pero“ čija je površina značajno veća od površine šiljka. Tu treba također pripaziti da centar mase bude na sredini osovine, inače pokaznik neće biti u balansu i neće pravilno pokazivati smjer vjetra. To je načinjeno na način da se nosač šiljka napravio veći, te se skraćivao, dok se nije pronašla ravnoteža prednjeg i zadnjeg kraja pokaznika. Nakon toga se nanovo isprintao dio točno te dimenzije. Slika 4.43. pokazuje sve dijelove senzora, slika 4.44 sastavljeni senzor. Slanje informacije o poziciji vrši se analogno pomoću napona, radi pouzdanosti prijenosa

(jer će uređaj biti u vanjskim uvjetima, gdje postoji šansa za atmosferska pražnjenja, koja mogu onemogućiti digitalnu I2C komunikaciju), a signal je vidljiv na slici 4.45.

Slika 4.43. Dijelovi senzora smjera vjetra

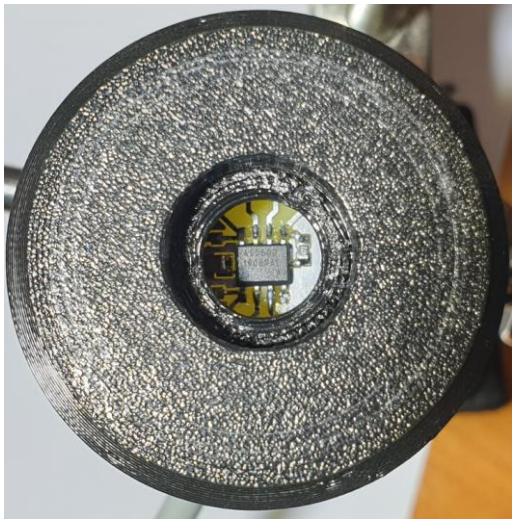


Slika 4.44. Slika sklopljenog senzora smjera vjetra

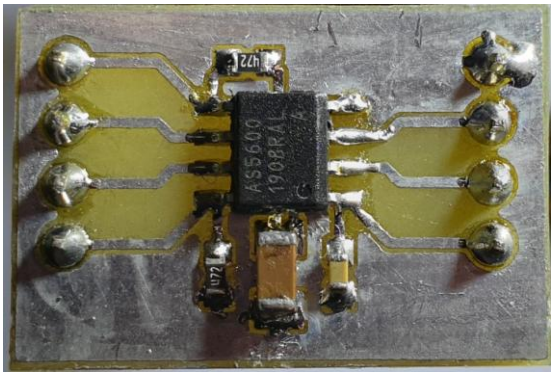


Slika 4.45. Izlazni signal s senzora (rotacija pokaznika udesno)

Treba napomenuti da je vrlo bitno da integirani krug bude u sredini osovine (slika 4.46) i da magnet bude na dovoljnoj udaljenosti od integriranog kruga (nije ispravno da bude ni preblizu, a ni predaleko od integriranog kruga), a da bi se to izvelo i da bi ga se moglo pričvrstiti za kućište, napravljena je mala tiskana pločica (slika 4.47).



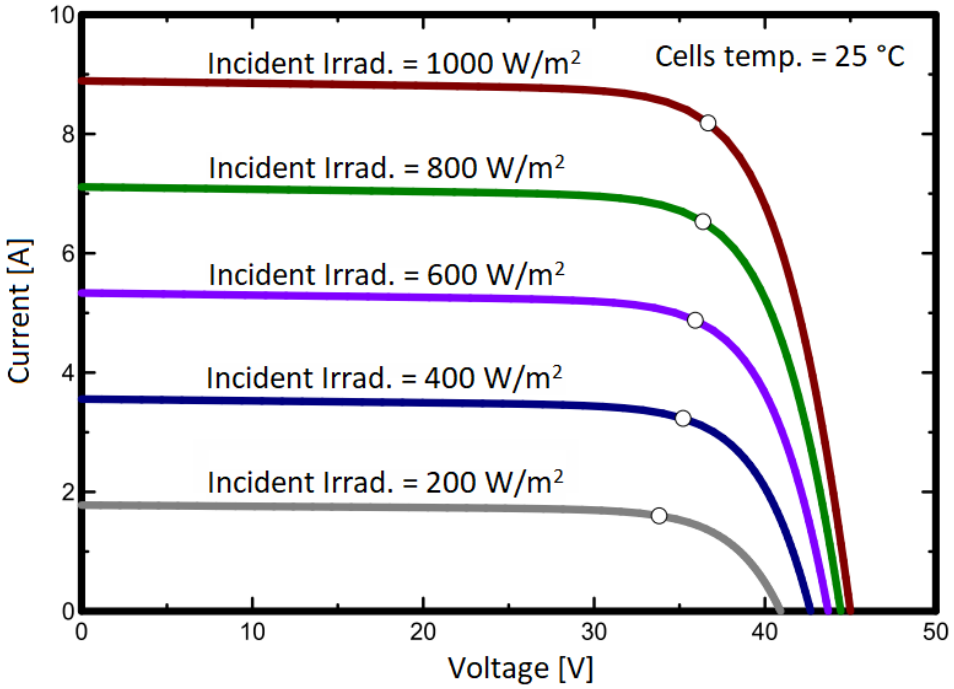
Slika 4.46. Integrirani krug u sredini osovine (za ispravnu detekciju)



Slika 4.47. Tiskana pločica AS5600 integriranog kruga

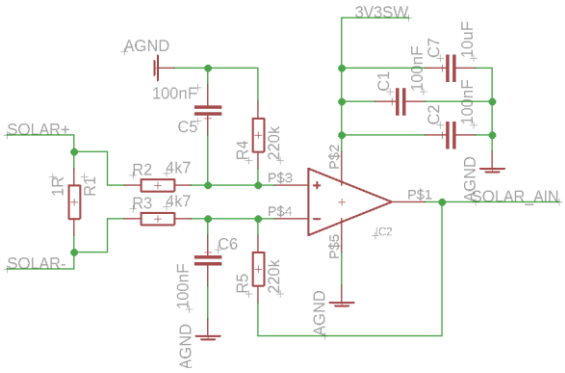
4.4.3. Senzor sunčevog zračenja

Senzor sunčevog zračenja služi za detekciju količinu sunčeve energije koja pada na neku površinu. Najvažniji dio senzora jest solarna ćelija. Solarna ćelija može pretvoriti sunčevu energiju u električnu energiju, stoga ona sama po sebi može biti mjerni pretvornik. Način rada je da se mjeri struja kratkoga spoja kroz ćeliju. Razlog tomu jest da struja kratkog spoja linearno ovisi o količini energije koja pada na panel. Graf na slici 4.48 upravo prikazuje tu ovisnost. Može se primijetiti da struja kratkog spoja na 400 W/m^2 iznosi približno 3.55 A, dok na duplo većem iznosu energije od 800 W/m^2 iznosi približno 7.1 A, (vidljivo je da se struja također za duplo povećala). Time se dokazala linearnu ovisnost struje o količini sunčevog zračenja.



Slika 4.48. Graf ovisnosti struje i napona o količini izračene energije na panel [44]

To znači da ukoliko se može pretvoriti struja u napon koji može biti pretvoren u digitalnu riječ pomoću analogno-digitalnog pretvornika u mikroupravljaču, može se mjeriti količina sunčevog zračenja. To je izvedivo pomoću shunt otpornika. Treba pripaziti da otpor otpornika bude dovoljno malen da struja koja će prolaziti kroz njega bude zaista struja kratkog spoja ćelije (odnosno da napon bude što je bliže moguće nuli). No, ukoliko je napon premaloga iznosa, neće se ispravno moći izvršiti analogno-digitalna pretvorba. Iz tog razloga, potrebno je napraviti pojačalo koje će pojačati signal. Pojačalo koje se koristi jest MCP601RT-I/OT. U pitanju je operacijsko pojačalo radnoga opsega od 2.7 V do 6 V, te ima engl. *rail-to-rail* izlaz koji omogućuje da izlazni napon ima opseg jednak naponu napajanja. Shema je vidljiva na slici 4.49.



Slika 4.49. Shema diferencijalnog pojačala i pretvornika struje KS solarne ćelije u napon

Da bi se odredilo pojačanje pojačala, potrebno je provjeriti koji je najveći napon koji se može pojaviti na shunt otporniku. Ćelija koja se koristi daje 45 mA struje pri 1000 W/m^2. No treba imati na umu da je moguće dobiti i više od 1000 W/m^2, stoga računajući 50 % više dobijemo 1500 W/m^2. Ukoliko uzmemo pretpostavku da će se struja linearno povećavati, struja kratkog spoja bi iznosila:

$$I_{KS\,1500} = I_{KS\,1000} \cdot 1.5 = 67.5\,mA \tag{4-3}$$

$$U_{R1} = I_{R1} \cdot R_1 = 67.5\,mA \cdot 1\,\Omega = 67.5\,mV \tag{4-4}$$

Iz tih podataka moguće je dobiti traženo pojačanje pojačala:

$$A_V = \frac{3.3\,V}{U_{R1}} = \frac{3.3\,V}{67.5\,mV} = 48.8 \tag{4-5}$$

Ukoliko se odredi da su otpornici R_2 i R_3 isti, te da su otpornici R_4 i R_5, te ukoliko proizvoljno odredimo da je R_2 = 4.7 kΩ, dobije se da je R_4 jednak:

$$A_V = \frac{R_4}{R_2} => R_4 = A_V \cdot R_2 = 48.8 \cdot 4.7\, k\Omega = 229.36\, k\Omega \quad (4\text{-}6)$$

Najbliži otpornik koji se može koristiti jest 220 kΩ. Treba napomenuti da se ne smiju koristiti solarni paneli, već solarne ćelije, jer imaju niži unutrašnji otpor koji direktno utječe na točnost mjerenja.

4.5. Programski kod

U ovome dijelu će biti opisani neki od dijelova programskoga koda unutrašnje i vanjske jedinice. Programski jezik koji se koristi za programiranje vanjske jedinice je C, dok se za unutrašnju jedinicu koristi C/C++ prilagođen za Arduino programsku platformu. Programski kod na svakoj od jedinica je razdvojen u više datoteka. Da bi se posao programiranja olakšao, koriste se biblioteke s već napisanim funkcijama. Biblioteke koje se koriste su pod licencom otvorenog koda (engl. *open source*). Neke od korištenih biblioteka su:

- Adafruit BME280 – Arduino biblioteka za BME280 senzor
- Adafruit SGP30 – Arduino biblioteka za SGP30-2.5K senzor
- Adafruit GFX – Grafička Arduino biblioteka
- ArduinoJson – Arduino biblioteka za raščlanjivanje JSON podataka
- Inkplate – Arduino biblioteka za upravljanje *e-paper* zaslonom (rađena zajedno s firmom TAVU d.o.o. pod brandom e-radionica)

Ostale biblioteke (navedene u nastavku) su također pod licencom otvorenog koda, no da bi radile, potrebno je bilo napraviti razne modifikacije:

- SHT21 by e-radionica.com – Arduino bibioteka za SHT21 senzor
- Sparkfun BMP180 – Arduino biblioteka za BMP180 senzor
- RF24 – Arduino biblioteka za nRF24l01 radijski modul

Ostale biblioteke su dio programskog razvojnog okruženja (poput biblioteke za HTTP protokol, UDP protokol, WiFi komunikaciju kod ESP32 mikroupravljača ili poput biblioteke za RTC, SPI komunikaciju, I2C komunikaciju, upravljanje potrošnjom energije kod STM32L073 mikroupravljača) ili su vlastoručno napisane (poput biblioteke za segmentni LCD, Si1147 senzor, raščlanjivanje podataka vremenske prognoze i sl.).

4.5.1. Unutrašnja jedinica

Prilikom pokteranja unutrašnje jedinice potrebno se spojiti na odgovarajuću WiFi mrežu, te ispisati sve dostupne WiFi mreže na zaslon. Ukoliko mreža zahtjeva lozinku, pored imena će se ispisati simbol „*“ i brojčana vrijednost koja označava snagu prijama u dBm. Također, u tom dijelu koda se inicijaliziraju biblioteke za TSC2046 i MCP23017, te se mijenja ime uređaja na WiFi mreži (da bude lakše prepoznatljiv). Kodovi su vidljivi na slikama 4.50 i 4.51.

```
uint8_t wiFiRetry = 30;
int n = WiFi.scanNetworks();
if (n > 6) n = 6;
display.clearDisplay();
display.setCursor(0, 24);
display.println("ESP32 WeatherPaper\nTrazenje WiFi Mreza...");
display.display();
for (int i = 0; i < n; i++)
{
  display.setCursor(70, 100 + (i * 35));
  display.print(WiFi.SSID(i));
  display.print((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? ' ' : '*');
  display.print(' ');
  display.print(WiFi.RSSI(i), DEC);
}
display.partialUpdate();
display.setCursor(70, 550);
display.print(F("Spajanje na "));
display.print(ssid);
display.partialUpdate(false, true);
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED && wiFiRetry > 0)
{
  delay(1000);
  display.print('.');
  display.partialUpdate(false, true);
  wiFiRetry--;
}
if(WiFi.status() == WL_CONNECTED)
{
  display.print("spojeno! Pricekajte...");
  display.partialUpdate();
}
else
{
  display.print("Spajanje neuspjesno!");
  display.display();
  esp_deep_sleep_start();
  while (1);
}
```

Slika 4.50. Kod za pretragu dostupnih WiFi mreža

```
Serial.begin(115200);
Wire.begin();
display.begin();
EEPROM.begin(64);

// Set output mode of MCP INT pin
display.setIntOutputInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 1, 0, 0, 1);

// Set touchscreen controller INT pin
display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 13, INPUT_PULLUP);
display.setIntPinInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 13, FALLING);

// Set default pin state on touchscreen controller CS pin
display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 10, OUTPUT);
display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 10, HIGH);

// Set default state of nRF24l01 pins
display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 14, HIGH);
display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 15, HIGH);

// Set RTC INT PIN
display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 12, INPUT_PULLUP);
display.setIntPinInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 12, FALLING);

// Set 3V3SW pin as output
display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 11, OUTPUT);
display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 11, LOW);

// Clear all INT flags on MCP
display.getINTstateInternal(MCP23017_INT_ADDR, display.mcpRegsInt);

// Init GUI communication and RTC library
gui.init(&display);
rf.init(&radio, &display, &rtc);
rtc.RTCinit();

// Get the SPI object from Inkplate library
mySpi = display.getSPIptr();
mySpi->begin(14, 12, 13, 15);

// Init touchscreen controller
ts.begin(mySpi, &display, 10, 13);
ts.calibrate(800, 3420, 3553, 317, 0, 799, 0, 599);
```

Slika 4.51. Kod za inicijalizaciju biblioteka i podešavanje imena na WiFi mreži

Nakon toga, potrebno je dohvatiti vrijeme i datum s Internet mreže putem NTP protokola, koji daje vrijeme u EPOCH/POSIX obliku, što je predstavljeno s 32 bitnom vrijednošću (slika 4.52)

```
bool readNTP(time_t *_epoch)
{
  IPAddress ntpIp;
  WiFiUDP udp;
  const char* NTPServer = "hr.pool.ntp.org";
  uint16_t ntpPort = 123;
  uint8_t ntpPacket[48];
  unsigned long _ntpTimeout;

  udp.begin(8888);
  if (!WiFi.hostByName(NTPServer, ntpIp)) return 0;

  ntpPacket[0] = B11100011; //Clock is unsync, NTP version 4, Symmetric passive
  ntpPacket[1] = 0;     // Stratum, or type of clock
  ntpPacket[2] = 60;     // Polling Interval
  ntpPacket[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  ntpPacket[12]  = 49;
  ntpPacket[13]  = 0x4E;
  ntpPacket[14]  = 49;
  ntpPacket[15]  = 52;
  udp.beginPacket(ntpIp, 123);
  udp.write(ntpPacket, 48);
  udp.endPacket();
  _ntpTimeout = millis();
  while ((unsigned long)(millis() - _ntpTimeout) < 5000)
  {
    if (udp.parsePacket() >= 48)
    {
      udp.read(ntpPacket, 48);
      uint32_t unix = ntpPacket[40] << 24 | ntpPacket[41] << 16 | ntpPacket[42] << 8 | ntpPacket[43];
      *_epoch = unix - 2208988800UL + currentWeather.timezone;
      return true;
    }
  }
  return false;
}
```

Slika 4.52. Kod za dohvaćanje sata i datuma putem NTP protokola

Vremensku prognozu je potrebno preuzeti s Interneta koristeći HTTP protokol i JSON notaciju zapisa podataka putem API zahtjeva. API zahtjev za podacima se vrši pomoću linka i HTTP GET naredbe. Nakon što su podaci primljeni, potrebno je provjeriti ispravnost podataka, te ih onda raščlaniti u strukture podataka vremenske prognoze (slike 4.53 i 4.54).

```
uint8_t OWMWeather::getForecastWeather(const char* _url, struct forecastListHandle *_f,
struct forecastDisplayHandle *_d)
{
  WiFiClient client;
  HTTPClient http;
  DynamicJsonDocument doc(24576);
  http.useHTTP10(true);

  if (http.begin(client, _url))
  {
    if (http.GET() > 0)
    {
      DeserializationError err = deserializeJson(doc, http.getStream());
      if (err) return 0;
      if (atoi(doc["cod"]) == 200)
      {
        _f->numberOfData = doc["cnt"];
        for (int i = 0; i < _f->numberOfData; i++)
        {
          removeCroLetters(strncpy(_f->forecast[i].weatherDesc,
doc["list"][i]["weather"][0]["description"], sizeof(_f->forecast[i].weatherDesc) - 1));
          strncpy(_f->forecast[i].weatherIcon, doc["list"][i]["weather"][0]["icon"],
sizeof(_f->forecast[i].weatherIcon) - 1);
          _f->forecast[i].clouds = doc["list"][i]["clouds"]["all"];
          _f->forecast[i].humidity = doc["list"][i]["main"]["humidity"];
          _f->forecast[i].probability = (float)(doc["list"][i]["pop"]) * 100;
          _f->forecast[i].weatherId = doc["list"][i]["weather"][0]["id"];
          _f->forecast[i].pressureGnd = doc["list"][i]["main"]["grnd_level"];
          _f->forecast[i].visibility = doc["list"][i]["visibility"];
          _f->forecast[i].pressure = doc["list"][i]["main"]["pressure"];
          _f->forecast[i].windDir = doc["list"][i]["wind"]["deg"];
          _f->forecast[i].minTemp = doc["list"][i]["main"]["temp_min"];
          _f->forecast[i].temp = doc["list"][i]["main"]["temp"];
          _f->forecast[i].maxTemp = doc["list"][i]["main"]["temp_max"];
          _f->forecast[i].feelsLike = doc["list"][i]["main"]["feels_like"];
          _f->forecast[i].windSpeed = doc["list"][i]["wind"]["speed"];
          _f->forecast[i].windGust = doc["list"][i]["wind"]["gust"];
          _f->forecast[i].rain = doc["list"][i]["rain"]["3h"];
          _f->forecast[i].snow = doc["list"][i]["snow"]["3h"];
          _f->forecast[i].timestamp = doc["list"][i]["dt"];
        }
```

Slika 4.53. Dio koda za dohvaćanje vremenske prognoze s Interneta

Potpuni kod za dohvaćanje vremenske prognoze, raščlambu podataka i izračunavanje maksimalnih, srednjih i minimalnih vrijednosti nalazi se u prilogu 11.

```
// Struct for one element of 5 days/3h forecast
struct forecastWeatherHandle
{
  char weatherDesc[50];
  char weatherIcon[4];
  uint8_t clouds;
  uint8_t humidity;
  uint8_t probability;
  uint16_t weatherId;
  uint16_t pressureGnd;
  uint16_t visibility;
  uint16_t pressure;
  uint16_t windDir;
  float temp;
  float maxTemp;
  float minTemp;
  float feelsLike;
  float windSpeed;
  float windGust;
  float rain;
  float snow;
  uint32_t timestamp;
};
```

```
// Struct for 5 days/3h forecast
struct forecastListHandle
{
  uint8_t startElement[7];
  uint8_t numberOfData;
  uint8_t shiftDay;
  struct forecastWeatherHandle forecast[45];
};
```

Slika 4.54. Struktura podataka za pohranu podataka vremenske prognoze

Na kraju, te je podatke potrebno ispisati na zaslon, a kod koji to izvodi nalazi se u prilogu, prilog 12.

4.5.2. Vanjska jedinica

Prilikom pokretanja vanjske jedinice, prvo se inicijalizira HAL programski sloj, podešavaju se izvori takta i njihovi djelitelji, inicijalizira se periferija mikroupravljača, zatim se inicijalizira i testira LCD i nakon čega se inicijalizira i testira dostupnost svakog od senzora na I2C sabirnici i radijskog modula na SPI sabirnici, te ukoliko dođe do problema, ispisuje se greška na LCD zaslonu i obustavlja se izvođenje programa (slika 4.55 i 4.56).

```
/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_I2C1_Init();
MX_RTC_Init();
MX_SPI1_Init();
MX_USART2_UART_Init();
MX_ADC_Init();
```

Slika 4.55. Inicijalizacija sustava i periferije mikroupravljača

```
// Init LCD Driver
glassLCD_Begin();

// Test LCD (and wait for debugger to connect)
glassLCD_WriteData((char*) lcdTest);
glassLCD_SetDot(0b11111111);
glassLCD_WriteArrow(0b11111111);
glassLCD_Update();
HAL_Delay(2000);

// Dummy ADC readings to calibrate ADC and remove ranks
getADC(ADC_CHANNEL_0);
getADC(ADC_CHANNEL_1);
getADC(ADC_CHANNEL_4);

// Init Pressure & Temp sensor
if (!BMP180_Init()) writeError(BMP180_ERROR, DEEP_SLEEP);

// Init Humidity & Temp sensor
if (!SHT21_Init()) writeError(SHT21_ERROR, DEEP_SLEEP);

// Init Si1147 Sensor for ALS and UV
if (!Si1147_Init()) writeError(SI1147_ERROR, DEEP_SLEEP);

RF24_init(NRF24_CE_GPIO_Port, NRF24_CE_Pin, NRF24_CSN_GPIO_Port, NRF24_CSN_Pin);
if (!RF24_begin()) writeError(NRF24_ERROR, DEEP_SLEEP);

// Enable UV meas.
Si1147_SetUV();

// Set time on RTC
RTC_SetTime(1609459200);

// Setup RF communication (speed, RF Channel, RF Power, rtc)
communication_Setup();
```

Slika 4.56. Inicijalizacija senzora koji se nalaze na I2C sabirnici i radijskog modula

Nakon toga je potrebno podesiti radijsku komunikaciju (slika 4.57), a parametri koji se podešavaju su frekvencija, odnosno kanal, brzina, snaga, adresu prijama i adresu slanja podataka, te postavke za potvrdu dostave paketa). Bitno je da te postavke budu identične, osim adresa prijama i slanja koje moraju biti obrnute.

```
uint64_t addr[2] = {0x65646F4E31, 0x65646F4E32};
const char syncStr[] = {"SYNC %3d"};

void communication_Setup()
{
    RF24_setAutoAck(1);
    RF24_enableAckPayload();
    RF24_setChannel(0);
    RF24_setDataRate(RF24_250KBPS);
    RF24_setPALevel(RF24_PA_MAX, 1);
    RF24_openWritingPipe(addr[0]);
    RF24_openReadingPipe(1, addr[1]);
    RF24_stopListening();
}
```

Slika 4.57. Podešavanje parametara radijske komunikacije

Zatim počinje proces sinkronizacije u kojem se vrši usklađivanje satova na principu slanja EPOCH/POSIX sata, te se ujedno šalje vrijeme slanja izmjerenih podataka u istome obliku (slika 4.58). Da bi se paket raspoznavao od ostalih, na početku se nalazi zaglavlje koje označava što se nalazi unutar paketa (izgled paketa za sinkronizaciju vidljiv je na slici 4.59).

```
uint8_t communication_Transmit(void* _transmitBuffer, uint8_t _txSize, uint8_t* _receiveBuffer)
{
    uint8_t _succ = 0;
    RF24_write(_transmitBuffer, _txSize, 0);
    if (RF24_isAckPayloadAvailable())
    {
        while (RF24_available(NULL))
        {
            RF24_read(_receiveBuffer, 32);
        }
        _succ = 1;
    }
    return _succ;
}
```

Slika 4.58. Programski kod za slanje i dohvaćanje paketa

```
if (syncSuccess)
{
  struct tm hTime;

  RTC_SetTime(syncStruct.myEpoch);
  RTC_SetAlarmEpoch(syncStruct.sendEpoch, RTC_ALARMMASK_DATEWEEKDAY);
  sendInterval = syncStruct.sendEpoch - syncStruct.myEpoch;
  firstTimeSync = 1;

  glassLCD_WriteData("SYNC OK");
  glassLCD_Update();
  HAL_Delay(1000);

  hTime = RTC_EpochToHuman(syncStruct.myEpoch);
  sprintf(lcdTemp, " %2d%02d%02d", hTime.tm_hour, hTime.tm_min, hTime.tm_sec);
  glassLCD_WriteData(lcdTemp);
  glassLCD_SetDot(0b00101000);
  glassLCD_Update();
  HAL_Delay(1000);
}
else
{
  sendInterval = 600;
  RTC_SetAlarmEpoch(RTC_GetTime() + sendInterval, RTC_ALARMMASK_DATEWEEKDAY);
  glassLCD_WriteData("NO SYNC");
  glassLCD_Update();
  HAL_Delay(1000);
}
```

Slika 4.59. Postupak podešavanja sata i sami sinkronizacijski

Na kraju se vrši početno mjerenje podataka koji će biti ispisani na LCD zaslonu, koje će korisnik pritiskom na tipku moći odabirati. Slika 4.60. prikazuje kod koji omogućava da korisnik bira koji će podatak biti ispisan na LCD zaslonu.

```
uint8_t lcdDot = 0;
uint8_t lcdArrow = 0;
if (k == 0)
{
  int16_t t = round(currentWeatherData.tempSHT * 10);
  int16_t h = round(currentWeatherData.humidity * 10);
  sprintf(lcdTemp, "%3d%01d %2d%01d", t / 10, abs(t % 10), abs(h / 10), abs(h % 10));
  lcdDot = 0b00100010;
  lcdArrow = 0b10000000;
}
if (k == 1)
{
  uint16_t p = round(currentWeatherData.pressure * 10);
  sprintf(lcdTemp, "%4d%1d", abs(p / 10), abs(p % 10));
  lcdDot = 0b00010000;
  lcdArrow = 0b01000000;
}
if (k == 2)
{
  int16_t uv = (currentWeatherData.uv * 10);
  int16_t vis = currentWeatherData.light;
  sprintf(lcdTemp, "%2d%1d %4d", abs(uv / 10), abs(uv % 10), vis);
  lcdDot = 0b01000000;
  lcdArrow = 0b00100000;
}
if (k == 3)
{
  int energyJ = round(currentWeatherData.solarJ * 10);
  sprintf(lcdTemp, "%2d%1d %4d", abs(energyJ / 10), abs(energyJ % 10), (int) (currentWeatherData.solarW));
  lcdDot = 0b01000000;
  lcdArrow = 0b00010000;
}
if (k == 4)
{
  int wind = round(currentWeatherData.windSpeed * 10);
  sprintf(lcdTemp, "%3d%1d", abs(wind / 10), abs(wind % 10));
  lcdDot = 0b00100000;
  lcdArrow = 0b00001000;
}
if (k == 5)
{
  sprintf(lcdTemp, "%3s %3d", windStr[(int) ((currentWeatherData.windDir / 22.5) + 0.5) % 16], currentWeatherData.windDir);
  lcdArrow = 0b00000100;
}
if (k == 6)
{
  uint16_t batt = round((currentWeatherData.battery) * 100);
  struct tm t = RTC_EpochToHuman(RTC_GetTime());
  sprintf(lcdTemp, "%2d%02d %1d%02d", t.tm_hour, t.tm_min, batt / 100, abs(batt % 100));
  lcdArrow = 0b00000010;
  lcdDot = 0b01000100;
}
```

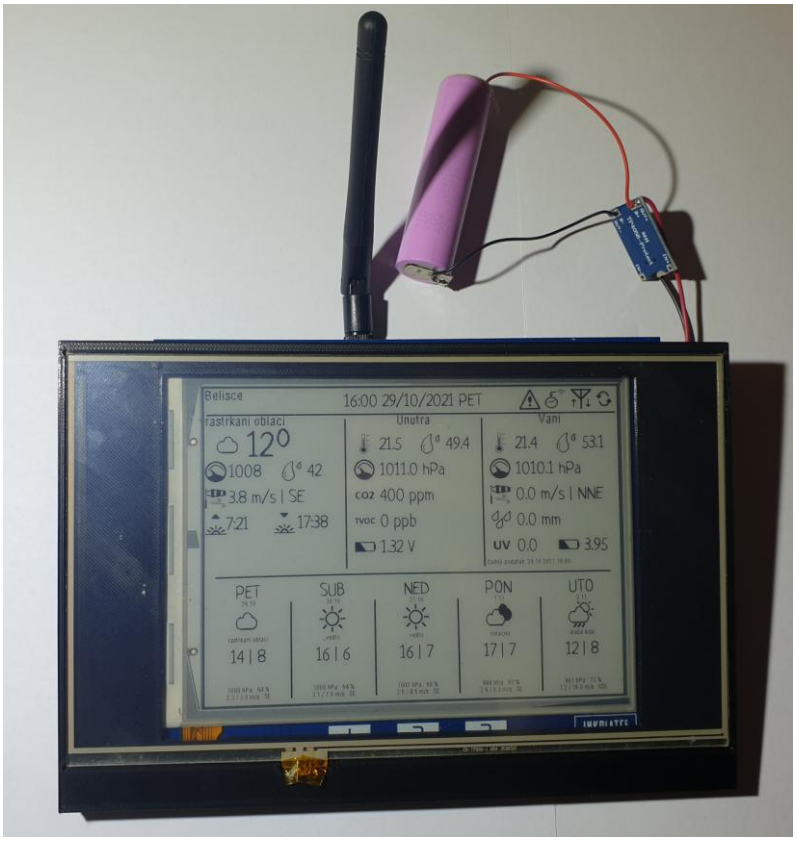
Slika 4.60. Izbor podatka koji će biti ispisan na LCD zaslonu

Nakon toga mikroupravljač odlazi u režim niske potrošnje energije i ostaje u njemu sve dok korisnik ne pritisne tipku ili se dogodi RTC alarm koji daje do znanja da mikroupravljaču da se izmjere novi podaci koji će biti poslani unutrašnjoj jedinici i da se očekuje novo usklađivanje satova.

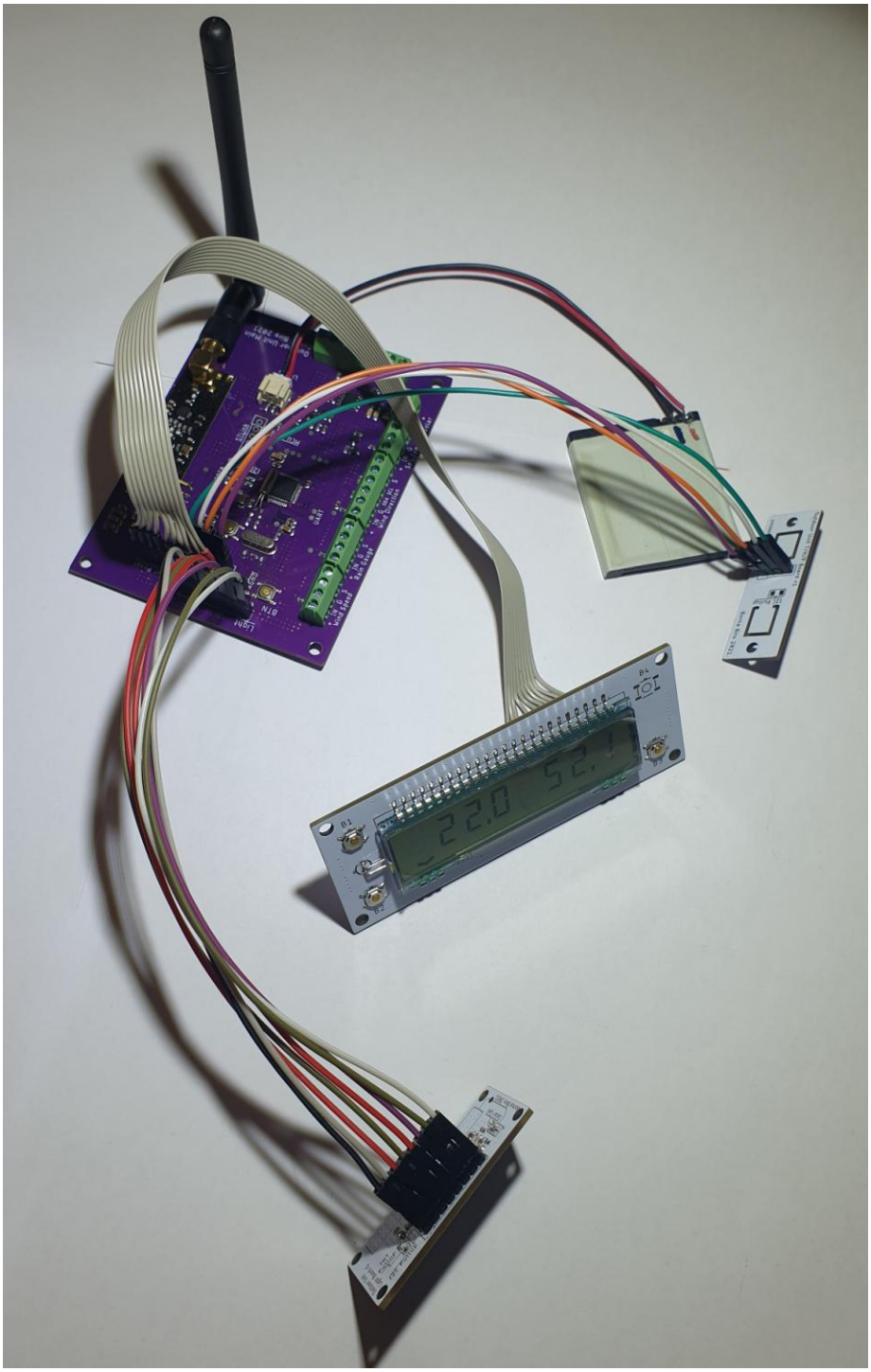
Glavni programski kod za unutrašnju i vanjsku jedinicu (bez vanjskih biblioteka) se nalazi u prilogu, prilog 13 i prilog 14.

5. REZULTATI

Jedan od bitnih kriterija rada ove meteorološke stanice jest da i unutrašnja i vanjska jedinica moraju raditi s vrlo malom potrošnjom energije da bi se mogle napajati pomoću baterije solarnog panela. Da bi se provjerila uspješnost tog kriterija, potrebno je izmjeriti struju pojedine jedinice. Struja se mjeri na način da se ampermetar postavi između baterije i jedinice, s tim da solarni panel nije spojen kako ne bi utjecao mjerenje. Rezultat jest da vanjska jedinica dok je u režimu niske potrošnje energije (u kojem će biti većinu vremena) troši 33.6 µA (od kojih 3.5 µA troši sami mikroupravljač), dok unutrašnja jedinica troši 4.8 mA. Razlog visoke potrošnje jest što unutrašnja jedinica ne koristi tzv. engl. *deep sleep* već koristi engl. *light sleep* koji ima veću potrošnju energije, no zato svi podaci ostaju zadržani u RAMu kada jedinica se prebaci u režim niske potrošnje. Dok radi, bez WiFi veze, tada troši oko 50 mA, a dok postoji WiFI veza, srednja potrošnja je od 120 mA do 150 mA. Točne iznose je moguće samo dobiti ukoliko se koristi specijalizirana oprema za mjerenje potrošnje IoT uređaja. Slika gotove unutrašnje jedinice vidljiva je na slici 5.1, a vanjske na slici 5.2.



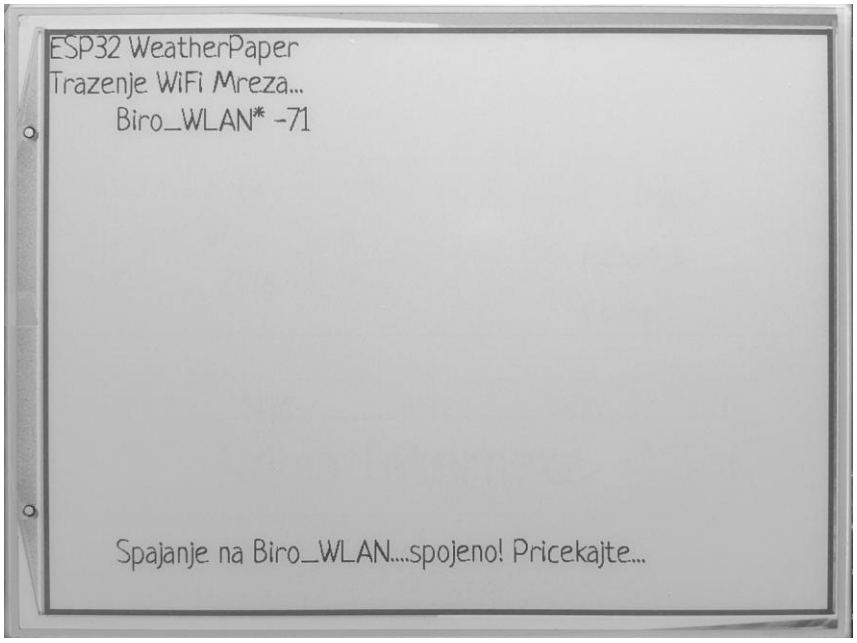
Slika 5.1. Unutrašnja jedinica



Slika 5.2. Vanjska jedinica

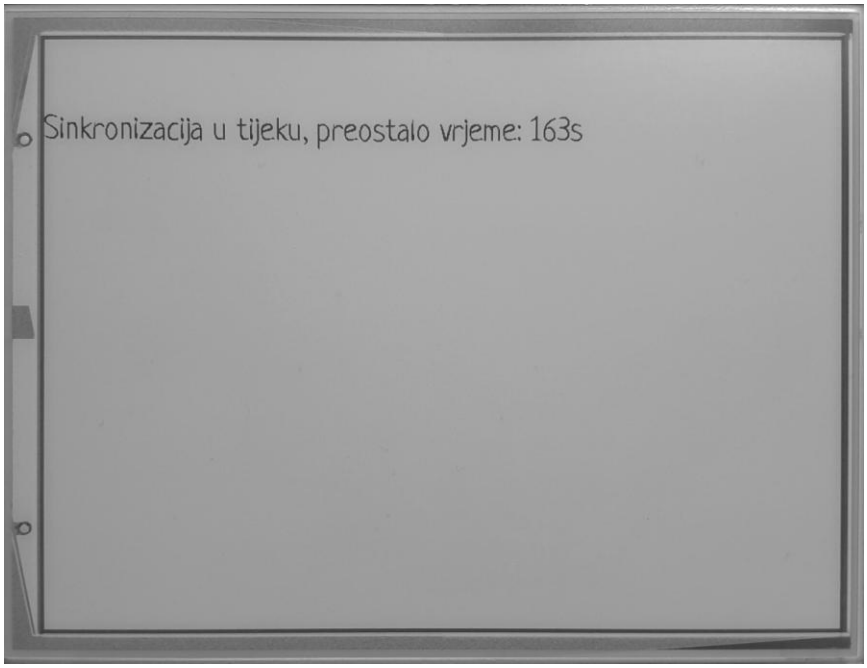
5.1. Rukovanje unutrašnjom jedinicom

Unutrašnja jedinica se aktivira pomakom tipke za aktivaciju napajanja prema gore. Nakon što se unutrašnja jedinica aktivira i pokrene, traže se dostupne WiFi mreže (slika 5.3), te ukoliko je dostupna predefinirana mreža, vrši se spajanje i preuzimanje vremenske prognoze i točnoga sata.



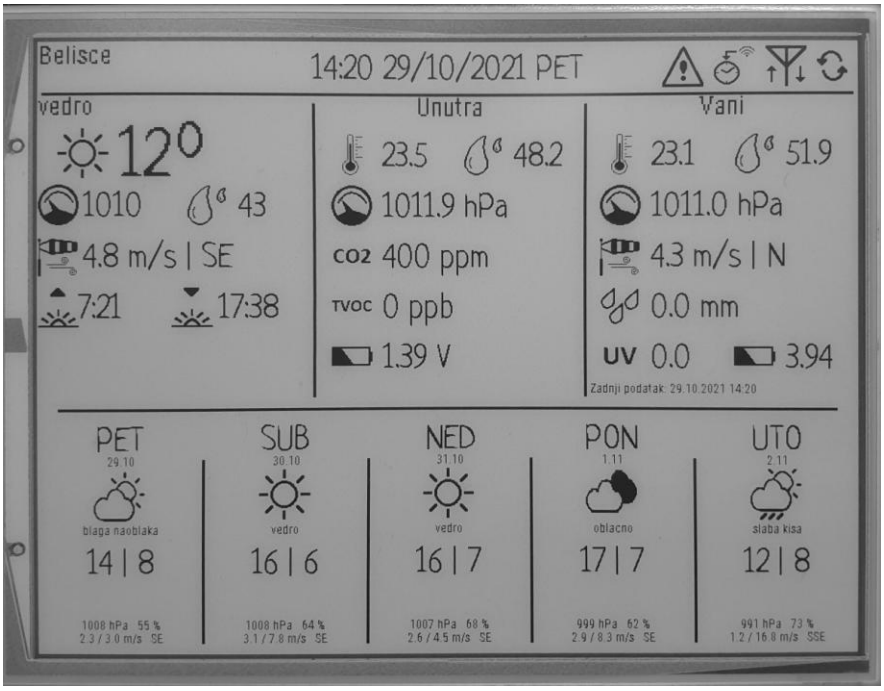
Slika 5.3. Spajanje unutrašnje jedinice na WiFi mrežu

Nakon uspješnog dohvaćanja sata i vremenske prognoze, počinje proces sinkronizacije gdje se traži signal vanjske jedinice u trajanju od 3 minute (slika 5.4), nakon čega se ukoliko je signal pronađen, šalje odgovor u obliku EPOCH / POSIX podatka trenutnog sata i vremena buđenja i slanja izmjerenih podatka.



Slika 5.4. Sinkronizacija unutrašnje jedinice s vanjskom

Ukoliko signal nije pronađen, unutrašnja jedinica će raditi no bez dostupnih podataka s vanjske jedinice. Nakon toga je vidljiv početni zaslon (slika 5.5).

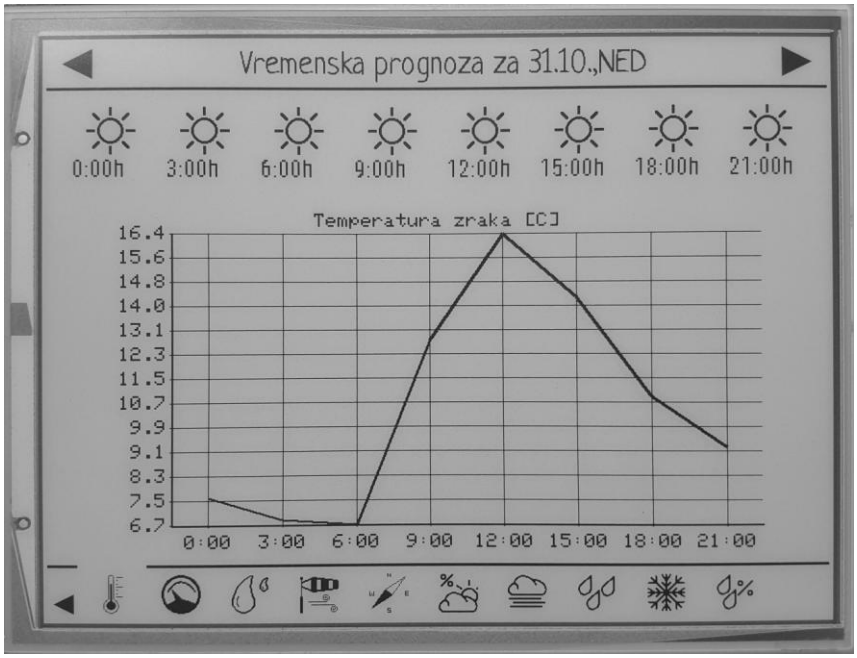


Slika 5.5. Izgled početnog zaslona

Početni zaslon je podijeljen u više dijelova. U donjem dijelu zaslona vidljiva je vremenska prognoza za narednih 5 dana. Za svaki dan se prikazuje datum i dan. Ispod toga se nalazi opis vremena s ikonicom u 15 h toga dana, a nakon toga minimalna i maksimalna temperatura zraka, srednja vrijednost tlaka zraka, srednja vrijednost relativne vlažnosti zraka, srednja vrijednost brzine vjetra, maksimalna brzina vjetra i srednji smjer vjetra toga dana (slika 5.6).

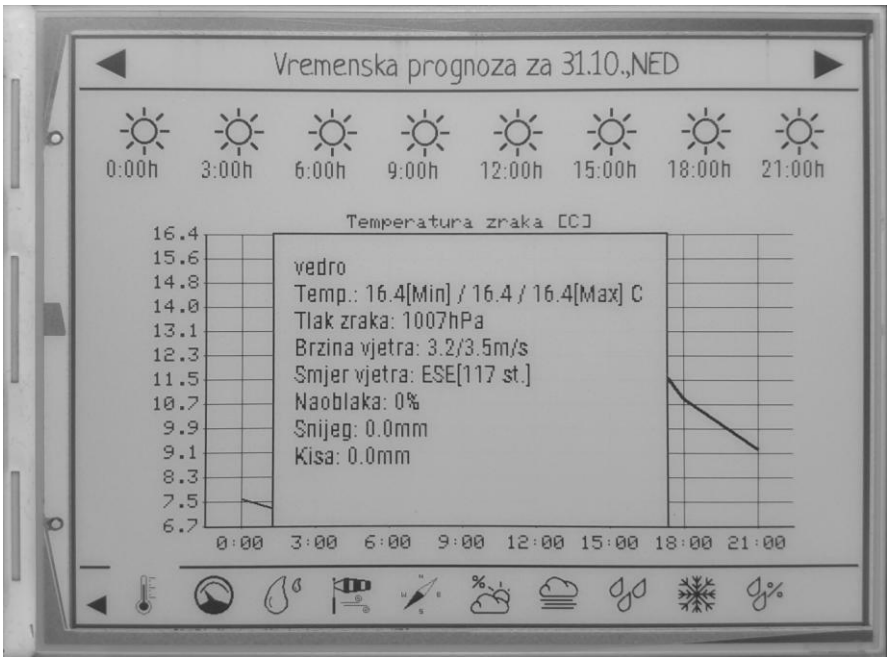
Slika 5.6. Prikaz prognoze jednoga dana

Dodirom (odabirom) jednog od dana, prikazuje se novi prikaz s odabranim danom gdje se detaljnije mogu vidjeti parametri prognoze izabranog dana (slika 5.7).



Slika 5.7. Detaljan prikaz prognoze jednoga dana

Ukoliko se dodirne jedan od vremenskih intervala, prikazuje se mali prozor u kojem pišu svi podaci o tome danu u zadanome vremenskome intervalu (slika 5.8).

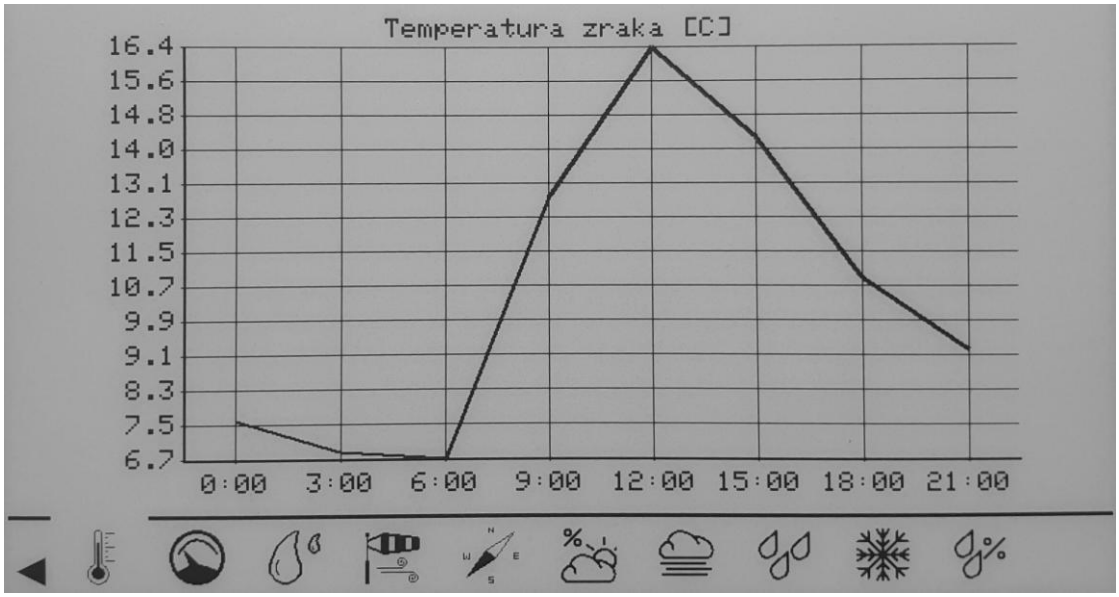


Slika 5.8. Detaljan prikaz svih dostupnih podataka

Ukoliko se želi prebaciti na slijedeći dan, to se može učiniti dodirom na strelice u gornjem lijevom u desnom kutu (ukoliko ne postoje, tada znači da podaci za slijedeći ili prethodni dan ne postoje).

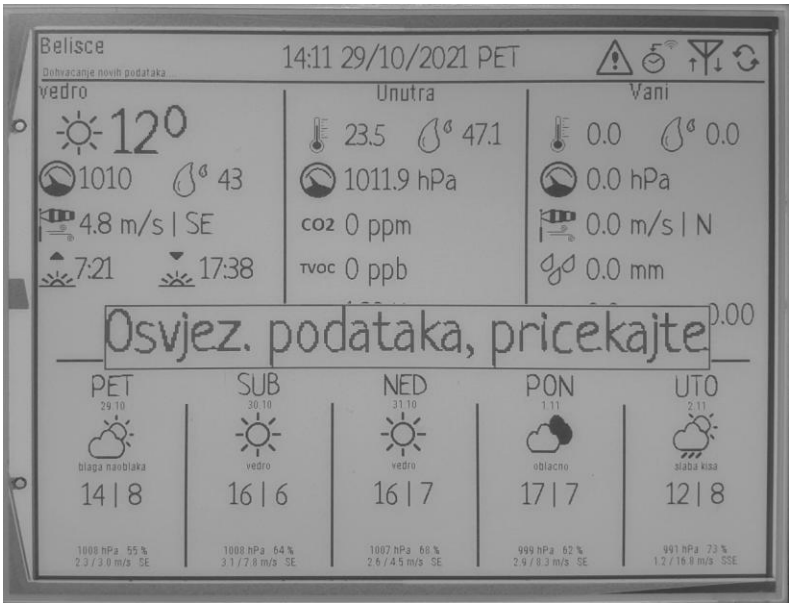
Dodirom na ikonice dolje (slika 5.9), pojavljuju se grafovi koji pokazuju kako se zadana veličina mijenja po danu (slika 5.10).

Slika 5.9. Ikonice za prikaz grafova



Slika 5.10. Prikaz grafa zadane veličine

Ukoliko se poželi vratit na početni zaslon, to se može učiniti dodirom na strelicu u donjem lijevom kutu. U gornjem dijelu početnoga zaslona mogu se vidjeti za koji grad se trenutno prikazuju podaci, zadnje vrijeme osvježavanja podataka, da li postoje upozorenja na opasne vremenske nepogode i tipka za prisilno osvježavanje podataka vremenske prognoze. Dodirom na tu tipku, pojaviti će se mali prozor gdje će se obavijestiti korisnika da se novi podaci dohvaćaju (slika 5.11).

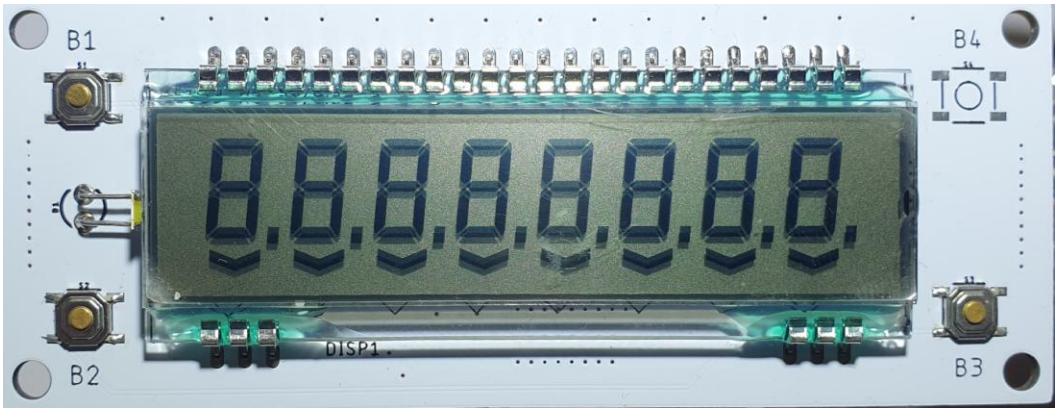


Slika 5.11. Osvježavanje podataka vremenske prognoze

Ovo što se na početnome zaslonu može vidjeti su još podaci o trenutnim uvjetima vani dohvaćeni s Interneta, zatim podaci izmjereni od strane unutrašnje jedinice, te zadnji podaci izmjereni od strane vanjske jedinice. Unutrašnja jedinica će sama svakih 30 minuta dohvatiti nove podatke o vremenskoj prognozi. Simboli u gornjem lijevom kutu na slici imaju značenja (od lijeva prema desno) upozorenja na iznimno loše vrijeme, tipka za sinkronizaciju sata unutrašnje jedinice putem NTP protokola, tipka za sinkronizaciju s vanjskom jedinicom i tipka za osvježavanje podataka (bez spremanja podataka u memoriju).

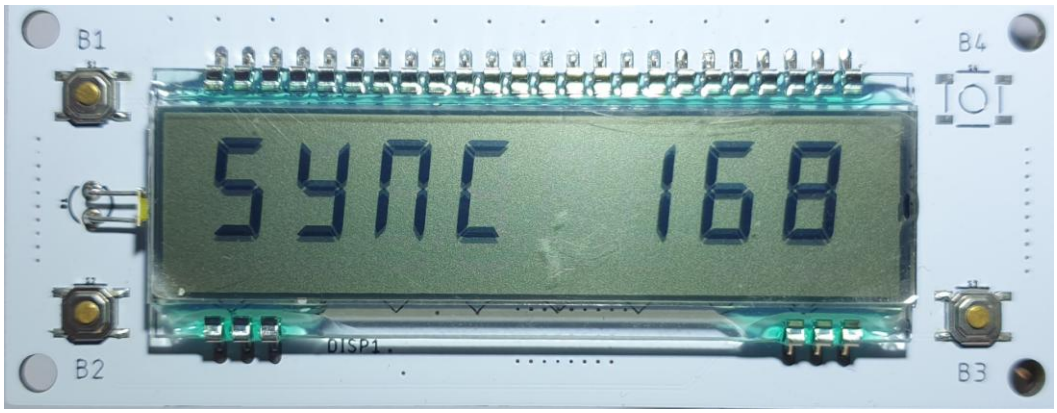
5.2. Rukovanje vanjskom jedinicom

Nakon što se napajanje dovede na vanjsku jedinicu putem baterije ili solarnoga panela, na LCD zaslonu se ispisuje testni ispis na kojemu moraju biti prikazani svi segmenti LCD zaslona (slika 5.12) da bi se korisnik uvjerio kako nije došlo do oštećenja zaslona.



Slika 5.12. Test ispis LCD zaslona

Nakon toga započinje proces sinkronizacije s unutrašnjom jedinicom, gdje se na LCD zaslonu ispisuje riječ „SYNC“ i preostalo vrijeme u sekundama (slika 5.13).



Slika 5.13. Sinkronizacija

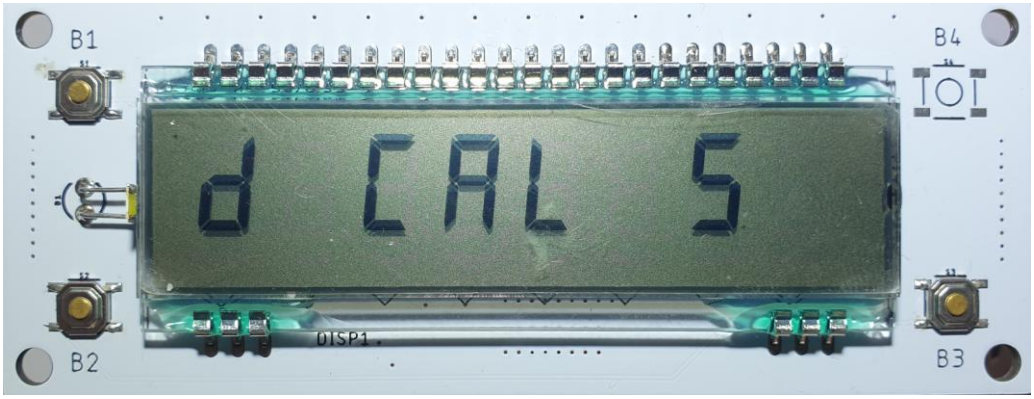
Ukoliko se ne postigne sinkronizacija, nakon isteka vremena se na zaslonu ispisuje „NO SYNC“ (slika 5.14), nakon čega uređaj će nastaviti raditi, ali neće slati podatke unutrašnjoj jedinici, već će ih samo ispisivati na LCD zaslonu.

Slika 5.14. Neuspješna sinkronizacija

Ukoliko se želi ponovno pokušati izvršiti sinkronizacija, potrebno je pritisnuti tipku RST na glavnoj ploči. Ukoliko je sinkronizacija uspješna, obavijestit će se korisnika s porukom „SYNC OK“ (slika 5.15), nakon čega će se ispisati trenutan sat (slika 5.16) koji je postavljen na vanjsku jedinicu potrem sinkronizacije, a nedugo nakon toga će se ispisati „ D CAL x“ (slika 5.17), gdje x predstavlja preostale sekunde kalibracije To znači da se izvršava proces kalibracije senzora smjera vjetra, koji u tim trenutcima mora biti orijentiran prema sjeveru.

Slika 5.15. Uspješna sinkronizacija

Slika 5.16. Prikaz sata nakon sinkronizacije



Slika 5.17. Kalibracija senzora smjera vjetra

Zatim se vrše mjerenja, ispsuju se mjereni podaci na LCD zaslonu i uređaj prelazi u režim niske potrošnje. Redni broj strelice pokazuje koji se trenutno podatak prikazuje (slika 5.18).

Slika 5.18. Prikaz trenutnog mjerenja (temperatura i vlaga zraka)

Ukoliko se želi listati po mjerenjima, to se može raditi s lijevom tipkom, a ukoliko se želi željeni podatak ponovno izmjeriti, to se može napraviti s desnom tipkom. Treba imati na umu da se ta

dodatna mjerenja ne šalju. Mjerenja idu ovim redom: Temperatura (°C) i vlaga zraka (%), tlak zraka (hPa), UV zračenje i količina svjetlosti (lux), količina sunčevog zračenja u J/cm^2 i W/m^2, brzina vjetra (m/s), smjer vjetra, te trenutno vrijeme i napon baterije u voltima. Kada se podaci trebaju poslati, korisnik će biti obaviješten porukom „SEND“ na LCD zaslonu, gdje će nakon toga biti ispisano koliko je još pokušaja slanja preostalo (slika 5.19).

Slika 5.19. Slanje podataka

6. ZAKLJUČAK

Zadatak je uspješno odrađen, jer meteorološka stanica odrađuje sve što je zadano u opisu zadatka, poput korištenja WiFi veze za dohvaćanje vremenske prognoze, prikaz podataka na e-ink (*e-paper*) zaslonu, mjerenje unutrašnjih parametara poput temperature zraka, relativne vlažnosti zraka, tlaka zraka i CO2 (eCO2). Ono što je učinjeno van zadatka je kompletna vanjska jedinica iz razloga jer je riječ o meteorološkoj postaji, stoga se očekuje se da mjere i neki vanjski parametri. Pošto je unutrašnja jedinica napravljena na principu *low power* uređaja, potrebno je bio napraviti da i vanjska jedinica bude isto. Ono što je još dodano van zadataka jest da se obje jedinice napajanju preko baterije i solarnog panela, što omogućava da se uređaji koriste bez da ih je potrebno s vremena na vrijeme puniti. Moguće je izvesti niz unaprjeđenja poput kućišta vanjske jedinice, dodavanje senzora za količinu padavina, temperaturu tla, senzor grmljavine itd, kao i neka programska unaprjeđenja poput dodavanja pravog korisničkog grafičkog sučelja pomoću kojeg bi korisnik mogao mijenjati sve parametre sustava (poput koliko često da se mjere i šalju podaci, izmjena i unos API ključa, odabir WiFi veze i unos lozinke pomoću zaslona osjetljivog na dodir itd), te neke optimizacije za povećanje brzine rada i smanjenja koda.

LITERATURA

[1] http://www.vrijeme.net/meteo-pojave/meteorologija-kao-znanost-clanak-9, pristupljeno dana 29.6.2019.

[2] http://www.radiosamobor.hr/2014/07/30/trazi-se-lokacija-i-motritelj-za-klimatolosku-postaju-u-samoboru/, pristupljeno dana 29.6.2019.

[3] https://scied.ucar.edu/weather-balloons, autor UCAR, pristupljeno dana 29.6.2019.

[4] https://hr.wikipedia.org/wiki/Meteorolo%C5%A1ka_postaja, pristupljeno dana 29.6.2019.

[5] https://meteo.hr/podaci.php?section=podaci_mjerenja&param=satelit, pristupljeno dana 29.6.2019.

[6] https://vrijeme.hr/satelit/MSG3-2000.jpg, pristupljeno dana 29.6.2019.

[7] https://www.ambientweather.com/amws5300.html, pristupljeno dana 29.6.2019.

[8] https://en.wikipedia.org/wiki/Amazon_Kindle#/media/File:Amazon_Kindle_-_Wikipedia.jpg, pristupljeno dana 28.6.2019.

[9] https://www.researchgate.net/figure/Working-principle-of-the-microcapsule-based-electrophoretic-display_fig2_324179552, pristupljeno dana 28.6.2019.

[10] https://pihw.files.wordpress.com/2013/05/2013-05-11-10-00-49.jpg, pristupljeno dana 28.6.2019.

[11] https://www.researchgate.net/figure/Schematic-displays-of-electrowetting-working-principle-and-device-structure-a-In-an_fig2_316238641, pristupljeno dana 28.6.2019.

[12] https://www.waveshare.com/4.3inch-e-paper.htm, pristupljeno dana 28.6.2019.

[13] https://www.waveshare.com/7.5inch-e-Paper-HAT.htm, pristupljeno dana 28.6.2019.

[14] https://www.waveshare.com/6inch-e-Paper-HAT.htm, pristupljeno dana 28.6.2019.

[15] https://www.indiamart.com/proddetail/esp8266-wifi-module-18795392230.html, pristupljeno dana 28.6.2019.

[16] https://store.arduino.cc/arduino-ethernet-shield-without-poe-module, pristupljeno dana 28.6.2019.

[17] https://www.st.com/en/evaluation-tools/nucleo-l053r8.html, pristupljeno dana 25.9.2021.

[18] https://www.elecrow.com/esp32-wrover-i-4mb-spi-flash-4mb-psram-wifi-bt-ble-mcu-module.html, pristupljeno dana 25.9.2021.

[19] https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, pristupljeno dana 25.9.2021.

[20] https://www.st.com/resource/en/datasheet/stm32l073v8.pdf, pristupljeno dana 25.9.2021.

[21] https://morioh.com/p/1a072d5dc8d6, pristupljeno dana 25.9.2021.

[22] https://tech.memoryimprintstudio.com/arduino-with-a-lgdp4535-tft-lcd-touch-screen/, pristupljeno dana 25.9.2021.

[23] https://www.newhavendisplay.com/app_notes/TPcompare.pdf, pristupljeno dana 25.9.2021.

[24] https://www.researchgate.net/figure/Resistive-touchscreen-Source-http-wwwtcide_fig54_321527129, pristupljeno dana 25.9.2021.

[25] https://www.embedded.com/getting-in-touch-with-capacitance-sensor-algorithms/, pristupljeno dana 25.9.2021.

[26] https://www.peetbros.com/shop/category.aspx?catid=35, pristupljeno dana 25.9.2021.

[27] https://e-radionica.com/hr/bme280-breakout-made-by-e-radionica.html, pristupljeno dana 25.9.2021.

[28] https://e-radionica.com/hr/senzor-kvalitete-zraka-sgp30-2-5k-made-by-e-radionica-com.html, pristupljeno dana 25.9.2021.

[29] https://e-radionica.com/hr/433mhz-rf-odasiljac-i-prijemnik.html, pristupljeno dana 25.9.2021.

[30] https://www.thethingsnetwork.org/docs/lorawan/limitations/, pristupljeno dana 25.9.2021.

[31] https://e-radionica.com/hr/2-4ghz-nrf24l01-odasiljac-prijemnik-s-pojacalom-antenom.html, pristupljeno dana 25.9.2021.

[32] https://eneloop101.com/wp-content/uploads/2017/02/HR-3UTGB.pdf, pristupljeno dana 25.9.2021.

[33] https://www.powerstream.com/p/us18650vtc5-vtc5.pdf, pristupljeno dana 25.9.2021.

[34] https://e-radionica.com/hr/18650-baterija-samsung-inr18650-29e-2900mah.html, pristupljeno dana 25.9.2021.

[35] https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf, pristupljeno dana 25.9.2021.

[36] http://essentialscrap.com/eink/electronics.html, pristupljeno dana 28.6.2019.

[37] https://cdn.hackaday.io/files/1603266859387584/ED060SC7-2.0.pdf, pristupljeno dana 28.6.2019.

[38] https://www.circuitvalley.com/2017/03/eink-e-ink-kindle-dispaly-clock-reuse-weather-monitor-temperature-eink-driver-arduino.html, pristupljeno dana 28.6.2019.

[39] https://www.freeimages.com/photo/old-car-2-1508187, pristupljeno dana 25.9.2021.

[40] https://www.ti.com/lit/ds/symlink/tsc2046.pdf, pristupljeno dana 25.9.2021.

[41] https://www.ti.com/lit/ds/symlink/tps7a26.pdf, pristupljeno dana 25.9.2021.

[42] https://www.fondriest.com/media/catalog/product/cache/f40361026fba84c886705ea8cd3376d4/r/m/rm_young_81000_lg.jpg, pristupljeno dana 25.9.2021.

[43] https://ams.com/documents/20143/36005/AS5600_DS000365_5-00.pdf, pristupljeno dana 25.9.2021.

[44] https://cdn.energypal.com/panels/ws-335/energypal-solar-panel-spec-datasheet-waaree-energies-aditya-series-ws-300-350-ws-335.pdf, pristupljeno dana 25.9.2021.

SAŽETAK

Tema ovoga zadataka je bilo osmisliti meteorološku postaju koja kao zaslon ima e-ink (*e-paper*) zaslon vrlo niske potrošnje na kojem će se prikazivati vremenska prognoza i izmjereni podaci poput temperature zraka, relativne vlažnosti zraka, tlak zraka i CO2 (eCO2) u zraku. Rad mora trošiti što je manje moguće energije uz pomoć *low power* režima rada. Rad mora koristit WiFi mrežu za spajanje na Internet da bi se dohvatila vremenska prognoza. Osim toga, rad mora izmjerene podatke pohranjivati na neku vrstu memorije iz koje kasnije te podatke može dohvatiti i prikazati. Poželjno je da se izmjereni podaci postave na Internet, te da ih se isto može po potrebi dohvatiti i prikazati. Dodana je i vanjska jedinica koja mjeri podatke poput temperature zraka, tlaka zraka, relativne vlažnosti zraka, UV zračenje, količinu svjetlosti, količinu sunčevog zračenja, brzinu vjetra i smjer vjetra. Podaci se šalju periodično unutrašnjoj jedinici putem radio veze. Unutrašnja i vanjska jedinica koriste bateriju i solarni panel kao izvor energije.

Ključne riječi: Meteorološka postaja, ESP32, Arduino IDE, STM32, STM32L073, WiFi, niska potrošnja energije, epaper, eink, reverse engineering, nRF24l01, bežična komunikacija, NTP, API servisi, openweathermaps, SGP30, SHT21, BME280, BMP180, Si1147, anemometar, vjetrokaz, UV zračenje, temperatura zraka, relativna vlažnost zraka, tlak zraka, grafički prikaz podataka, senzor sunčevog zračenja

ABSTRACT

The task of his project was to make a weather station that uses e-ink (e-paper) display that has extremely low power consumption for displaying measured data such as air temperature, relative air humidity, atmospheric pressure and concentration of CO2 (eCO2) in the air. The device needs to use as little energy as possible by using low-power mode. The device also has to use WiFi technology in order to connect to the Internet and get the weather forecast data. Also, the device has to store measured data in some kind of memory device from where data can be fetched and displayed. Preferably, the device can send data to the Internet and also can get the same data if needed. There is also added outdoor unit device that measures data such as temperature, relative air humidity, atmospheric pressure, UV radiation, light intensity, irradiated solar power, wind speed and wind direction. Data is transmitted to the indoor unit by using RF communication. Both indoor and outdoor units are using batteries and solar panels as sources of power.

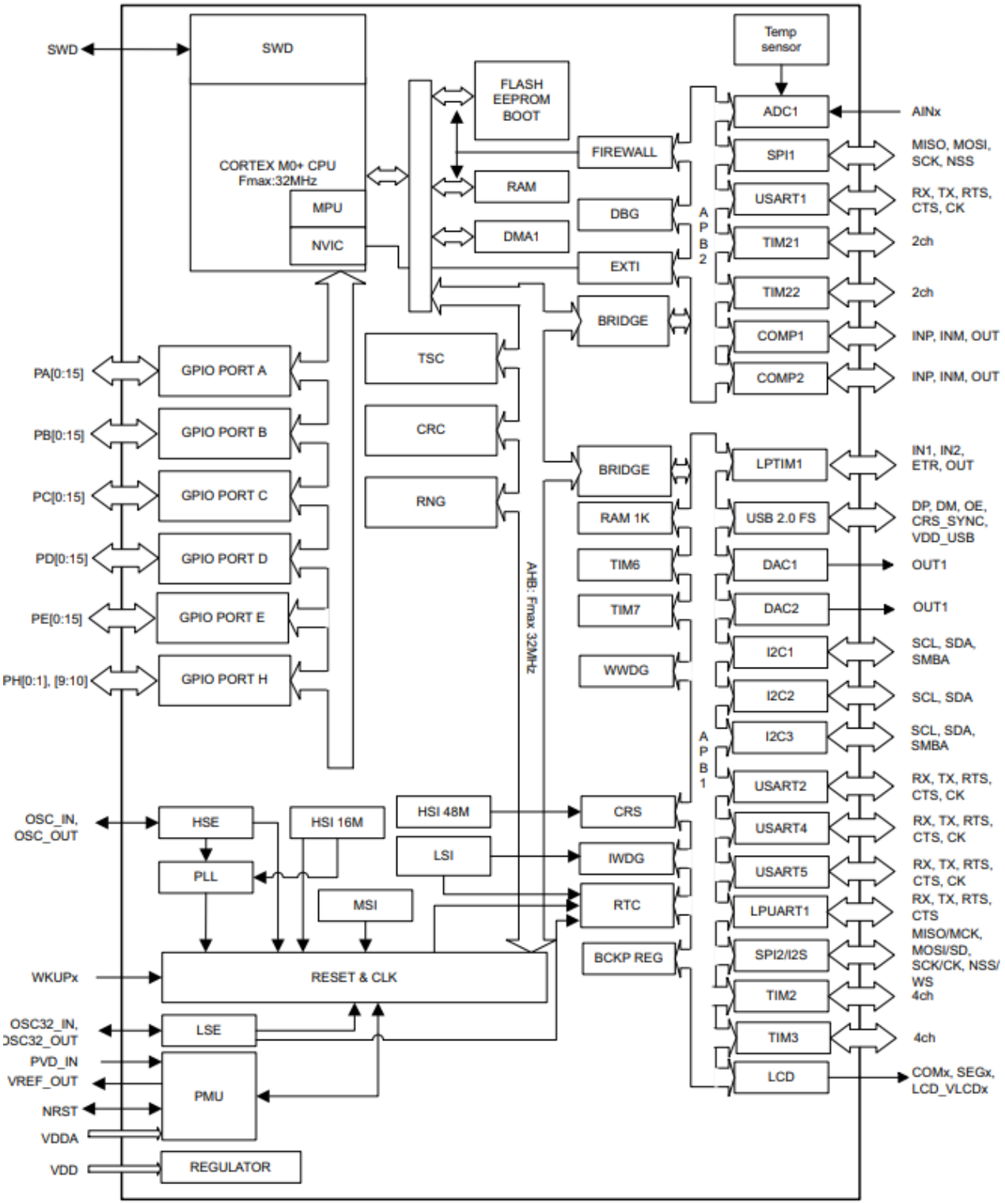
Keywords: Weather station, ESP32, Arduino IDE, STM32, STM32L073, WiFi, low power, epaper, eink, reverse engineering, nRF24l01, wireless communication, NTP, API service, openweathermaps, SGP30, SHT21, BME280, BMP180, Si1147, weather anemometer, weather wind vane, UV radiation, air temperature, relative air humidity, air pressure, graphical display of dana, solar irradiance sensor

ŽIVOTOPIS

Borna Biro rođen je 13.7.1994. u Osijeku. Pohađao je osnovnu školu Ivana Kukuljevića u Belišću i završio ju 2009. godine. Iste godine upisao je Srednju Školu Valpovo u Valpovu, smjer elektrotehničar. Za vrijeme kraja osnovne škole i cijelim dijelom srednje škole je bio član radiokluba Belišće (9A1KDE), no bez položene radioamaterske klase gdje je stekao osnovna znanja radiotehnike i elektronike. Za vrijeme pohađanja srednje škole sudjelovao je na dvije smotre radova, prva je bila državna smotra radova održana 2012. godine u Selcu, na kojoj se predstavio s projektom „Audio Pojačalo Snage“ te je osvojio drugo mjesto. Druga je bila 2013. godine, održana u Zagrebu na FER-u, na koju se predstavio s projektom „Kompenzator jalove snage“. Iste te godine je završio srednju školu, te polagao državnu maturu. Nakon polaganja državne mature, upisao je elektrotehnički fakultet u Osijeku (Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek), gdje je 2017. godine stekao titulu sveučilišni prvostupnik inženjer elektrotehnike. Iste godine je upisao diplomski smjer „Komunikacijske tehnologije“ na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Za vrijeme druge godine studija, nakon izvršene prakse, nastavio je raditi u firmi gdje je bila odrađivana praksa, te gdje je stekao iskustva rada s razvojem elektroničkih proizvoda i sklopova. Krajem druge godine diplomskog studija, 2019. godine osvaja rektorovu nagradu na sveučilištu Josipa Jurja Strossmayera u Osijeku za rad “Vremenska stanica“ iz predmeta primjena mikroupravljačkih sustava.

PRILOG

PRILOG 1

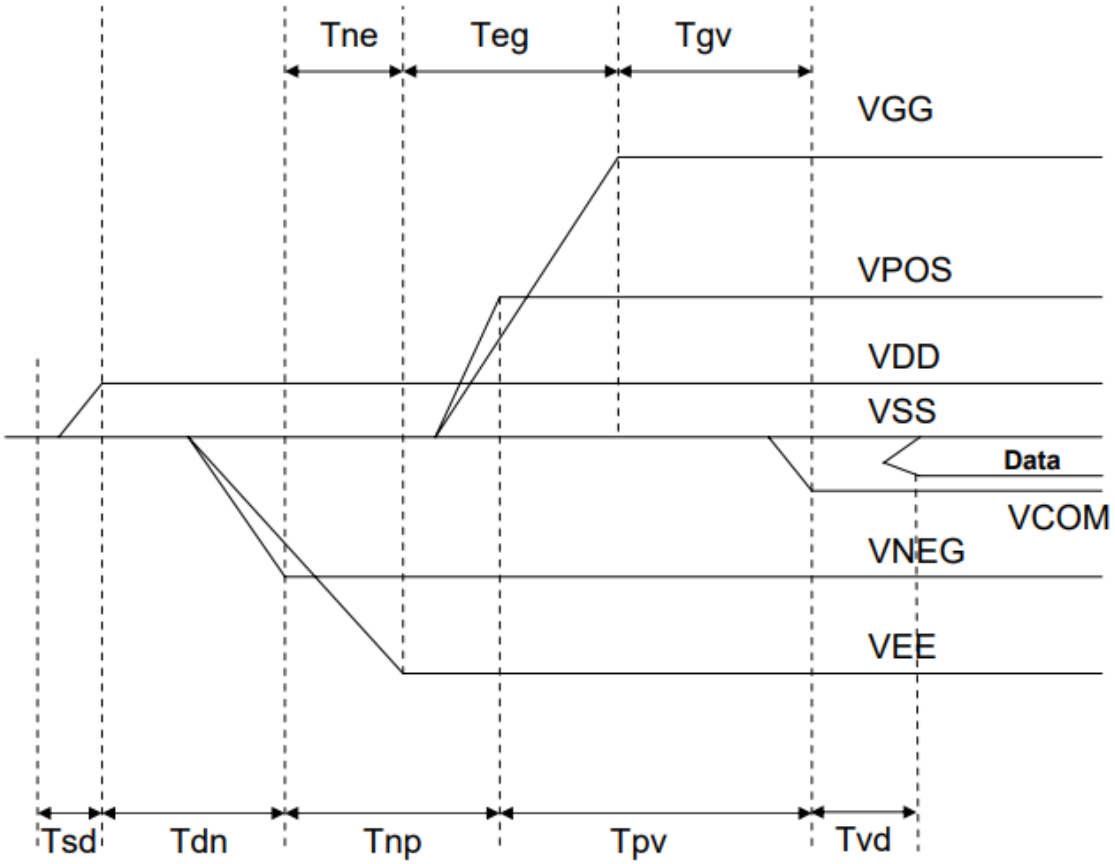


PRILOG 2

Pin #	Signal	Description	Remark
1	VNEG	Negative power supply source driver	
2	VPOS	Positive power supply source driver	
3	VNEG	Negative power supply source driver	
4	VPOS	Positive power supply source driver	
5	VDD	Digital power supply drivers	
6	VSS	Ground	
7	VDD	Digital power supply drivers	
8	VSS	Ground	
9	XCL	Clock source driver	
10	XLE	Latch enable source driver	
11	XOE	Output enable source driver	
12	XSTL	Start pulse source driver	
13	D0	Data signal source driver	
14	D1	Data signal source driver	
15	D2	Data signal source driver	
16	D3	Data signal source driver	
17	D4	Data signal source driver	
18	D5	Data signal source driver	
19	D6	Data signal source driver	
20	D7	Data signal source driver	
21	VCOM	Common connection	
22	SPI_SCL	Serial Data Clock for Flash memory	Note5-1
23	VCOM	Common connection	
24	SPI_SDI	Serial Data Input for Flash memory	Note5-1
25	VGG	Positive power supply gate driver	
26	MODE1	Output mode selection gate driver	
27	VEE	Negative power supply gate driver	
28	CKV	Clock gate driver	
29	VEE	Negative power supply gate driver	
30	SPV	Start pulse gate driver	
31	VSS	Ground	
32	BORDER	Border connection	
33	SPI_NCS	Chip Select for Flash memory	Note5-1
34	SPI_SDO	Serial Data Output for Flash memory	Note5-1

PRILOG 3

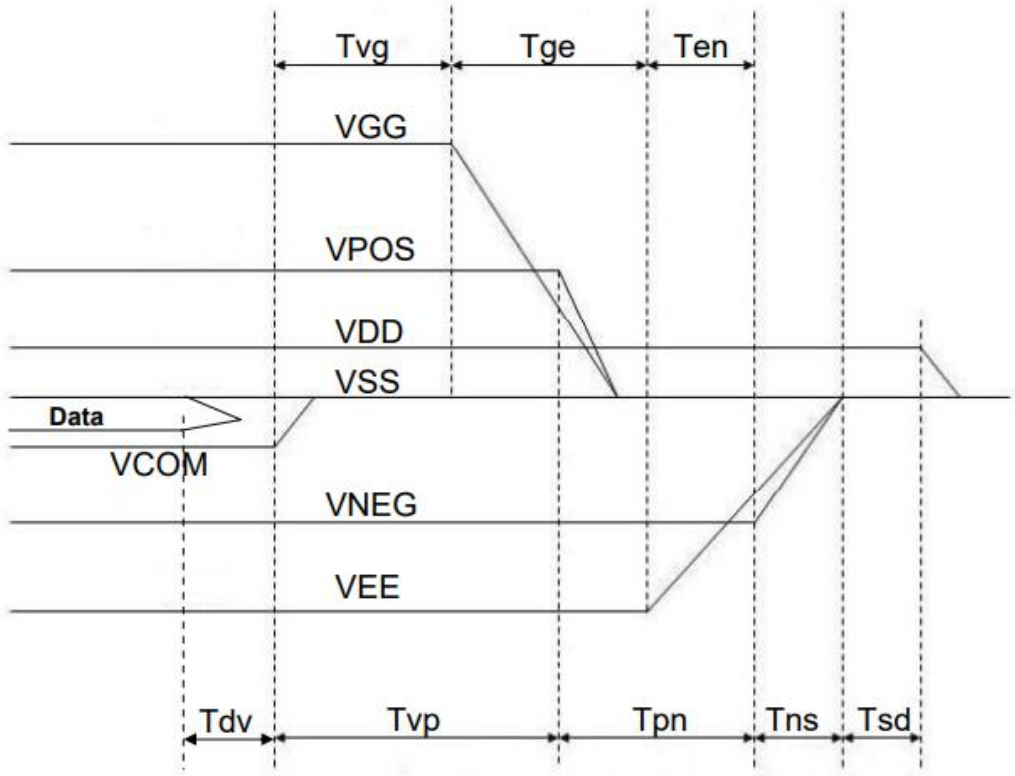
POWER ON



	Min	Max
Tsd	100us	-
Tdn	100us	-
Tnp	1000us	-
Tpv	100us	-
Tvd	100us	-
Tne	0us	-
Teg	1000us	-
Tgv	100us	-

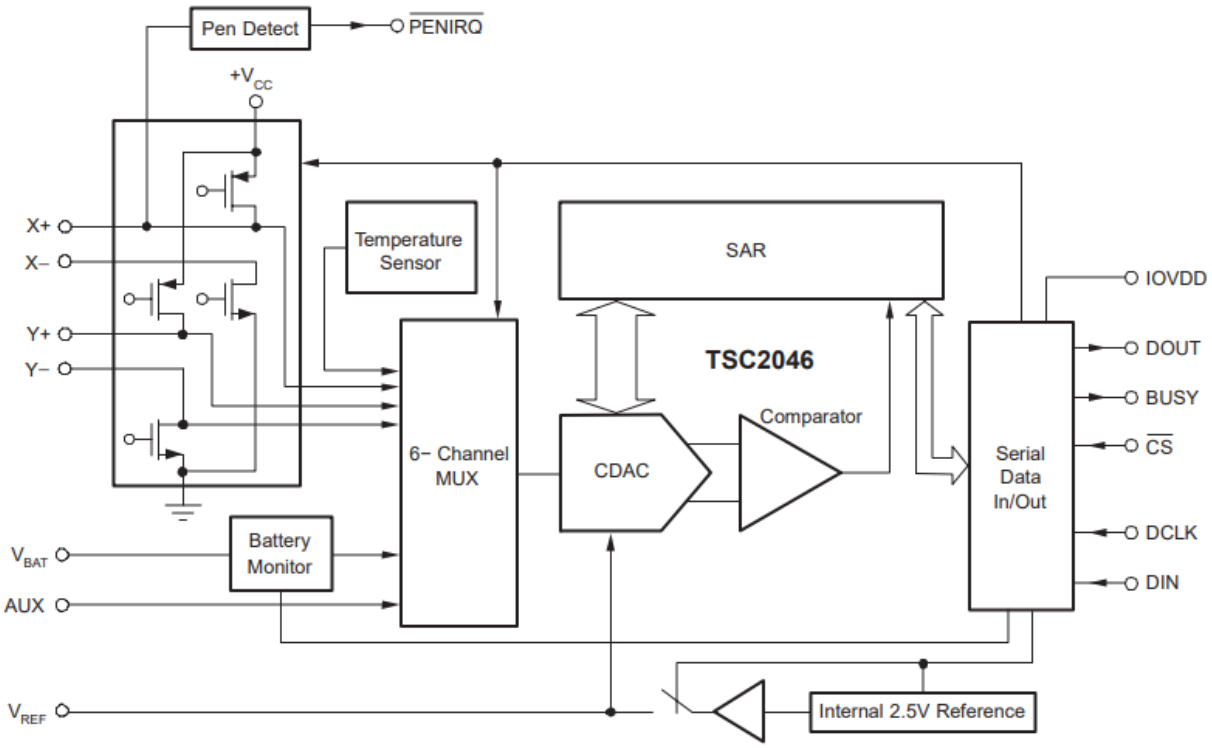
PRILOG 4

POWER DOWN

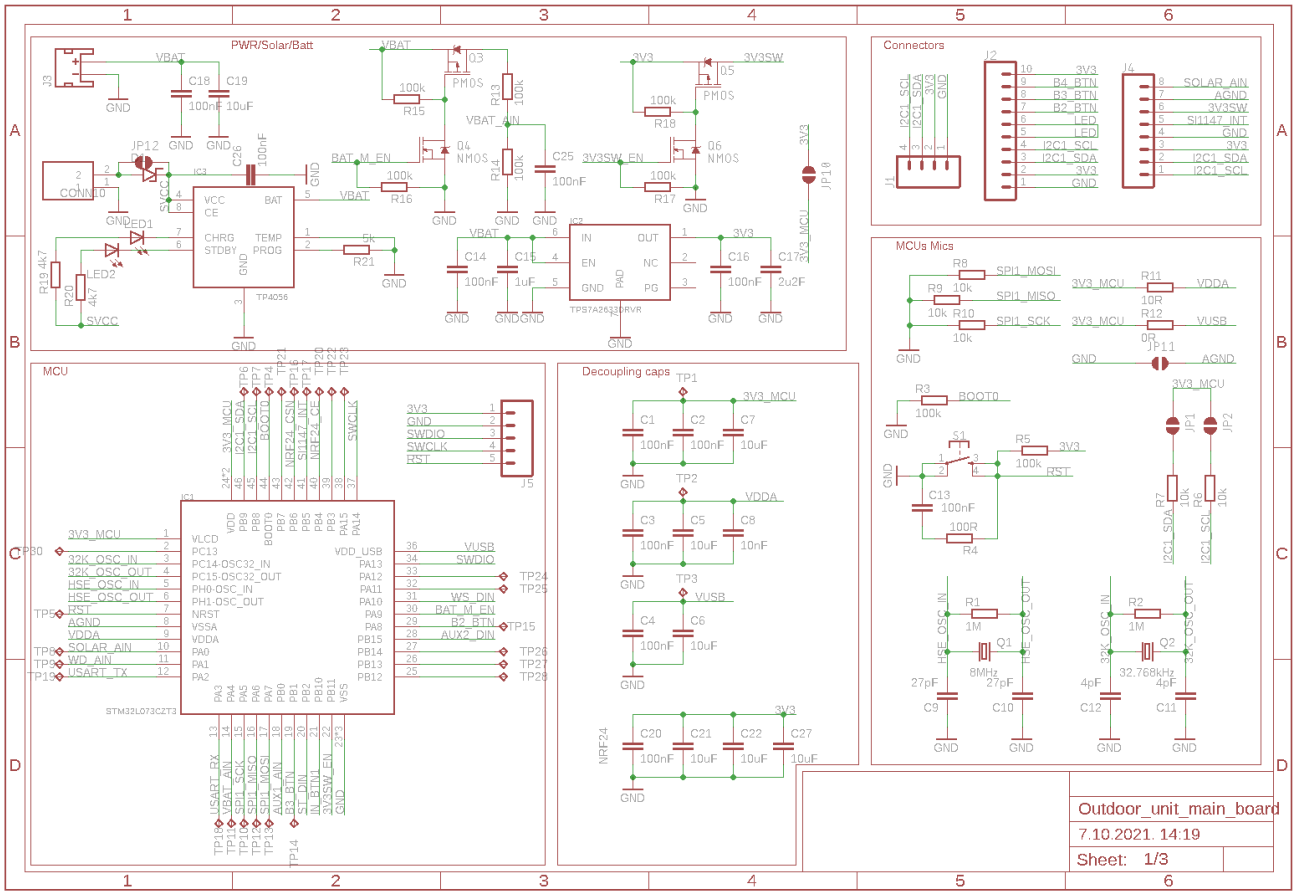


	Min	Max
Tdv	100 μ s	-
Tvp	0 μ s	-
Tpn	0 μ s	-
Tns	-	1000ms
Tsd	100 μ s	-
Tvg	0 μ s	-
Tge	0 μ s	-
Ten	0 μ s	-

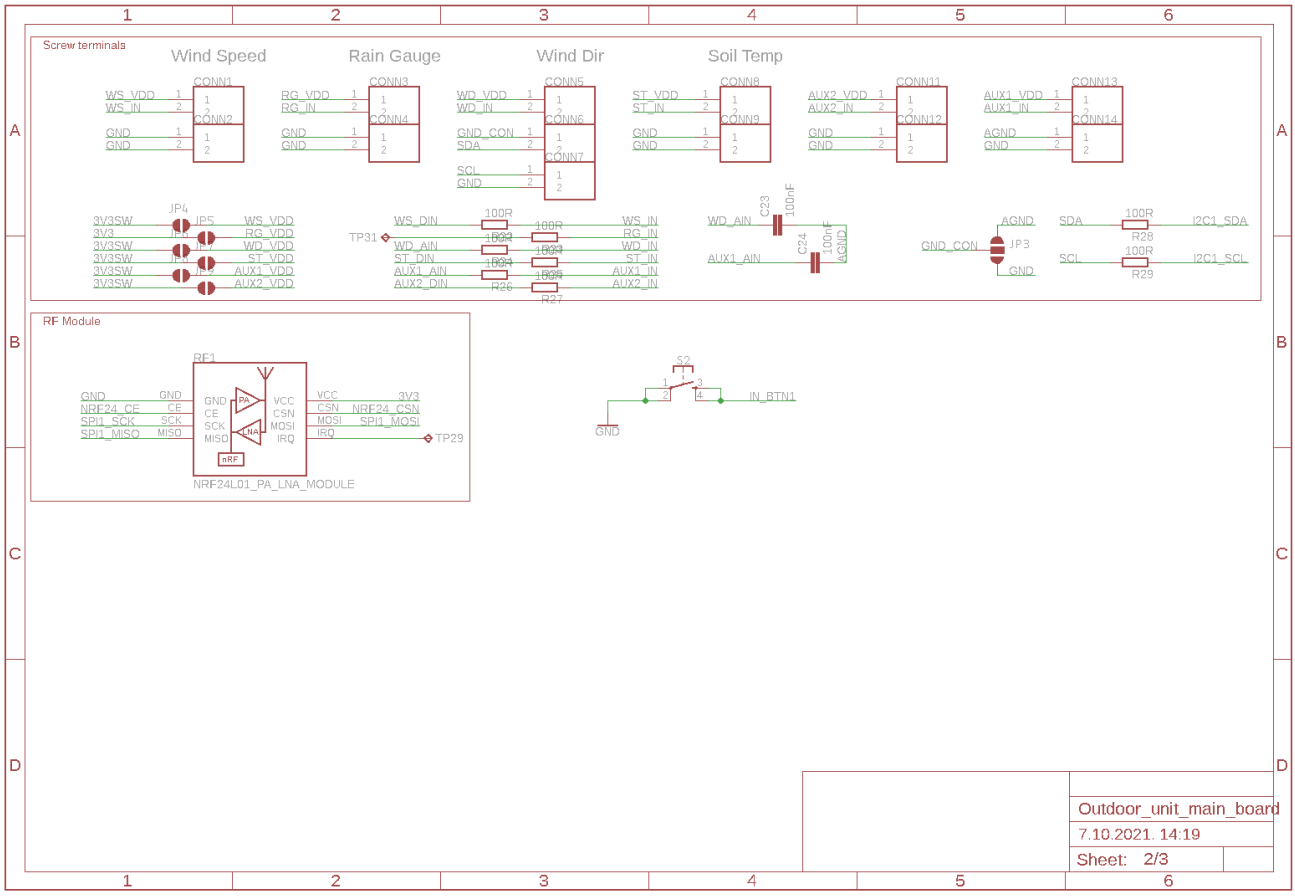
PRILOG 5


Pen Detect
PENIRQ
+VCC
X+
X–
Y+
Y–
Temperature Sensor
SAR
TSC2046
6– Channel MUX
CDAC
Comparator
Serial Data In/Out
IOVDD
DOUT
BUSY
CS
DCLK
DIN
Battery Monitor
VBAT
AUX
VREF
Internal 2.5V Reference

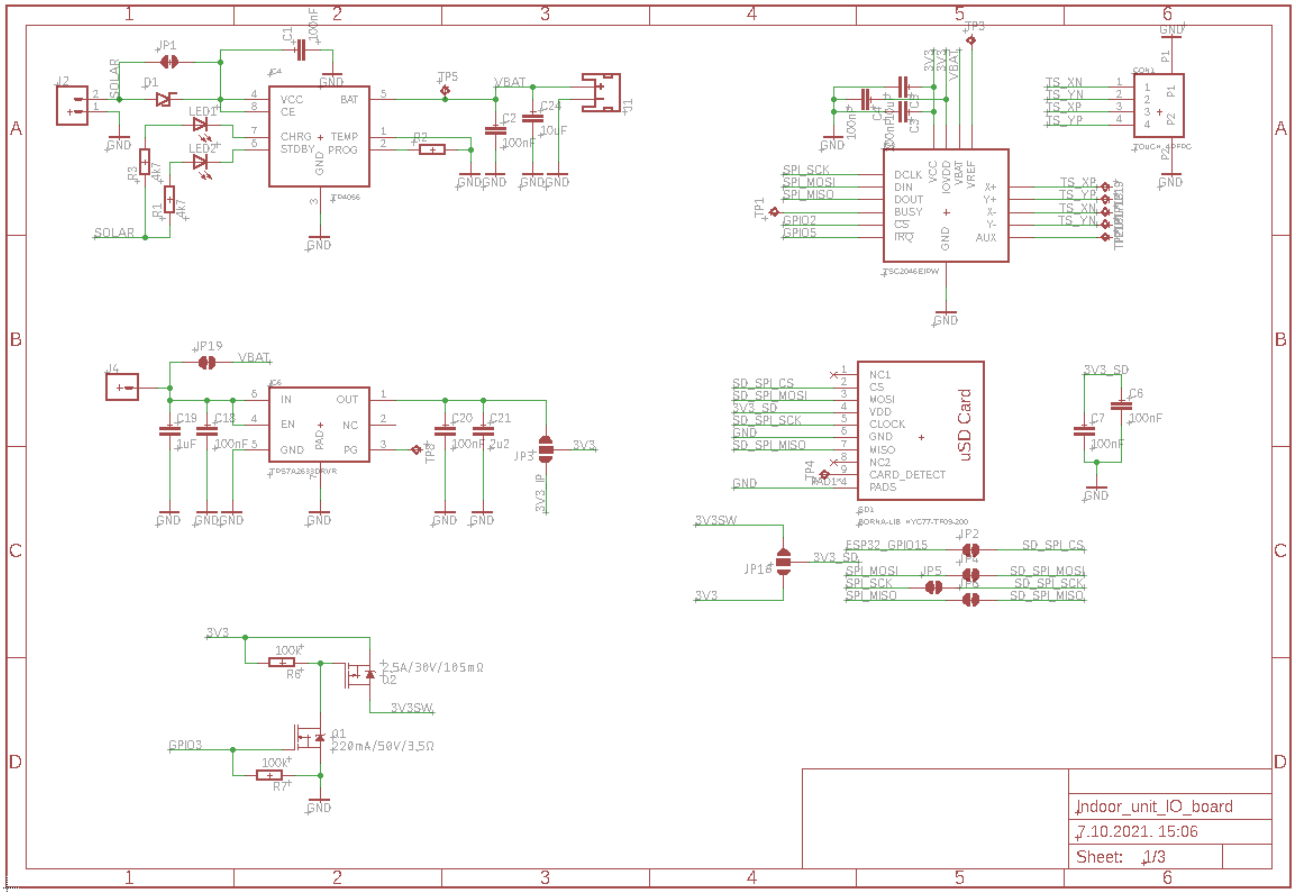
PRILOG 6


PWR/Solar/Batt
Connectors
MCUs Mics
MCU
Decoupling caps
Outdoor_unit_main_board
7.10.2021. 14:19
Sheet: 1/3

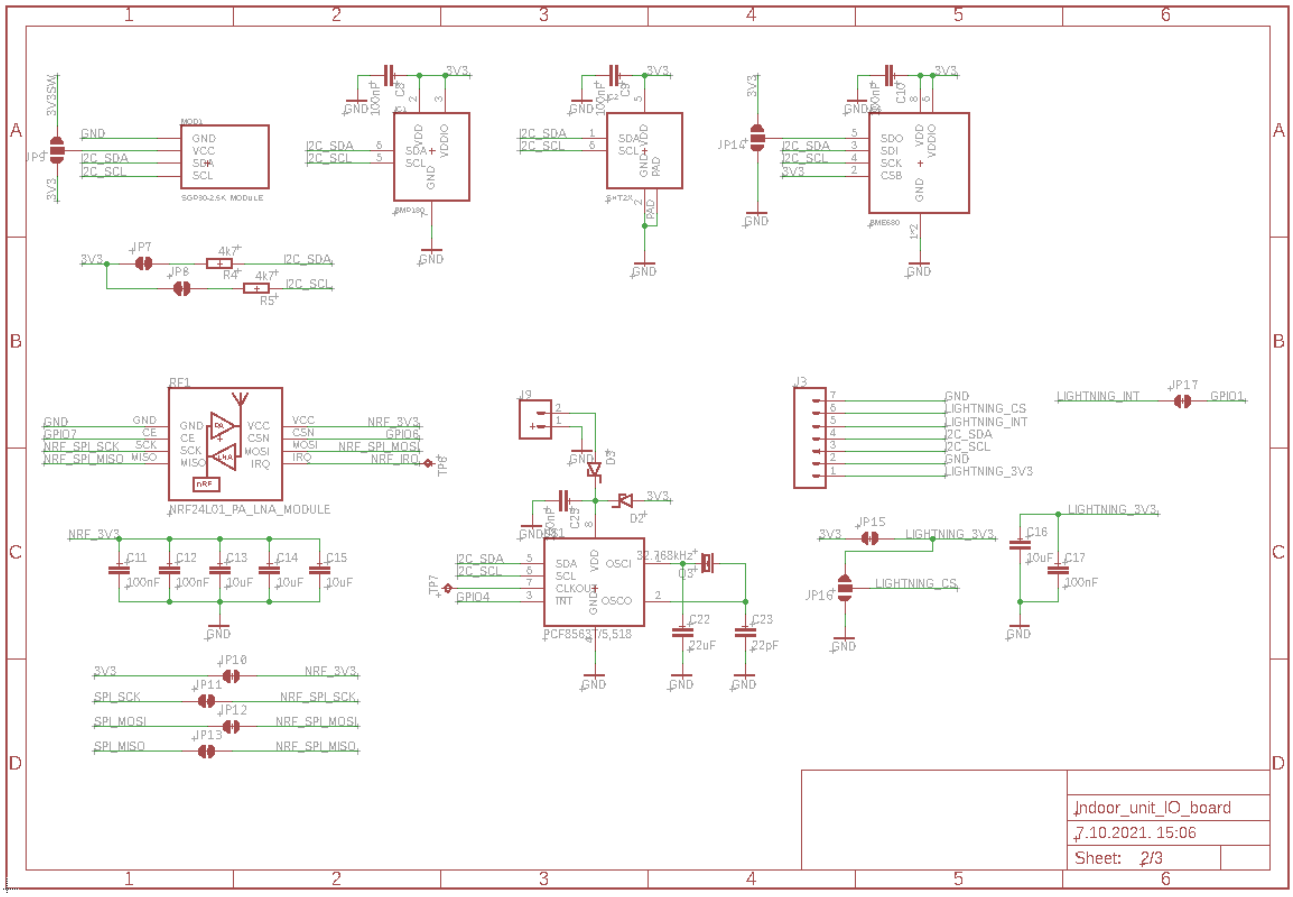
PRILOG 7


Screw terminals
Wind Speed
Rain Gauge
Wind Dir
Soil Temp
CONN1
CONN2
CONN3
CONN4
CONN5
CONN6
CONN7
CONN8
CONN9
CONN11
CONN12
CONN13
CONN14
WS_VDD
WS_IN
GND
RG_VDD
RG_IN
WD_VDD
WD_IN
GND_CON
SDA
SCL
ST_VDD
ST_IN
AUX2_VDD
AUX2_IN
AUX1_VDD
AUX1_IN
AGND
JP4
JP5
3V3SW
3V3
WS_VDD
RG_VDD
WD_VDD
ST_VDD
AUX1_VDD
AUX2_VDD
TP31
100R
WS_DIN
WD_AIN
ST_DIN
AUX1_AIN
AUX2_DIN
R26
R27
WS_IN
RG_IN
WD_IN
ST_IN
AUX1_IN
AUX2_IN
C23
C24
100nF
AGND
JP3
R28
R29
I2C1_SDA
I2C1_SCL
RF Module
RF1
NRF24_CE
SPI1_SCK
SPI1_MISO
SPI1_MOSI
NRF24_CSN
TP29
NRF24L01_PA_LNA_MODULE
S2
IN_BTN1
Outdoor_unit_main_board
7.10.2021. 14:19
Sheet: 2/3

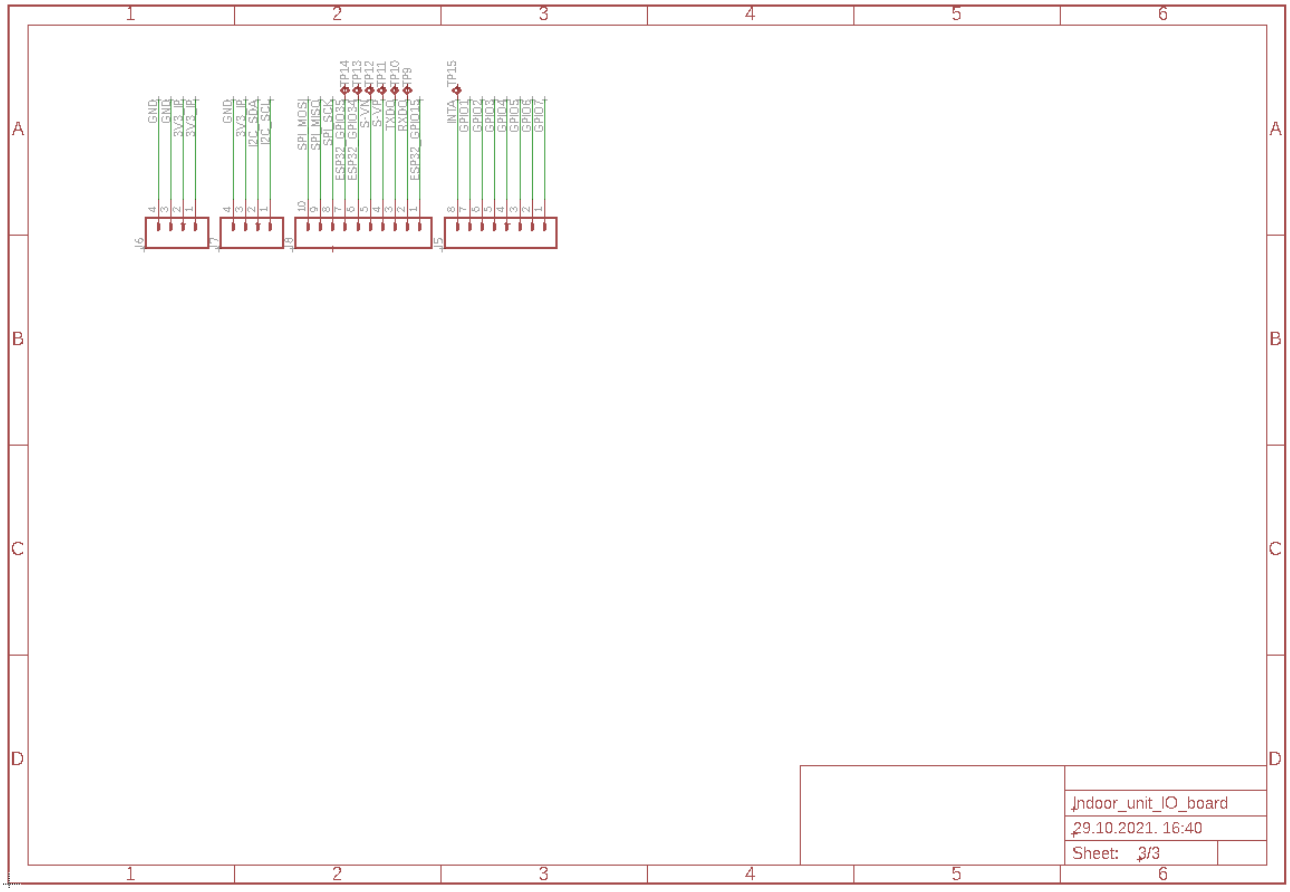
PRILOG 8


indoor_unit_IO_board
7.10.2021. 15:06
Sheet: 1/3
uSD Card
TSC2046EIPW
SOLAR
VBAT
3V3
GND
SPI_SCK
SPI_MOSI
SPI_MISO
TS_XP
TS_YP
TS_XN
TS_YN
SD_SPI_CS
SD_SPI_MOSI
SD_SPI_SCK
SD_SPI_MISO
3V3SW
GPIO3
ESP32_GPIO15
5A/30V/105mΩ
220mA/50V/3.5Ω

PRILOG 9



PRILOG 10


1
2
3
4
5
6
A
B
C
D
GND
GND
3V3_UP
3V3_UP
J6
GND
3V3_UP
I2C_SDA
I2C_SCL
J7
SPI_MOSI
SPI_MISO
SPI_SCK
ESP32_GPIO35
ESP32_GPIO34
ESP32_GPIO39
S_VN
S_VP
TXD0
RXD0
ESP32_GPIO9
J9
TP14
TP13
TP12
TP11
TP10
TP9
INTA
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
TP15
J5
Indoor_unit_IO_board
29.10.2021. 16:40
Sheet: 3/3

PRILOG 11

```
uint8_t OWMWeather::getForecastWeather(const char* _url, struct forecastListHandle *_f, struct
forecastDisplayHandle *_d)
{
  WiFiClient client;
  HTTPClient http;
  DynamicJsonDocument doc(24576);
  http.useHTTP10(true);

  if (http.begin(client, _url))
  {
    if (http.GET() > 0)
    {
      DeserializationError err = deserializeJson(doc, http.getStream());
      if (err) return 0;
      if (atoi(doc["cod"]) == 200)
      {
        _f->numberOfData = doc["cnt"];
        for (int i = 0; i < _f->numberOfData; i++)
        {
          removeCroLetters(strncpy(_f->forecast[i].weatherDesc, doc["list"][i]["weather"][0]["description"],
sizeof(_f->forecast[i].weatherDesc) - 1));
          strncpy(_f->forecast[i].weatherIcon, doc["list"][i]["weather"][0]["icon"], sizeof(_f-
>forecast[i].weatherIcon) - 1);
          _f->forecast[i].clouds = doc["list"][i]["clouds"]["all"];
          _f->forecast[i].humidity = doc["list"][i]["main"]["humidity"];
          _f->forecast[i].probability = (float)(doc["list"][i]["pop"]) * 100;
          _f->forecast[i].weatherId = doc["list"][i]["weather"][0]["id"];
          _f->forecast[i].pressureGnd = doc["list"][i]["main"]["grnd_level"];
          _f->forecast[i].visibility = doc["list"][i]["visibility"];
          _f->forecast[i].pressure = doc["list"][i]["main"]["pressure"];
          _f->forecast[i].windDir = doc["list"][i]["wind"]["deg"];
          _f->forecast[i].minTemp = doc["list"][i]["main"]["temp_min"];
          _f->forecast[i].temp = doc["list"][i]["main"]["temp"];
          _f->forecast[i].maxTemp = doc["list"][i]["main"]["temp_max"];
          _f->forecast[i].feelsLike = doc["list"][i]["main"]["feels_like"];
          _f->forecast[i].windSpeed = doc["list"][i]["wind"]["speed"];
          _f->forecast[i].windGust = doc["list"][i]["wind"]["gust"];
          _f->forecast[i].rain = doc["list"][i]["rain"]["3h"];
          _f->forecast[i].snow = doc["list"][i]["snow"]["3h"];
          _f->forecast[i].timestamp = doc["list"][i]["dt"];
        }

        // Group data into "days"
        // Skip first element
        int n = 1;
        _f->startElement[0] = 0;
        for (int i = 1; i < _f->numberOfData; i++)
        {
          if (epochToHuman(_f->forecast[i].timestamp).tm_hour == 0)
          {
            _f->startElement[n] = i;
            n++;
          }
        }
        _f->startElement[n] = _f->numberOfData;

        // Calculate avg and max data for forecast display
        memset(_d, 0, sizeof(forecastDisplayHandle) * 7);
        for (int i = 0; i < 6; i++)
        {
          int nElements = _f->startElement[i + 1] - _f->startElement[i];
          _d[i].maxTemp = _f->forecast[_f->startElement[i]].maxTemp;
          _d[i].minTemp = _f->forecast[_f->startElement[i]].minTemp;
          _d[i].maxWindSpeed = _f->forecast[_f->startElement[i]].windGust;
          float eastWestVectSum = 0;
          float northSouthVectSum = 0;
          for (int j = _f->startElement[i]; j < _f->startElement[i + 1]; j++)
          {
            _d[i].avgPressure += _f->forecast[j].pressureGnd;
            _d[i].avgHumidity += _f->forecast[j].humidity;
            eastWestVectSum += _f->forecast[j].windSpeed * sin(_f->forecast[j].windDir * PI / 180);
            northSouthVectSum += _f->forecast[j].windSpeed * cos(_f->forecast[j].windDir * PI / 180);
            if (_d[i].maxTemp < _f->forecast[j].maxTemp) _d[i].maxTemp = round(_f->forecast[j].maxTemp);
            if (_d[i].minTemp > _f->forecast[j].minTemp) _d[i].minTemp = round(_f->forecast[j].minTemp);
            if (_d[i].maxWindSpeed < _f->forecast[j].windGust) _d[i].maxWindSpeed = _f->forecast[j].windGust;
            if (epochToHuman(_f->forecast[j].timestamp).tm_hour == 15)
            {
              _d[i].weatherIcon = _f->forecast[j].weatherIcon;
```

```
            _d[i].weatherDesc = _f->forecast[j].weatherDesc;
          }
        }
        _d[i].avgPressure /= nElements;
        _d[i].avgHumidity /= nElements;
        eastWestVectSum /= nElements;
        northSouthVectSum /= nElements;
        _d[i].avgWindSpeed = sqrt((eastWestVectSum * eastWestVectSum) + (northSouthVectSum * northSouthVectSum));
        _d[i].avgWindDir = atan2(eastWestVectSum, northSouthVectSum) * 180 / PI;
        _d[i].avgWindDir = (_d[i].avgWindDir >= 0 ? _d[i].avgWindDir : _d[i].avgWindDir + 360);
      }
      _f->shiftDay = 1;
      if (_f->startElement[1] - _f->startElement[0] < 3) _f->shiftDay = 0;
    }
    else
    {
        return 0;
    }
  }
  http.end();
 }
}
```

PRILOG 12

```
void GUI::init(Inkplate *_inkPtr)
{
  _ink = _inkPtr;
}

void GUI::drawMainScreen(struct sensorData *_sensor, struct currentWeatherHandle *_current, struct
forecastListHandle *_forecastList, struct forecastDisplayHandle *_displayForecast, struct oneCallApiHandle *_one,
struct tm *_time)
{
    char tmp[50];
    _ink->clearDisplay();
    _ink->setTextSize(1);
    _ink->fillRect(20, 360, 760, 3, BLACK);
    _ink->setFont(DISPLAY_FONT_SMALL);
    _ink->setTextSize(2);
    _ink->setCursor(1, 70);
    _ink->print(_current->weatherDesc);
    _ink->setCursor(1, 20);
    _ink->print(_current->city);
    _ink->setTextSize(1);
    _ink->drawBitmap(770, 0, refIcon, 30, 30, BLACK);

    sprintf(tmp, "%1d:%02d %d/%d/%04d %3s", _time->tm_hour,  _time->tm_min,  _time->tm_mday,  _time->tm_mon +
1,  _time->tm_year + 1900, DOW[ _time->tm_wday]);

    _ink->setFont(DISPLAY_FONT);
    _ink->fillRect(4, 50, 796, 3, BLACK);
    _ink->setCursor(267, 35);
    _ink->print(tmp);

    _ink->drawBitmap(20, 85, weatherIcon(atoi(_current->weatherIcon)), 50, 50, BLACK);
    _ink->setTextSize(2);
    _ink->setCursor(80, 130);
    _ink->print(round(_current->temp), 0);
    _ink->setCursor(_ink->getCursorX() + 5, 110);
    _ink->print('o');
    _ink->setTextSize(1);
    _ink->drawBitmap(1, 140, iconTlak, 40, 40, BLACK);
    _ink->setCursor(45, 170);
    _ink->print(_current->pressureGnd, DEC);
    _ink->drawBitmap(145, 140, iconVlaga, 40, 40, BLACK);
    _ink->setCursor(195, 170);
    _ink->print(_current->humidity);
    _ink->drawBitmap(1, 190, iconVjetar, 40, 40, BLACK);
    _ink->setCursor(45, 220);
    _ink->print(_current->windSpeed, 1);
    _ink->print(" m/s | ");
    _ink->print(oznakeVjetar[int((_current->windDir / 22.5) + .5) % 16]);
    _ink->drawBitmap(1, 240, iconSunrise, 40, 40 , BLACK);
    sprintf(tmp, "%d:%02d", epochToHuman(_current->sunrise).tm_hour, epochToHuman(_current->sunrise).tm_min);
    _ink->setCursor(40, 270);
    _ink->print(tmp);
    _ink->drawBitmap(130, 240, iconSunset, 40, 40 , BLACK);
    sprintf(tmp, "%d:%02d", epochToHuman(_current->sunset).tm_hour, epochToHuman(_current->sunset).tm_min);
    _ink->setCursor(175, 270);
    _ink->print(tmp);

    _ink->drawBitmap(300, 70, indoorIcon, 40, 40, BLACK);
    _ink->drawBitmap(300, 140, iconTlak, 40, 40, BLACK);
    _ink->setCursor(350, 170);
    _ink->print(_sensor->pressure, 1);
    _ink->drawBitmap(300, 190, iconVlaga, 40, 40, BLACK);
    _ink->setCursor(350, 220);
    _ink->print(_sensor->humidity, 1);
    _ink->drawBitmap(300, 240, iconTemp, 40, 40, BLACK);
    _ink->setCursor(350, 270);
    _ink->print(_sensor->temp, 1);

    if (strlen(_one->alertEvent)) _ink->drawBitmap(720, 5, iconWarning, 40, 40, BLACK);

    int xOffset, xOffsetText;
    for (int i = 1; i < 6; i++)
    {
        xOffset = i * 160;
        xOffsetText = ((i - 1) * 160) + 10;
        _ink->fillRect(xOffset, 410, 3, 180, BLACK);
        _ink->drawBitmap(xOffset - 105, 420, weatherIcon(atoi(_displayForecast[i - _forecastList-
>shiftDay].weatherIcon)), 50, 50, BLACK);
```

```
        _ink->setTextSize(1);
        _ink->setFont(DISPLAY_FONT_SMALL);
        printStringCenter(_displayForecast[i - _forecastList->shiftDay].weatherDesc, xOffset - 80, 482);
        sprintf(tmp, "%d.%d", epochToHuman(_forecastList->forecast[_forecastList->startElement[i - _forecastList-
>shiftDay]].timestamp).tm_mday, epochToHuman(_forecastList->forecast[_forecastList->startElement[i -_forecastList-
>shiftDay]].timestamp).tm_mon + 1);
        printStringCenter(tmp, xOffset - 80, 415);
        sprintf(tmp, "%d hPa   %d %%", _displayForecast[i - _forecastList->shiftDay].avgPressure,
_displayForecast[i - _forecastList->shiftDay].avgHumidity);
        printStringCenter(tmp, xOffset - 80, 575);
        sprintf(tmp, "%.1f / %.1f m/s   %s", _displayForecast[i - _forecastList->shiftDay].avgWindSpeed,
_displayForecast[i - _forecastList->shiftDay].maxWindSpeed, oznakeVjetar[int((_displayForecast[i - _forecastList-
>shiftDay].avgWindDir / 22.5) + .5) % 16]);
        printStringCenter(tmp, xOffset - 80, 588);
        _ink->setFont(DISPLAY_FONT);
        sprintf(tmp, "%d | %d", _displayForecast[i - _forecastList->shiftDay].maxTemp, _displayForecast[i -
_forecastList->shiftDay].minTemp);
        printStringCenter(tmp, xOffset - 80, 520);
        printStringCenter((char*)DOW[epochToHuman(_forecastList->forecast[_forecastList->startElement[i -
_forecastList->shiftDay]].timestamp).tm_wday], xOffset - 80, 400);
    }

    for (int i = 1; i < 3; i++)
    {
        _ink->fillRect(267 * i, 60, 3, 290, BLACK);
    }
    _ink->display();
}
```

PRILOG 13

Program unutrašnje jedinice

```
// Includes for communication and periph.
#include <Wire.h>
#include <SPI.h>
#include "driver/rtc_io.h"
#include "timeAndDate.h"

// WiFi and Internet includes
#include <WiFi.h>
#include <esp_wifi.h>
#include <WiFiUdp.h>
#include "Weather.h"

// Display includes (driver, icons, fonts, etc)
#include <Inkplate.h>
#include "GUI.h"
#include "Neucha_Regular17pt7b.h"
#include "RobotoCondensed_Regular6pt7b.h"
#include "icons.h"
#include <TSC2046E_Inkplate.h>

// Sensor includes
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <Adafruit_SGP30.h>

// Other includes
#include "sys/time.h"
#include "structs.h"
#include <EEPROM.h>
#include "PCF85063.h"
#include "communication.h"
#include "RF24_Inkplate.h"

// SD Card Library
#include <SdFat.h>

//.ttf to .h: https://rop.nl/truetype2gfx/
//Open Source fonts https://fonts.google.com/
#define DISPLAY_FONT &Neucha_Regular17pt7b
#define DISPLAY_FONT_SMALL &RobotoCondensed_Regular6pt7b

#define NEW_INKPLATE

// Objects / constructors
Inkplate display(INKPLATE_1BIT);
TSC2046E_Inkplate ts;
SPIClass *mySpi = NULL;
Adafruit_BME280 bme;
OWMWeather owm;
GUI gui;
pcf85063 rtc;
communication rf;
RF24_Inkplate radio(14, 15, 2000000);
Adafruit_SGP30 sgp;

// Consts for hour, minutes, seconds, ...
struct tm tNow;
time_t timeToWake;
int timeOffset;

const char ssid[] = "WIFI_SSID";
const char pass[] = "WIFI_PASSWORD";

uint8_t modeSelect = 0;
uint8_t forcedRef = 0;
uint8_t selectedDay = 0;
uint8_t selectedGraph = 0;
uint8_t wifiCounter = 0;
uint32_t sdDataDayOffset = 0;
uint16_t sdDataOffset = 0;

struct sensorData sensor;
struct currentWeatherHandle currentWeather;
struct forecastListHandle forecastList;
struct forecastDisplayHandle forecastDisplay[7];
struct oneCallApiHandle oneCallApi;
```

```
struct syncStructHandle syncStruct;
struct measruementHandle outdoorData;

void setup()
{
  // Varibles for storing status of each module while init
  uint8_t sdError = 0;
  uint8_t bmeError = 0;
  uint8_t radioError = 0;

  // Init System libraries
  Serial.begin(115200);
  Wire.begin();
  display.begin();
  EEPROM.begin(64);

  // Set output mode of MCP INT pin
  display.setIntOutputInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 1, 0, 0, 1);

  // Set touchscreen controller INT pin
  display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 13, INPUT_PULLUP);
  display.setIntPinInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 13, FALLING);

  // Set default pin state on touchscreen controller CS pin
  display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 10, OUTPUT);
  display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 10, HIGH);

  // Set default state of nRF24l01 pins
  display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 14, HIGH);
  display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 15, HIGH);

  // Set RTC INT PIN
  display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 12, INPUT_PULLUP);
  display.setIntPinInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 12, FALLING);

  // Set 3V3SW pin as output
  display.pinModeInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 11, OUTPUT);
  display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 11, LOW);

  // Clear all INT flags on MCP
  display.getINTstateInternal(MCP23017_INT_ADDR, display.mcpRegsInt);

  // Init GUI communication and RTC library
  gui.init(&display);
  rf.init(&radio, &display, &rtc);
  rtc.RTCinit();

  // Get the SPI object from Inkplate library
  mySpi = display.getSPIptr();
  mySpi->begin(14, 12, 13, 15);

  // Init touchscreen controller
  ts.begin(mySpi, &display, 10, 13);
  ts.calibrate(800, 3420, 3553, 317, 0, 799, 0, 599);

  // Init all the modules
  radioError = radio.begin(mySpi, &display);
  bmeError = bme.begin(0x76);
  sdError = display.sdCardInit();

  display.setTextWrap(false);
  display.setFont(DISPLAY_FONT);
  display.setTextColor(BLACK, WHITE);
  display.setCursor(0, 24);

  // Show is there any error with init of sensors
  if (!(sdError && bmeError && radioError))
  {
    display.print("Pogreška s ");
    if (!sdError)
    {
      display.print("SD karticom");
      display.setCursor(0, display.getCursorY() + 24);
    }
    if (!bmeError)
    {
      display.print("senzorom temp., vlage i tlaka");
      display.setCursor(0, display.getCursorY() + 24);
    }
    if (!radioError)
    {
```

```
    display.print("radijskim modulom");
    display.setCursor(0, display.getCursorY() + 24);
  }
  display.print("Provjerite modul i ponovno pokrenite uredjaj!");
  display.display();
  esp_deep_sleep_start();
  while (1);
}

// Change WiFi name
WiFi.mode(WIFI_STA);
WiFi.disconnect();
WiFi.setHostname("WeatherPaper");

// Setup a temperatue, humidity and pressure sensor
bme.setSampling(Adafruit_BME280::MODE_FORCED,
                Adafruit_BME280::SAMPLING_X16, // temperature
                Adafruit_BME280::SAMPLING_X16, // pressure
                Adafruit_BME280::SAMPLING_X16, // humidity
                Adafruit_BME280::FILTER_X16   );

// Try to connect to WiFi
uint8_t wiFiRetry = 30;
int n = WiFi.scanNetworks();
if (n > 6) n = 6;
display.clearDisplay();
display.setCursor(0, 24);
display.println("ESP32 WeatherPaper\nTrazenje WiFi Mreza...");
display.display();
for (int i = 0; i < n; i++)
{
  display.setCursor(70, 100 + (i * 35));
  display.print(WiFi.SSID(i));
  display.print((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? ' ' : '*');
  display.print(' ');
  display.print(WiFi.RSSI(i), DEC);
}
display.partialUpdate();
display.setCursor(70, 550);
display.print(F("Spajanje na "));
display.print(ssid);
display.partialUpdate(false, true);
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED && wiFiRetry > 0)
{
  delay(1000);
  display.print('.');
  display.partialUpdate(false, true);
  wiFiRetry--;
}
if(WiFi.status() == WL_CONNECTED)
{
  display.print("spojeno! Pricekajte...");
  display.partialUpdate();
}
else
{
  display.print("Spajanje neuspjesno!");
  display.display();
  esp_deep_sleep_start();
  while (1);
}

// Read data from sensors and from the Internet
readSensor(&sensor);
updateWeatherData();

// Check if the time is set and if it not set it by connecting to NTP serer
// After that try to sync with outdoor unit and set new wakeup time and save it in EEPROM
time_t myTime;
if (!rtc.isClockSet() && readNTP(&myTime))
{
  rtc.setClock(myTime);
  radio.powerUp();
  rf.setupCommunication();
  if (rf.sync(&syncStruct))
  {
    display.clearDisplay();
    display.print("Sinkronizacija uspjesna!");
    display.display();
    timeToWake = (time_t)syncStruct.sendEpoch;
```

```
      rtc.setAlarm(timeToWake);
      EEPROM.put(0, timeToWake);
      EEPROM.commit();
    }
    else
    {
      // If sync failed, make a new wakeup time, just in case and save it to EEPROM
      display.clearDisplay();
      display.print("Sinkronizacija nije uspjela");
      display.display();
      timeToWake = rf.newWakeupTime(rtc.getClock());
      rtc.setAlarm(timeToWake);
      EEPROM.put(0, timeToWake);
      EEPROM.commit();
    }
  }
  // Check if the time to wake has already passed. If it is, calculate new wakeup time
  EEPROM.get(0, timeToWake);
  if ((rtc.getClock() >= timeToWake) || (abs(timeToWake - rtc.getClock()) > (60 * WAKEUP_INTERVAL)))
  {
    timeToWake = rf.newWakeupTime(rtc.getClock());
    rtc.setAlarm(timeToWake);
    EEPROM.put(0, timeToWake);
    EEPROM.commit();
  }
  tNow = epochToHuman(rtc.getClock());
  gui.drawMainScreen(&sensor, &currentWeather, &forecastList, forecastDisplay, &oneCallApi, &outdoorData, &tNow);
  esp_wifi_stop();
  btStop();
}

void loop()
{
  int tsX, tsY;
  rtc_gpio_isolate(GPIO_NUM_12);
  // Make ESP32 GPIO34 INT pin that wakes up ESP32
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_34, 1);
  esp_light_sleep_start();

  // If ESP32 has woken up, see what caused wake up and clear the INT flags in MCP23017
  uint16_t intPins = display.getINTInternal(MCP23017_INT_ADDR, display.mcpRegsInt);
  display.getINTstateInternal(MCP23017_INT_ADDR, display.mcpRegsInt);

  // If the touchscreen woke ESP32 up, get the X and Y of touch and check what has been pressed
  if ((intPins & (1 << 13)) && ts.available(&tsX, &tsY))
  {
    switch (modeSelect)
    {
      case 0:
        // Touched one of weather forecast days
        if (touchArea(tsX, tsY, 0, 410, 800, 190))
        {
          selectedDay = tsX / 160;
          drawDays(selectedDay, true);
          modeSelect = 1;
        }

        // NTP Sync button
        if (touchArea(tsX, tsY, 660, 5, 40, 40))
        {
          time_t _myNewTime;
          time_t _myNewAlarmTime;
          uint8_t _wiFiRetry = 0;
          writeInfoBox(350, "Sinkronizacija sata...");
          display.partialUpdate();
          if (readNTP(&_myNewTime))
          {
            WiFi.begin(ssid, pass);
            while (WiFi.status() != WL_CONNECTED && _wiFiRetry < 15)
            {
              delay(1000);
              _wiFiRetry++;
            }
            if(WiFi.status() == WL_CONNECTED)
            {
              rtc.setClock(_myNewTime);
              _myNewAlarmTime = rf.newWakeupTime(_myNewTime);
              rtc.setAlarm(_myNewAlarmTime);
              EEPROM.put(0, _myNewAlarmTime);
              EEPROM.commit();
            }
```

```
          }
          esp_wifi_stop();
          btStop();
          gui.drawMainScreen(&sensor, &currentWeather, &forecastList, forecastDisplay, &oneCallApi, &outdoorData,
&tNow);
        }

        // Outdoor unit sync
        if (touchArea(tsX, tsY, 710, 5, 40, 40))
        {
          radio.powerUp();
          rf.setupCommunication();
          uint8_t _myNewSync = rf.sync(&syncStruct);
          if (_myNewSync)
          {
            rtc.setAlarm(syncStruct.sendEpoch);
            EEPROM.put(0, (time_t)syncStruct.sendEpoch);
            EEPROM.commit();
          }
          gui.drawMainScreen(&sensor, &currentWeather, &forecastList, forecastDisplay, &oneCallApi, &outdoorData,
&tNow);
        }

        // Forced data refresh button
        if (touchArea(tsX, tsY, 760, 0, 40, 50))
        {
          display.setFont(DISPLAY_FONT);
          display.setCursor(610, 35);
          writeInfoBox(350, "Osvjez. podataka, pricekajte");
          display.partialUpdate();
          forcedRef = 1;
        }

        // If it's pressed, show outdoor unit data by today
        if (touchArea(tsX, tsY, 534, 60, 267, 290))
        {
          modeSelect = 2;
          gui.drawOutdoorData(&rf, rtc.getClock(), 0, &sdDataOffset, selectedGraph, 1);
          display.display();
        }

        // If it's pressed, show indoor unit data starting by today
        if (touchArea(tsX, tsY, 267, 60, 267, 290))
        {
          modeSelect = 3;
          gui.drawIndoorData(&rf, rtc.getClock(), 0, &sdDataOffset, selectedGraph, 1);
          display.display();
        }
        break;

      // Weather forecast
      case 1:
        if (touchArea(tsX, tsY, 10, 570, 30, 30))
        {
          gui.drawMainScreen(&sensor, &currentWeather, &forecastList, forecastDisplay, &oneCallApi, &outdoorData,
&tNow);
          modeSelect = 0;
          selectedDay = 0;
          selectedGraph = 0;
          sdDataDayOffset = 0;
        }

        // Go one day back (if it's possible)
        if (touchArea(tsX, tsY, 0, 0, 75, 75) && selectedDay > 0)
        {
          selectedDay--;
          drawDays(selectedDay, false);
          modeSelect = 1;
        }

        // Go one day FW (if it's possible)
        if (touchArea(tsX, tsY, 730, 0, 70, 50) && selectedDay < 4)
        {
          selectedDay++;
          drawDays(selectedDay, false);
          modeSelect = 1;
        }

        // Show graph for selected data
        if (touchArea(tsX, tsY, 50, 550, 700, 150))
        {
```

```
          selectedGraph = (tsX - 50) / 70;
          drawSelectedGraph(selectedGraph, selectedDay);
          display.partialUpdate();
        }

        // Show pop-up "window" with data of selected day and time
        if (touchArea(tsX, tsY, 0, 70, 800, 70))
        {
          int selectedElement = tsX / 100;
          int _yPos = 240;
          const int _fontScale = 2;
          const int _fontYSize = 14 * _fontScale;
          if (selectedElement < (forecastList.startElement[selectedDay + 2 - forecastList.shiftDay] -
forecastList.startElement[selectedDay + 1 - forecastList.shiftDay]))
          {
            int element = forecastList.startElement[selectedDay + 1 - forecastList.shiftDay] + selectedElement;
            display.fillRect(198, 198, 404, 304, BLACK);
            display.fillRect(200, 200, 400, 300, WHITE);
            display.setFont(DISPLAY_FONT_SMALL);
            display.setTextSize(_fontScale);
            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print(forecastList.forecast[element].weatherDesc);
            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print("Temp.: ");
            display.print(forecastList.forecast[element].minTemp, 1);
            display.print("[Min] / ");
            display.print(forecastList.forecast[element].temp, 1);
            display.print(" / ");
            display.print(forecastList.forecast[element].maxTemp, 1);
            display.print("[Max] C");

            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print("Tlak zraka: ");
            display.print(forecastList.forecast[element].pressureGnd);
            display.print("hPa");

            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print("Brzina vjetra: ");
            display.print(forecastList.forecast[element].windSpeed, 1);
            display.print('/');
            display.print(forecastList.forecast[element].windGust, 1);
            display.print("m/s");

            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print("Smjer vjetra: ");
            display.print(oznakeVjetar[int((forecastList.forecast[element].windDir / 22.5) + .5) % 16]);
            display.print('[');
            display.print(forecastList.forecast[element].windDir);
            display.print(" st.]");

            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print("Naoblaka: ");
            display.print(forecastList.forecast[element].clouds);
            display.print('%');

            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print("Snijeg: ");
            display.print(forecastList.forecast[element].snow, 1);
            display.print("mm");

            display.setCursor(220, _yPos);  _yPos += _fontYSize;
            display.print("Kisa: ");
            display.print(forecastList.forecast[element].rain, 1);
            display.print("mm");
          }
          display.setTextSize(1);
          display.partialUpdate();
        }
        break;

        // Indoor and outdoor graphs
        case 2:
        case 3:
          // Move one day back
          if (touchArea(tsX, tsY, 0, 0, 75, 75))
          {
            sdDataDayOffset++;
            sdDataOffset = 0;
            if (modeSelect == 2) gui.drawOutdoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
```

```
            if (modeSelect == 3) gui.drawIndoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            display.partialUpdate();
          }

          // Move one day FW
          if (touchArea(tsX, tsY, 730, 0, 70, 50) && sdDataDayOffset != 0)
          {
            sdDataDayOffset--;
            sdDataOffset = 0;
            if (modeSelect == 2) gui.drawOutdoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            if (modeSelect == 3) gui.drawIndoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            display.partialUpdate();
          }

          // Get back to main screen
          if (touchArea(tsX, tsY, 10, 570, 30, 30))
          {
            sdDataDayOffset = 0;
            modeSelect = 0;
            sdDataOffset = 0;
            selectedGraph = 0;
            gui.drawMainScreen(&sensor, &currentWeather, &forecastList, forecastDisplay, &oneCallApi, &outdoorData,
&tNow);
          }

          // Move one dataset FW
          if (touchArea(tsX, tsY, 90, 140, 30, 30))
          {
            if (sdDataOffset != 0) sdDataOffset--;
            if (modeSelect == 2) gui.drawOutdoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            if (modeSelect == 3) gui.drawIndoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            display.partialUpdate();
          }

          // Move one dataset back
          if (touchArea(tsX, tsY, 680, 140, 30, 30))
          {
            sdDataOffset++;
            if (modeSelect == 2) gui.drawOutdoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            if (modeSelect == 3) gui.drawIndoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            display.partialUpdate();
          }

          // Go back to the first measurement in day
          if (touchArea(tsX, tsY, 40, 140, 30, 30))
          {
            sdDataOffset = 0;
            if (modeSelect == 2) gui.drawOutdoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            if (modeSelect == 3) gui.drawIndoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph);
            display.partialUpdate();
          }

          // Go back to the last measurement in day
          if (touchArea(tsX, tsY, 730, 140, 30, 30))
          {
            sdDataDayOffset = 0;
            if (modeSelect == 2) gui.drawOutdoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph, 1);
            if (modeSelect == 3) gui.drawIndoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset,
selectedGraph, 1);
            display.partialUpdate();
          }

          if (touchArea(tsX, tsY, 50, 550, 750, 150) && modeSelect == 2)
          {
            selectedGraph = (tsX - 50) / 70;
            gui.drawOutdoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset, selectedGraph);
            display.partialUpdate();
          }

          if(touchArea(tsX, tsY, 50, 490, 750, 150) && modeSelect == 3)
          {
```

```
            selectedGraph = (tsX - 50) / 70;
            gui.drawIndoorData(&rf, rtc.getClock(), sdDataDayOffset, &sdDataOffset, selectedGraph);
            display.partialUpdate();
          }
          break;
      }
    }

    if ((rtc.checkAlarmFlag() && (intPins & (1 << 12))) || forcedRef)
    {
      uint8_t _rxOk = 0;
      time_t _sgpTimeout = rtc.getClock() + 16;
      uint16_t eco2Base, tvocBase;

      // Power up and init C02 sensor
      display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 11, HIGH);
      delay(10);
      sgp.begin();
      EEPROM.get(8, eco2Base);
      EEPROM.get(10, tvocBase);
      sgp.setIAQBaseline(eco2Base, tvocBase);
      display.setFont(DISPLAY_FONT_SMALL);
      display.setTextSize(1);
      display.setCursor(5, 46);
      display.print("Dohvacanje novih podataka");
      display.partialUpdate();

      // If it's not the user forced update of data, that means is RTC and it's time to get the data from outdook
unit
      if (!forcedRef)
      {
        wifiCounter++;
        rtc.clearAlarm();
        _rxOk = rf.getData(&syncStruct, &outdoorData);
        timeToWake = (time_t)syncStruct.sendEpoch;
        rtc.setAlarm(timeToWake);
        EEPROM.put(0, timeToWake);
        EEPROM.commit();
      }

      // Get new weather data from the internet every 30 minutes
      if (wifiCounter > 2 || forcedRef)
      {
        wifiCounter = 0;
        esp_wifi_start();
        WiFi.mode(WIFI_STA);
        WiFi.begin(ssid, pass);
        int i = 0;
        while (WiFi.status() != WL_CONNECTED && (i < 15))
        {
          delay(1000);
          display.print('.');
          display.partialUpdate();
        }
        if (WiFi.status() == WL_CONNECTED) updateWeatherData();
        esp_wifi_stop();
        btStop();
      }

      // CO2 sensor must work at leasts 15 seconds in order to get any data
      while(rtc.getClock() < _sgpTimeout)
      {
          display.print('.');
          display.partialUpdate();
          delay(1000);
      }
      sgp.IAQmeasure();
      sgp.IAQmeasureRaw();
            readSensor(&sensor);
      // Save new baseline and power down sensor
      sgp.getIAQBaseline(&eco2Base, &tvocBase);
      display.digitalWriteInternal(MCP23017_INT_ADDR, display.mcpRegsInt, 11, LOW);
      EEPROM.put(8, eco2Base);
      EEPROM.put(10, tvocBase);
      EEPROM.commit();
      if (!forcedRef) rf.saveDataToSD(&sensor, _rxOk?&outdoorData:NULL);
      forcedRef = 0;
      modeSelect = 0;
      selectedDay = 0;
      selectedGraph = 0;
      sdDataDayOffset = 0;
```

```
    sdDataOffset = 0;
    tNow = epochToHuman(rtc.getClock());
    gui.drawMainScreen(&sensor, &currentWeather, &forecastList, forecastDisplay, &oneCallApi, &outdoorData, &tNow);
  }
}

void writeInfoBox(int y, char* c)
{
  int16_t  x1, y1;
  uint16_t w, h;
  int x;
  display.setTextSize(2);
  display.getTextBounds(c, 0, y, &x1, &y1, &w, &h);
  x = (800 - w) / 2;
  display.getTextBounds(c, x, y, &x1, &y1, &w, &h);
  display.fillRect(x1 - 3, y1 - 3, w + 6, h + 6, WHITE);
  display.drawRect(x1 - 2, y1 - 2, w + 4, h + 4, BLACK);
  display.drawRect(x1 - 1, y1 - 1, w + 2, h + 2, BLACK);
  display.setCursor(x, y);
  display.print(c);
  display.setTextSize(1);
}

void readSensor(struct sensorData *_s)
{
  _s->epoch = (uint32_t)rtc.getClock();
  bme.takeForcedMeasurement();
  _s->temp = bme.readTemperature();
  _s->humidity = bme.readHumidity();
  _s->pressure = bme.readPressure() / 100.0F;
  _s->eco2 = sgp.eCO2;
  _s->tvoc = sgp.TVOC;
  _s->rawH2 = sgp.rawH2;
  _s->rawEthanol = sgp.rawEthanol;
  _s->battery = ts.getBatteryVoltage();
}

bool readNTP(time_t *_epoch)
{
  IPAddress ntpIp;
  WiFiUDP udp;
  const char* NTPServer = "hr.pool.ntp.org";
  uint16_t ntpPort = 123;
  uint8_t ntpPacket[48];
  unsigned long _ntpTimeout;

  udp.begin(8888);
  if (!WiFi.hostByName(NTPServer, ntpIp)) return 0;

  ntpPacket[0] = B11100011; //Clock is unsync, NTP version 4, Symmetric passive
  ntpPacket[1] = 0;     // Stratum, or type of clock
  ntpPacket[2] = 60;     // Polling Interval
  ntpPacket[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  ntpPacket[12]  = 49;
  ntpPacket[13]  = 0x4E;
  ntpPacket[14]  = 49;
  ntpPacket[15]  = 52;
  udp.beginPacket(ntpIp, 123);
  udp.write(ntpPacket, 48);
  udp.endPacket();
  _ntpTimeout = millis();
  while ((unsigned long)(millis() - _ntpTimeout) < 5000)
  {
    if (udp.parsePacket() >= 48)
    {
      udp.read(ntpPacket, 48);
      uint32_t unix = ntpPacket[40] << 24 | ntpPacket[41] << 16 | ntpPacket[42] << 8 | ntpPacket[43];
      *_epoch = unix - 2208988800UL + currentWeather.timezone;
      return true;
    }
  }
  return false;
}

void updateWeatherData()
{
  owm.getCurrentWeather("http://api.openweathermap.org/data/2.5/weather?&lat=[LAT]&lon=[LON]&units=metric&lang=hr&A
PPID=[API_KEY]", &currentWeather);
  owm.getForecastWeather("http://api.openweathermap.org/data/2.5/forecast?lat=[LAT]&lon=[LON]&units=metric&lang=hr&
APPID=[API_KEY]", &forecastList, forecastDisplay);
```

```
  owm.oneCall("http://api.openweathermap.org/data/2.5/onecall?lat=[LAT]&lon=[LON]&units=metric&lang=hr&exclude=curr
ent,minutely,hourly,daily&appid=[API_KEY]", &oneCallApi);  //Upozorenja
}

void changeLetters(char *p)
{
  int webTextSize = strlen(p);
  for (int16_t i = 0; i < webTextSize; i++)
  {
    if (p[i] == 196 || p[i] == 197) memmove(&p[i], &p[i + 1], webTextSize - i);
    if (p[i] == 141 || p[i] == 135) p[i] = 'c';
    if (p[i] == 161) p[i] = 's';
    if (p[i] == 190) p[i] = 'z';
  }
}

void drawDays(uint8_t n, bool fullUpdate)
{
  display.clearDisplay();
  display.setFont(DISPLAY_FONT_SMALL);
  display.setTextSize(2);
  uint8_t xPos = 0;
  uint8_t start = forecastList.startElement[-forecastList.shiftDay + n + 1];
  uint8_t end = forecastList.startElement[-forecastList.shiftDay + n + 2];
  for (int i = start; i < end; i++)
  {
    display.drawBitmap(xPos * 95 + 45, 70, gui.weatherIcon(atoi(forecastList.forecast[i].weatherIcon)), 50, 50,
BLACK);
    display.setCursor(xPos * 95 + 30, 140);
    display.print(epochToHuman(forecastList.forecast[i].timestamp).tm_hour, DEC);
    display.print(":00h");
    xPos++;
  }
  display.setTextSize(1);
  if (n > 0) display.fillTriangle(50, 10, 50, 40, 20, 25, BLACK);
  if (n < 4) display.fillTriangle(750, 10, 750, 40, 780, 25, BLACK);
  display.fillTriangle(10, 580, 30, 570, 30, 590, BLACK);

  display.setFont(DISPLAY_FONT);
  display.fillRect(4, 50, 792, 3, BLACK);
  display.setCursor(200, 35);
  display.print("Vremenska prognoza za ");
  display.print(epochToHuman(forecastList.forecast[start].timestamp).tm_mday, DEC);
  display.print('.');
  display.print(epochToHuman(forecastList.forecast[start].timestamp).tm_mon + 1, DEC);
  display.print(".,");
  display.print(DOW[epochToHuman(forecastList.forecast[start].timestamp).tm_wday]);

  for (int i = 0; i < 10; i++)
  {
    const uint8_t *iconList[] = {iconTemp, iconTlak, iconVlaga, iconVjetar, iconWindDir, iconClouds,
iconVisability, iconRainDrop, iconSnowflake, iconProbability};
    display.drawBitmap((i * 70) + 50, 550, iconList[i], 40, 40, BLACK);
  }
  drawSelectedGraph(selectedGraph, n);

  if (fullUpdate)
  {
    display.display();
  }
  else
  {
    display.partialUpdate();
  }
}

void drawSelectedGraph(uint8_t _graph, uint8_t _day)
{
  uint8_t start = forecastList.startElement[-forecastList.shiftDay + _day + 1];
  uint8_t end = forecastList.startElement[-forecastList.shiftDay + _day + 2];
  uint8_t _n = end - start;
  display.fillRect(4, 540, 792, 3, BLACK);
  display.fillRect((selectedGraph * 70) + 35, 540, 70, 3, WHITE);
  display.setFont(NULL);
  display.setTextSize(2);
  display.fillRect(0, 170, 800, 360, WHITE);
  switch (_graph)
  {
    case 0:
      gui.printAlignText("Temperatura zraka [C]", 400, 195, ALIGMENT_CENTERBOT);
```

```
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].temp), _n, sizeof(struct forecastWeatherHandle), 12, DATATYPE_FLOAT,
GRAPHSTYLE_LINE);
      break;
    case 1:
      gui.printAlignText("Tlak zraka [hPa]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].pressureGnd), _n, sizeof(struct forecastWeatherHandle), 12, DATATYPE_UINT16_T,
GRAPHSTYLE_LINE);
      break;
    case 2:
      gui.printAlignText("Relat. vlaznost zraka [%]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].humidity), _n, sizeof(struct forecastWeatherHandle), 10, DATATYPE_UINT8_T,
GRAPHSTYLE_LINE, 0, 100);
      break;
    case 3:
      gui.printAlignText("Brzina vjetra [m/s]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].windSpeed), _n, sizeof(struct forecastWeatherHandle), 12, DATATYPE_FLOAT,
GRAPHSTYLE_LINE, 0);
      break;
    case 4:
      gui.printAlignText("Smjer vjetra [stupnjevi]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].windDir), _n, sizeof(struct forecastWeatherHandle), 12, DATATYPE_UINT16_T,
GRAPHSTYLE_DOT, 0, 360);
      break;
    case 5:
      gui.printAlignText("Kolicina naoblake [%]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].clouds), _n, sizeof(struct forecastWeatherHandle), 10, DATATYPE_UINT8_T,
GRAPHSTYLE_COLUMN, 0, 100);
      break;
    case 6:
      gui.printAlignText("Vidljivost [m]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].visibility), _n, sizeof(struct forecastWeatherHandle), 12, DATATYPE_UINT16_T,
GRAPHSTYLE_LINE);
      break;
    case 7:
      gui.printAlignText("Kolicina padalina [mm]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].rain), _n, sizeof(struct forecastWeatherHandle), 12, DATATYPE_FLOAT,
GRAPHSTYLE_COLUMN, 0);
      break;
    case 8:
      gui.printAlignText("Kolicina snijega [mm]" , 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].snow), _n, sizeof(struct forecastWeatherHandle), 12, DATATYPE_FLOAT,
GRAPHSTYLE_COLUMN, 0);
      break;
    case 9:
      gui.printAlignText("Vjerojatnost oborina [%]", 400, 195, ALIGMENT_CENTERBOT);
      gui.drawGraph(130, 200, 600, 300, &(forecastList.forecast[start].timestamp),
&(forecastList.forecast[start].probability), _n, sizeof(struct forecastWeatherHandle), 10, DATATYPE_UINT8_T,
GRAPHSTYLE_COLUMN, 0, 100);
      break;
  }
}

uint8_t touchArea(int16_t tsX, int16_t tsY, int16_t x, int16_t y, int16_t w, int16_t h)
{
  if ((tsX >= x) && (tsY >= y) && (tsX < (x + w)) && (tsY < (y + h))) return 1;
  return 0;
}
```

PRILOG 14

Program vanjske jedinice

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * <h2><center>© Copyright (c) 2021 STMicroelectronics.
 * All rights reserved.</center></h2>
 *
 * This software component is licensed by ST under BSD 3-Clause license,
 * the "License"; You may not use this file except in compliance with the
 * License. You may obtain a copy of the License at:
 *                        opensource.org/licenses/BSD-3-Clause
 *
 ******************************************************************************
 */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "myStructs.h"
#include "glassLCD.h"
#include "SHT21.h"
#include "rtc.h"
#include "sleep.h"
#include "BMP180.h"
#include "Si1147.h"
#include "RF24.h"
#include "communication.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define NO_DEEP_SLEEP   0
#define DEEP_SLEEP      1
#define BMP180_ERROR    1
#define SHT21_ERROR     2
#define SI1147_ERROR    3
#define NRF24_ERROR     4

#define ALL_MEASUREMENTS        0b111111111
#define TEMP_MEASUREMENT        0b000000001
#define HUMIDITY_MEASUREMENT    0b000000010
#define PRESSURE_MEASUREMENT    0b000000100
#define UV_MEASUREMENT          0b000001000
#define LIGHT_MEASUREMENT       0b000010000
#define SOLAR_MEASUREMENT       0b000100000
#define WINDSPEED_MEASUREMENT   0b001000000
#define WINDDIR_MEASUREMENT     0b010000000
#define BATTERY_MEASUREMENT     0b100000000
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

I2C_HandleTypeDef hi2c1;
```

```
RTC_HandleTypeDef hrtc;

SPI HandleTypeDef hspi1;

TIM HandleTypeDef htim6;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
struct syncStructHandle syncStruct = {SYNC_HEADER};
struct measruementHandle weatherData;
struct measruementHandle currentWeatherData;
const char lcdTest[] = {"88888888"};
const char errStr[] = {"ERR-%03d"};
const char* windStr[] = {"N", "NNE", "NE", "ENE", "E", "ESE", "SE", "SSE", "S", "SSW", "SW", "WSW", "W", "WNW",
"NW", "NNW"};
volatile uint32_t interruptButton = 0;
volatile uint8_t alarmInterruptFlag = 0;
const uint16_t measurementTable[] = { TEMP_MEASUREMENT | HUMIDITY_MEASUREMENT, PRESSURE_MEASUREMENT, UV_MEASUREMENT
| LIGHT_MEASUREMENT, SOLAR_MEASUREMENT, WINDSPEED_MEASUREMENT, WINDDIR_MEASUREMENT, BATTERY_MEASUREMENT};
uint32_t sendInterval;
int16_t windDirCalibration = 0;
char lcdTemp[20];
uint8 t firstTimeSync = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_RTC_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */
void writeError(uint8_t _e, uint8_t _forceSleep);
void readWeatherData(struct measruementHandle *_w, uint16_t _flags);
void getSolarData(double *_energyJ, double *_energy);
uint32_t getADC(uint32_t _ch);
float getWindSpeed();
int16_t getWindDir(uint32_t _pin, int16_t _offset);
float getWindSpeed();
float getBatteryVoltage();
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */


  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
```

```
MX_I2C1_Init();
MX_RTC_Init();
MX_SPI1_Init();
MX USART2 UART Init();
MX ADC Init();
MX TIM6 Init();
/* USER CODE BEGIN 2 */
// Init LCD Driver
glassLCD_Begin();

// Test LCD (and wait for debugger to connect)
glassLCD_WriteData((char*) lcdTest);
glassLCD_SetDot(0b11111111);
glassLCD_WriteArrow(0b11111111);
glassLCD_Update();
HAL_Delay(2000);

// Dummy ADC readings to calibrate ADC and remove ranks
getADC(ADC CHANNEL 0);
getADC(ADC_CHANNEL_1);
getADC(ADC_CHANNEL_4);

// Init Pressure & Temp sensor
if (!BMP180_Init()) writeError(BMP180_ERROR, DEEP_SLEEP);

// Init Humidity & Temp sensor
if (!SHT21_Init()) writeError(SHT21_ERROR, DEEP_SLEEP);

// Init Si1147 Sensor for ALS and UV
if (!Si1147_Init()) writeError(SI1147_ERROR, DEEP_SLEEP);

RF24 init(NRF24 CE GPIO Port, NRF24 CE Pin, NRF24 CSN GPIO Port, NRF24 CSN Pin);
if (!RF24_begin()) writeError(NRF24_ERROR, DEEP_SLEEP);

// Enable UV meas.
Si1147_SetUV();

// Set time on RTC
RTC SetTime(1609459200);

// Setup RF communication (speed, RF Channel, RF Power, rtc)
communication_Setup();

// Wait for sync signal from indoor station
uint8_t syncTimeout = 180;
uint8_t rxBuffer[32] = {0};
uint8_t syncSuccess = 0;
while (!syncSuccess && syncTimeout > 0)
{
  if (communication_Transmit(&syncStruct, sizeof(syncStruct), rxBuffer))
  {
    if (rxBuffer[0] == SYNC_HEADER)
    {
      memcpy(&syncStruct, rxBuffer, sizeof(syncStruct));
      syncSuccess = 1;
    }
  }
  char lcdTemp[9];
  sprintf(lcdTemp, "SYNC %3d", syncTimeout--);
  glassLCD_WriteData(lcdTemp);
  glassLCD_Update();
  HAL_Delay(1000);
}
// Flush all unsent data and power down the module
RF24 flush rx();
RF24 flush tx();
RF24_powerDown();

if (syncSuccess)
{
  struct tm hTime;

  RTC_SetTime(syncStruct.myEpoch);
  RTC_SetAlarmEpoch(syncStruct.sendEpoch, RTC_ALARMMASK_DATEWEEKDAY);
  sendInterval = syncStruct.sendEpoch - syncStruct.myEpoch;
  firstTimeSync = 1;

  glassLCD WriteData("SYNC OK");
  glassLCD Update();
  HAL Delay(1000);
```

```
    hTime = RTC_EpochToHuman(syncStruct.myEpoch);
    sprintf(lcdTemp, " %2d%02d%02d", hTime.tm_hour, hTime.tm_min, hTime.tm_sec);
    glassLCD_WriteData(lcdTemp);
    glassLCD_SetDot(0b00101000);
    glassLCD_Update();
    HAL_Delay(1000);
  }
  else
  {
    sendInterval = 600;
    RTC_SetAlarmEpoch(RTC_GetTime() + sendInterval, RTC_ALARMMASK_DATEWEEKDAY);
    glassLCD_WriteData("NO SYNC");
    glassLCD_Update();
    HAL_Delay(1000);
  }

  // Get North direction for wind direction calibration
  for (int i = 0; i < 5; i++)
  {
    sprintf(lcdTemp, "D CAL %d", 5 - i);
    glassLCD_WriteData(lcdTemp);
    glassLCD_Update();
    windDirCalibration += getWindDir(ADC_CHANNEL_1, 0);
    HAL_Delay(1000);
  }
  windDirCalibration /= 5;
  readWeatherData(¤tWeatherData, ALL_MEASUREMENTS);

  /* USER CODE END 2 */



  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  uint8_t k = 0;
  while (1)
  {
    uint8_t lcdDot = 0;
    uint8_t lcdArrow = 0;
    if (k == 0)
    {
      int16_t t = round(currentWeatherData.tempSHT * 10);
      int16_t h = round(currentWeatherData.humidity * 10);
      sprintf(lcdTemp, "%3d%01d %2d%01d", t / 10, abs(t % 10), abs(h / 10), abs(h % 10));
      lcdDot = 0b00100010;
      lcdArrow = 0b10000000;
    }
    if (k == 1)
    {
      uint16_t p = round(currentWeatherData.pressure * 10);
      sprintf(lcdTemp, "%4d%1d", abs(p / 10), abs(p % 10));
      lcdDot = 0b00010000;
      lcdArrow = 0b01000000;
    }
    if (k == 2)
    {
      int16_t uv = (currentWeatherData.uv * 10);
      int16_t vis = currentWeatherData.light;
      sprintf(lcdTemp, "%2d%1d %4d", abs(uv / 10), abs(uv % 10), vis);
      lcdDot = 0b01000000;
      lcdArrow = 0b00100000;
    }
    if (k == 3)
    {
      int energyJ = round(currentWeatherData.solarJ * 10);
      sprintf(lcdTemp, "%2d%1d %4d", abs(energyJ / 10), abs(energyJ % 10), (int) (currentWeatherData.solarW));
      lcdDot = 0b01000000;
      lcdArrow = 0b00010000;
    }
    if (k == 4)
    {
      int wind = round(currentWeatherData.windSpeed * 10);
      sprintf(lcdTemp, "%3d%1d", abs(wind / 10), abs(wind % 10));
      lcdDot = 0b00100000;
      lcdArrow = 0b00001000;
    }
    if (k == 5)
    {
      sprintf(lcdTemp, "%3s %3d", windStr[(int) ((currentWeatherData.windDir / 22.5) + 0.5) % 16],
currentWeatherData.windDir);
      lcdArrow = 0b00000100;
```

```
}
if (k == 6)
{
  uint16_t batt = round((currentWeatherData.battery) * 100);
  struct tm t = RTC_EpochToHuman(RTC_GetTime());
  sprintf(lcdTemp, "%2d%02d %1d%02d", t.tm_hour, t.tm_min, batt / 100, abs(batt % 100));
  lcdArrow = 0b00000010;
  lcdDot = 0b01000100;
}
glassLCD_WriteData(lcdTemp);
glassLCD_SetDot(lcdDot);
glassLCD_WriteArrow(lcdArrow);
glassLCD_Update();
Sleep_LightSleep();

if (alarmInterruptFlag == 1)
{
  alarmInterruptFlag = 0;
  readWeatherData(&weatherData, ALL_MEASUREMENTS);
  currentWeatherData = weatherData;
  uint8_t dataSent = 0;

  if (firstTimeSync)
  {
    RF24_powerUp();
    communication_Setup();
    uint8_t rxBuffer[32] = {0};
    uint8_t sendTimeout = 25;
    struct data1StructHandle data1 = {DATA1_HEADER};
    struct data2StructHandle data2 = {DATA2_HEADER};
    void* dataStructList[2] = {&data1, &data2};
    size_t dataStructListSize[2] = {sizeof(data1), sizeof(data2)};
    data1.uv = weatherData.uv * 100;
    data1.windDir = weatherData.windDir;
    data1.tempSHT = weatherData.tempSHT;
    data1.tempSoil = weatherData.tempSoil;
    data1.humidity = weatherData.humidity;
    data1.pressure = weatherData.pressure;
    data1.light = weatherData.light;
    data1.windSpeed = weatherData.windSpeed;
    data2.rain = weatherData.rain;
    data2.battery = weatherData.battery;
    data2.epoch = weatherData.epoch;
    data2.solarJ = weatherData.solarJ;
    data2.solarW = weatherData.solarW;
    while (dataSent < 2 && sendTimeout > 0)
    {
      if (communication_Transmit(dataStructList[dataSent], dataStructListSize[dataSent], rxBuffer))
      {
        if (rxBuffer[0] == SYNC_HEADER)
        {
          dataSent++;
          memcpy(&syncStruct, rxBuffer, sizeof(syncStruct));
        }
      }
      char temp[10];
      sprintf(temp, "SEND %d", sendTimeout--);
      glassLCD_WriteData(temp);
      glassLCD_Update();
      HAL_Delay(1000);
    }
    RF24_flush_rx();
    RF24_flush_tx();
    RF24_powerDown();
  }
  if (dataSent == 0)
  {
    RTC_SetAlarmEpoch(RTC_GetTime() + sendInterval - 25,
        RTC_ALARMMASK_DATEWEEKDAY);
  }
  else
  {
    RTC_SetTime(syncStruct.myEpoch);
    RTC_SetAlarmEpoch(syncStruct.sendEpoch, RTC_ALARMMASK_DATEWEEKDAY);
    sendInterval = syncStruct.sendEpoch - syncStruct.myEpoch;
  }
}

if (interruptButton & GPIO_PIN_8)
{
  interruptButton &= ~(GPIO_PIN_8);
```

```
      k++;
      k = k % 7;
    }

    if (interruptButton & GPIO_PIN_1)
    {
      interruptButton &= ~(GPIO_PIN_1);
      readWeatherData(¤tWeatherData, measurementTable[k]);
    }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC ClkInitTypeDef RCC ClkInitStruct = {0};
  RCC PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Configure LSE Drive Capability
  */
  HAL PWR EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_HIGH);
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSE
                              |RCC OSCILLATORTYPE LSE;
  RCC OscInitStruct.HSEState = RCC HSE ON;
  RCC OscInitStruct.LSEState = RCC LSE ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC ClkInitStruct.SYSCLKSource = RCC SYSCLKSOURCE HSE;
  RCC ClkInitStruct.AHBCLKDivider = RCC SYSCLK DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2|RCC_PERIPHCLK_I2C1
                              |RCC_PERIPHCLK_RTC;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
  PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSE;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC_Init(void)
{

  /* USER CODE BEGIN ADC Init 0 */

  /* USER CODE END ADC_Init 0 */
```

```
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC Init 1 */

  /* USER CODE END ADC Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc.Instance = ADC1;
  hadc.Init.OversamplingMode = ENABLE;
  hadc.Init.Oversample.Ratio = ADC OVERSAMPLING RATIO 16;
  hadc.Init.Oversample.RightBitShift = ADC RIGHTBITSHIFT 4;
  hadc.Init.Oversample.TriggeredMode = ADC_TRIGGEREDMODE_SINGLE_TRIGGER;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV64;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.SamplingTime = ADC_SAMPLETIME_12CYCLES_5;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.DataAlign = ADC DATAALIGN RIGHT;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.EOCSelection = ADC EOC SINGLE CONV;
  hadc.Init.Overrun = ADC OVR DATA PRESERVED;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerFrequencyMode = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error Handler();
  }
  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_1;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_4;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC_Init 2 */

  /* USER CODE END ADC_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00000E14;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C ADDRESSINGMODE 7BIT;
  hi2c1.Init.DualAddressMode = I2C DUALADDRESS DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
```

```
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  RTC_TimeTypeDef sTime = {0};
  RTC_DateTypeDef sDate = {0};
  RTC_AlarmTypeDef sAlarm = {0};

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */
  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
  hrtc.Init.SynchPrediv = 255;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutRemap = RTC_OUTPUT_REMAP_NONE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }

  /* USER CODE BEGIN Check_RTC_BKUP */

  /* USER CODE END Check_RTC_BKUP */

  /** Initialize RTC and set the Time and Date
  */
  sTime.Hours = 0;
  sTime.Minutes = 0;
  sTime.Seconds = 0;
  sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
  sTime.StoreOperation = RTC_STOREOPERATION_RESET;
  if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN) != HAL_OK)
  {
    Error_Handler();
  }
  sDate.WeekDay = RTC_WEEKDAY_MONDAY;
  sDate.Month = RTC_MONTH_JANUARY;
  sDate.Date = 1;
  sDate.Year = 0;

  if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BIN) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  /** Enable the Alarm A
  */
  sAlarm.AlarmTime.Hours = 0;
  sAlarm.AlarmTime.Minutes = 0;
  sAlarm.AlarmTime.Seconds = 0;
  sAlarm.AlarmTime.SubSeconds = 0;
  sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
  sAlarm.AlarmTime.StoreOperation = RTC STOREOPERATION RESET;
  sAlarm.AlarmMask = RTC ALARMMASK NONE;
  sAlarm.AlarmSubSecondMask = RTC ALARMSUBSECONDMASK ALL;
  sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
  sAlarm.AlarmDateWeekDay = 1;
  sAlarm.Alarm = RTC_ALARM_A;
  if (HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, RTC_FORMAT_BIN) != HAL_OK)
  {
    Error Handler();
  }
  /* USER CODE BEGIN RTC_Init 2 */

  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1 Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI TIMODE DISABLE;
  hspi1.Init.CRCCalculation = SPI CRCCALCULATION DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6 Init 1 */

  /* USER CODE END TIM6 Init 1 */
  htim6.Instance = TIM6;
```

```
  htim6.Init.Prescaler = 60;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 65535;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
    HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, EN_3V3SW_Pin|NRF24_CE_Pin|NRF24_CSN_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(BAT_M_EN_GPIO_Port, BAT_M_EN_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PB1 */
  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
  /*Configure GPIO pin : EN_3V3SW_Pin */
  GPIO_InitStruct.Pin = EN_3V3SW_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(EN_3V3SW_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : BAT_M_EN_Pin */
  GPIO_InitStruct.Pin = BAT_M_EN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(BAT_M_EN_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : WS_DIN_Pin */
  GPIO_InitStruct.Pin = WS_DIN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(WS_DIN_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : NRF24_CE_Pin NRF24_CSN_Pin */
  GPIO_InitStruct.Pin = NRF24_CE_Pin|NRF24_CSN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : SI1147_INT_Pin */
  GPIO_InitStruct.Pin = SI1147_INT_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(SI1147_INT_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_1_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);

  HAL_NVIC_SetPriority(EXTI4_15_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(EXTI4_15_IRQn);

}

/* USER CODE BEGIN 4 */
void writeError(uint8_t _e, uint8_t _forceSleep)
{
  char _c[9];
  sprintf(_c, errStr, _e);
  glassLCD_WriteData(_c);
  glassLCD_Update();
  HAL_Delay(50);
  if (_forceSleep) HAL_PWR_EnterSTANDBYMode();
}

void readWeatherData(struct measruementHandle *_w, uint16_t _flags)
{
  if (_flags & TEMP_MEASUREMENT) _w->tempSHT = SHT21_ReadTemperature();
  if (_flags & HUMIDITY_MEASUREMENT) _w->humidity = SHT21_ReadHumidity();
  if (_flags & PRESSURE_MEASUREMENT)
  {
    _w->pressure = 0;
    for (int i = 0; i < 4; i++)
    {
      _w->pressure += BMP180_ReadPressure();
    }
    _w->pressure /= 4;
  }
  if (_flags & SOLAR_MEASUREMENT) getSolarData(&(_w->solarJ), &(_w->solarW));
  if (_flags & UV_MEASUREMENT)
  {
    Si1147_ForceUV();
     w->uv = Si1147 GetUV();
  }
  if (_flags & UV_MEASUREMENT)
  {
```

```
    // Force the measurement of UV but it also measures visible light
    Si1147_ForceUV();
    _w->light = Si1147_GetVis() * 0.282 * 16.5;
  }
  if (_flags & WINDSPEED_MEASUREMENT) _w->windSpeed = getWindSpeed();
  if (_flags & WINDDIR_MEASUREMENT) _w->windDir = getWindDir(ADC_CHANNEL_1, windDirCalibration);
  if (_flags & BATTERY_MEASUREMENT) _w->battery = getBatteryVoltage();
  _w->epoch = (uint32_t)RTC_GetTime();
}

// Read solar data in Joules/cm2 and W/m2
void getSolarData(double *_energyJ, double *_energy)
{
  // Turn on supply to the OpAmp
  HAL_GPIO_WritePin(EN_3V3SW_GPIO_Port, EN_3V3SW_Pin, GPIO_PIN_SET);
  HAL_Delay(10);

  uint32_t rawAdc;
  double voltage;
  double current;

  // Get the measurement for  the solar cell
  rawAdc = getADC(ADC_CHANNEL_0);

  // Turn off supply to the OpAmp
  HAL_GPIO_WritePin(EN_3V3SW_GPIO_Port, EN_3V3SW_Pin, GPIO_PIN_RESET);

  // Calculations for voltage, current and energy
  voltage = (rawAdc / 4095.0 * 3.3) - 0.004;

  if (voltage < 0) voltage = 0;

  current = voltage / (220 / 4.7);
  *_energy = current / 45E-3 * 1000;
  *_energyJ = (1 / 1E4) * (*_energy) * 600;
}

uint32_t getADC(uint32_t  ch)
{
  uint32_t _result;
  ADC_ChannelConfTypeDef ch = {0};

  // Calibrate ADC
  HAL_ADCEx_Calibration_Start(&hadc, ADC_SINGLE_ENDED);

  // Select the channel (in order to work properly, on all channels ranks have to be set on RANK_NONE!)
  ch.Channel = _ch;
  ch.Rank = ADC_RANK_CHANNEL_NUMBER;
  HAL_ADC_ConfigChannel(&hadc, &ch);

  // Start ADC conversion of selected channel
  HAL_ADC_Start(&hadc);

  // Wait for conversion to be complete
  HAL_ADC_PollForConversion(&hadc, 1000);

  // Get the RAW ADC value
  _result = HAL_ADC_GetValue(&hadc);

  // Stop the ADC
  HAL_ADC_Stop(&hadc);

  // Remove channel from rank
  ch.Rank = ADC_RANK_NONE;
  HAL_ADC_ConfigChannel(&hadc, &ch);

  // Return the result
  return _result;
}

float getWindSpeed()
{
  // Turn on supply to the wind speed sensor
  HAL_GPIO_WritePin(EN_3V3SW_GPIO_Port, EN_3V3SW_Pin, GPIO_PIN_SET);
  HAL_Delay(5);

  // Activate the timer
  HAL_TIM_Base_Start(&htim6);

  // Get inital state of the pin
  uint8_t _initState = HAL_GPIO_ReadPin(WS_DIN_GPIO_Port, WS_DIN_Pin);
```

```
  // Get the current sysTick time (needed for timeout)
  uint32_t _timeout = HAL_GetTick();
  uint32 t  period = 0;

  // Wait for the edge of the signal (doesn't matter if is rising or falling)
  while ((HAL_GPIO_ReadPin(WS_DIN_GPIO_Port, WS_DIN_Pin) == _initState) && ((HAL_GetTick() - _timeout) < 1000));
  if ((HAL_GetTick() - _timeout) >= 1000) return 0;

  // Reset the timer counter and measure the time until next edge of the signal
  htim6.Instance->CNT = 0;
  while ((HAL_GPIO_ReadPin(WS_DIN_GPIO_Port, WS_DIN_Pin) != _initState) && (htim6.Instance->CNT < 65530));
  _period += (htim6.Instance->CNT);

  // Reset the timer counter and measure the time until next edge of the signal
  htim6.Instance->CNT = 0;
  while ((HAL_GPIO_ReadPin(WS_DIN_GPIO_Port, WS_DIN_Pin) == _initState)&& (htim6.Instance->CNT < 65530));
   period += (htim6.Instance->CNT);

  // Stop the timer
  HAL_TIM_Base_Stop(&htim6);

  // Turn on supply to the wind speed sensor
  HAL GPIO WritePin(EN 3V3SW GPIO Port, EN 3V3SW Pin, GPIO PIN RESET);

  // Calculate the frequency in hertz and return the result
  return ((float)(1 / (_period * 1E-6 * 15.25))) / 9.14;
}

int16_t getWindDir(uint32_t _pin, int16_t _offset)
{
  // Turn on the supply to the wind direction sensor (AS5600)
  HAL_GPIO_WritePin(EN_3V3SW_GPIO_Port, EN_3V3SW_Pin, GPIO_PIN_SET);
  HAL_Delay(10);

  int16_t _voltage = getADC(_pin);
  int16 t  angle;
   voltage = 1 / 4095.0 *  voltage * 3300;
   angle = (int16 t) ((360.0 / 3300.0) *  voltage);
  _angle = _angle - _offset;
  if (_angle < 0)
    _angle = 360 + _angle;

  // Turn off the supply to the wind direction sensor (AS5600)
  HAL_GPIO_WritePin(EN_3V3SW_GPIO_Port, EN_3V3SW_Pin, GPIO_PIN_RESET);

  return _angle;
}

float getBatteryVoltage()
{
  float _result;
  HAL_GPIO_WritePin(BAT_M_EN_GPIO_Port, BAT_M_EN_Pin, GPIO_PIN_SET);
  HAL_Delay(10);
  _result = getADC(ADC_CHANNEL_4) * 1 / 4095.0 * 3.3 * 2;
  HAL_GPIO_WritePin(BAT_M_EN_GPIO_Port, BAT_M_EN_Pin, GPIO_PIN_RESET);
  return _result;
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  interruptButton |= GPIO_Pin;
}

void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
{
  alarmInterruptFlag = 1;
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error Handler Debug */
}
```

```c
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
   tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```