УНИВЕРЗИТЕТ У БЕОГРАДУ

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Бежични сензорски систем

Дипломски рад

Ментор:

Др Слободан Савић, доцент

Кандидат:

Лука Марковић 0171/2015

Београд, Фебруар 2021.

САДРЖАЈ

СПИСАК СЛИКА

СПИСАК ТАБЕЛА

1. Увод

У овом раду ћемо објаснити како је могуће направити *LoRa* бежичну сензорску мрежу помоћу *Arduino* окружења, са једним или више сензорских предајника, у даљем тексту названи *Node*-ови, и главним пријемником, у даљем тексту названим *Master*. Овај рад представља солидну основу за развој некомерцијалних и комерцијалних, финансијски приступачних и ефикасних *IoT* система за употребу у кућним, али и индустријским условима. Читав систем, укључујући софтверски и хардверски део развијени су од почетка, а читав систем замишљен је као модуларан, тако да се његове могућности могу једноставно проширити и прилагодити новим потребама.

У другом поглављу ћемо објаснити шта је тачно *LoRa* технологија, што ће дати увид у разлоге зашто смо се определили за коришћење уређаја базираних на истој. Навешћемо где је нашла примену у модерном свету, као и где је могућа примена у будућности. Овде ће детаљније бити описан модул који је коришћен у овом раду, заједно са библиотеком за његово управљање уз опис основних и напредних функција и процедура.

У трећем поглављу ћемо објаснити шта је тачно *Arduino*, као и зашто смо се определили за његово коришћење. Размотрићемо могућности за повезивање *Arduino* уређај са *LoRa* модулом, мере опреза и проблеме на које се приликом тога може наићи, као и начине како су ти проблеми превазиђени у овом раду. Приказаћемо и основне две технологије комуникације које користе *Arduino* уређаји, као и разлоге за избор једне од њих у оквиру овог рада.

У четвртом поглављу ћемо описати целокупан сензорски систем, детаљно разматрајући све сензорске модуле, као и остале модуле који су коришћени при његовој изради. Објаснићемо како је могуће направити минималистичко микроконтролерско окружење у циљу унапређења перформанси и смањења потрошње електричне енергије. Приказаћемо и готове системе до израда штампаних плочица, начин њиховог конструисања, као и начин конструисања штампаних плочица.

У петом поглављу ћемо финално приказати рад целокупног система, након чега ћемо приказати резултате мерења, у графичкој форми, приликом тестирања у кратком, као и у дужем временском периоду.

Комплетан рад закључујемо у поглављу 6.

Уз овај рад, на самом крају документа, се налази додатак, у коме су дати сви C/C++ кодови који су коришћени у овом раду.

2. LoRa

У овом поглављу ћемо објаснити шта је то заправо *LoRa* стандард и где је тај стандард нашао примену у модерним телекомуникационим и информационим системима. Такође ћемо детаљније описати *LoRa* модул који је био коришћен у овом раду, као и детаље у вези са библиотеком која је коришћена за комуникацију између *LoRa* модула и *Arduino* модула.

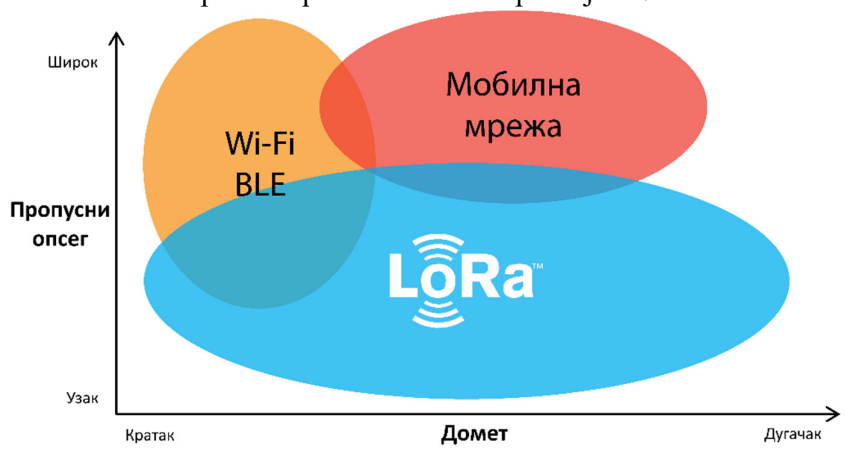
2.1. Шта је LoRa?

LoRa [1], скраћено од *Long Range* (срп. дугачак домет) је модулациона техника у проширеном спектру изведена из технологије чирп проширеног спектра (енг. *chirp spread spectrum technology*). *LoRa* уређаји, као и одговарајући *LoRaWAN* протокол, омогућавају ефикасне и елегантне *IoT* примене које решавају неке од највећих изазова данашњице, о којима ће бити речи у следећем потпоглављу.

Само неке о погодности које нуди *LoRa* протокол и одговарајући уређаји су:

- Велики домет;
- Висока енергетска ефикасност;
- Висока безбедност комуникације;
- Стандардизација;
- Једноставна геолокација;
- Висока мобилност;
- Задовољавајући капацитет преноса података и
- Ниска цена.

Један од главних разлога коришћења *LoRa* уређаја јесте њихова могућност искоришћавања слободних опсега конвенционалних ћелијских инфраструктурних технологија, као и Wi-Fi/BLE технологија. На слици 1 графички је приказано који од стандардних систем бежичних комуникација (Wi-Fi и мобилна телефонија) је ефикаснији и чешће коришћен у зависности од захтеваног протока података и растојања између корисника, као и поређење са *LoRa* стандардом. Са те слике може се закључити да, када год није потребно остварити бежичне везе великог капацитета, *LoRa* стандард може представљати оптималан избор за покривање великих растојања.



Слика 1. Оптимални домети и пропусни опсези за различите технологије бежичног приступа [1].

2.2. Примена

У овом потпоглављу ћемо укратко описати неке основне примене код којих је *LoRa* стандард нашао своје место. Иако је то стандард који је потекао из струке електро инжењера, његове примене могу бити од користи и разним другим струкама, а један од циљева овог рада је и представљање овог стандардна широј заједници и покушај његовог омасовљавања у привреди. Само неки од оваквих примера су савремена пољопривреда пред коју се сваким даном постављају све захтевнији циљеви у смислу ефикасности и висине прихода по површини обрадиве земље, а због тога је неопходно стално и прецизно мерење разних параметара земљишта и атмосфере на великим географским површинама.

2.2.1. Паметна пољопривреда, градови и здравство

Како би се повећала ефикасност и конкурентност тренутне пољопривреде, максимизирали приходи по површини обрадиве земље и смањио утицај на животну средину, у савременој пољопривреди потребно је повећати аутоматизацију, као што је приказано у [2]. У ту сврху потребно је у реалном времену мерити велики број параметара, као што су влажност и температура земље, а сензори би по правилу требало да буду распоређени на великом простору, као што је предложено у [3] и [4]. За ове примене *LoRa* стандард чини се као идеално решење, које омогућава бежичан приступ великог домета, мале потрошње електричне енергије и ниске цене. Након прикупљања података исти се могу централизовано обрађивати, а након тога могу се донети одлуке о току даљих поступака засноване на овим мерењима.

Осим примена у руралним срединама, *LoRa* протокол може наћи примену и у урбаним срединама, као што је приказано у [5] за аутоматизацију организације паркинг сервиса, за шта је обично потребно доста различитих информација, а већина тих информација могу се пренети и средствима комуникације малог протока. Још неки од оваквих примера су бежично повезивање градских сервиса као што су осветљење, паркинг сервис, санитарне услуге чишћења града и отклањање отпада, без потребе да се до сваког од тих места спроводе посебни бакарни или оптички водови, а познавање ових података у реалном времену може се искористити за оптимизацију ресурса инфраструктуре и сервиса града. *LoRa* протокол може се искористити и за *IoT* решења у оквиру здравства, где се може вршити целодневни надзор стања пацијента који ће моћи да се слободно креће носећи са собом уређај величине мобилног телефона, при чему ће тајност података бити загарантована.

2.2.2. Остале примене

У претходном потпоглављу наведене су само неке од основних примена *LoRa* система, али је битно нагласити да се ови системи користе и у паметним мерним системима за мониторинг електричне потрошње, потрошње воде и гаса, као и осталих природних ресурса. Могу се користити и у било којим складиштеним просторима за надгледање стања и залиха робе.

Осим такозваних паметних градова, који су на нивоу друштвене заједнице, могуће је интегрисати *LoRa* системе на нивоу појединачних домова у виду безбедносних система, као и такозваних паметних система за рестрикцију прекомерне потрошње ресурса.

2.3. *RA-02 LoRa модул*

LoRa модул под ознаком „RA-02“ је изабран као основа за сензорску мрежу која ће бити описана и направљена у оквиру овог рада. У овом потпоглављу ћемо дати кратак опис основних карактеристика овог модула.

Модул RA-02 је заснован на SX1278 чипу чији је произвођач компанија *Semtech*, светски лидер у области *LoRa* система. Ова серија чипова покрива фреквенцијски опсег од 410 MHz до 525 MHz, што омогућава велике домете. Како је реч о технологији преноса у проширеном спектру, ови модули су изузетно отпорни на интерференцију и подржавају велики број модулација, као што су FSK, GFSK, MSK, GMSK и OOK. Изглед једног оваквог модула приказан је на слици 2, а као референца о његовим димензијама може се искористити U.FL конектор који се налази у доњем-левом углу модула.

Као и већина *LoRa* модула, модул RA-02 се повезује са другим уређајима помоћу SPI стандарда комуникације, што га чини погодним за управљање помоћу различитих микроконтролера, као што су *Arduino*, STM32, *Teensy* и њима слични.

Слика 2. *LoRa* модул под ознаком RA-02 [6].

2.4. *Преглед коришћене библиотеке*

Библиотека која је коришћена за управљање *LoRa* модулом RA-02 је *Arduino-LoRa* библиотека од корисника *sandeepmistry* на *GitHub* сервису. Ова библиотека пружа основне могућности предаје и пријема пакета помоћу *LoRa* уређаја базираних на *Semtech*-овим *SX12XX* чиповима. Такође, постоје и неке напредне функције на нивоу интеракције са повезаним *Arduino* микроконтролером. Битно је наговестити да у овом раду нећемо улазити у детаље како је библиотека формулисана, нити ћемо улазити детаљно у архитектуру самог модула. У оквиру овог рада биће описан начин коришћења ове библиотеке и само они детаљи који су за то неопходни биће разматрани, а заинтересован читалац упућује се на додатну литературу.

2.4.1. Опис основних функција

Основне функције у склопу наведене библиотеке дате су у следећој табели, као и њихов кратак опис.

Табела 1. Основне функције коришћене библиотеке.

Назив функције	*Опис функције*
`#include <LoRa.h>`	Неопходан „хедер“ фајл како би *Arduino* могао да успостави комуникацију са библиотеком.
`LoRa.begin(frequency)`	Улазни параметар функције је централна фреквенција одговарајућег модула. Функција враћа вредност 1 ако је веза успешно успостављена, у супротном враћа 0.
`LoRa.end()`	Крај позивања библиотеке.
`LoRa.beginPacket()` `LoRa.beginPacket(implicitHeader)`	Започиње секвенцу слања пакета. Враћа вредност 1 ако је могуће преносити поруку у датом тренутку, у супротном враћа 0.
`LoRa.write(byte)` `LoRa.write(buffer, length)`	Уписивање информација у пакет. Помоћу прве функције се уписује само један бајт. Помоћу друге функције се уписује низ бајтова максималне дужине 256.
`LoRa.endPacket()` `LoRa.endPacket(async)`	Зауставља секвенцу слања пакета. Враћа вредност 1 ако је секвенца успешно заустављена, у супротном враћа 0.
`LoRa.parsePacket()` `LoRa.parsePacket(size)`	Проверава да ли је пристигао пакет. Повратна вредност је величина пакета у бајтовима или 0 ако није примљен пакет.

2.4.2. Опис напредних функција

Напредније функције у склопу посматране библиотеке дате су у наставку. Нећемо их детаљније описивати осим ако се појаве у даљем тексту, а мотивација за њихово набрајање је помоћ у раду корисницима ове библиотеке који би желели да самостално направе свој *LoRa* систем.

- `LoRa.setPins(ss, reset, dio0)` – подешавање пинова помоћу којих ће се обављати комуникација са модулом. Мора се позвати пре LoRa.begin() функције;
- `LoRa.setSPI(spi)` – подешавање коришћења неког другог SPI интерфејса, ако је доступан;
- `LoRa.setSPIFrequency(frequency)` – подешава фреквенцију SPI интерфејса;
- `LoRa.onTxDone(onTxDone)` – *interrupt* који се извршава након слања пакета;
- `LoRa.onReceive(onReceive)` – функција која помоћу DIO0 пина прави *interrupt* рутину сваки пут када се прими нови пакет;
- `LoRa.receive()` – ставља модул у режим константног пријема;
- `LoRa.packetRssi()` – враћа вредност RSSI (енг. *received signal strength indicator*) примљеног пакета;
- `LoRa.packetSnr()` – враћа процењену вредност SNR (енг. *signal-to-noise ratio*) примљеног пакета у dB;
- `LoRa.packetFrequencyError()` – враћа фреквенцијску грешку између централне фреквенције пријемног модула и долазног *LoRa* сигнала;
- `LoRa.available()` – враћа број бајта доступан за читање;
- `LoRa.peek()` – враћа следећи бајт у пакету, без очитавања, или враћа вредност -1 ако не постоји следећи бајт за читање;

- `LoRa.read()` – чита следећи бајт у пакету или враћа вредност -1 ако не постоји следећи бајт за читање.
- `LoRa.idle()` – подешава *standby* режим рада;
- `LoRa.sleep()` – ставља модул у режим ниске потрошње;
- `LoRa.setTxPower(txPower)` – подешава снагу на предаји;
- `LoRa.setFrequency(frequency)` – подешава централну фреквенцију;
- `LoRa.setSpreadingFactor(spreadingFactor)` – подешавање броја послатих чирпова у секунди;
- `LoRa.setSignalBandwith(signalBandwith)` – подешавање пропусног опсега;
- `LoRa.setCodingRate4(codingRateDenominator)` – подешавање брзине кодирања;
- `LoRa.setPreambleLength(preambleLength)` – подешавање дужине преамбуле;
- `LoRa.setSyncWord(syncWord)` – подешавање синхроне речи;
- `LoRa.enableCrc() / LoRa.disableCrc()` – укључивање/искључивање CRC теста;
- `LoRa.enableInvertIQ() / LoRa.disableInvertIQ()` – укључивање/искључивање I и Q грана.

Број напредних функција је значајно дужи од броја основних функција, све ове функције нису биле неопходне за израду овог рада, али за израду неких других система могу бити од користи.

3. Arduino

3.1. Шта је Arduino?

Arduino [7] је бесплатна, јавно доступна електронска платформа заснована на приступачном и стандардизованом хардверу и софтверу. *Arduino* микроконтролери, који сачињавају хардверски део, лако се управљају помоћу *Arduino* програмског језика, који је заснован на *Wiring* платформи, као и *Arduino* софтвера, базираног на *Processing* платформи.

Arduino не постоји као јединствен уређај, већ је сачињен од пуно различитих верзија микроконтролерских чипова, међу којима је један од најпознатијих ATmega328P [8] од компаније *Atmel*. Овај чип је уједно и један од најчешће коришћених чипова у најпопуларнијим *Arduino* серијама, UNO і NANO. У новијим верзијама *Arduino* микроконтролерских плоча користе се и микроконтролери других произвођача, међутим *Atmel*-ови чипови се и дан данас најчешће сусрећу у пракси.

3.2. Зашто Arduino?

Arduino уређаји су знатно приступачнији у односу на друге микропроцесорксе плоче, што их чини популарним како међу хоби електроничарима, тако и међу професионалцима.

Софтвер који контролише микроконтролерске плоче је доступан на свим популарнијим платформама који се користе у данашњици (*Windows*, *MacOS*, *Linux*), док је већина других микроконтролерских екосистема лимитирана само на Windows платформу.

Једноставан и лако разумљив програмски интерфејс за почетнике и познат програмски језик за професионалце само су још неки од разлога тако велике популарности *Arduino* платформе, а овоме дефинитивно доприноси и бесплатан софтверски пакет, велики број форума и добра документација, што читав овај екосистем чини изузетно привлачним окружењем за тестирање и развој нових микропроцесорских система.

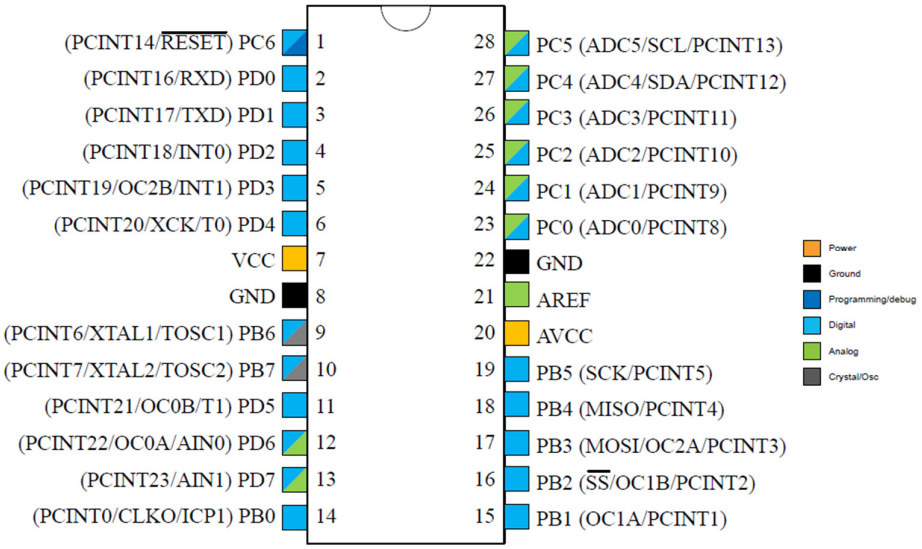
3.3. Повезивање Arduino и LoRa модула

Да би било могуће управљати *LoRa* модулима помоћу *Arduino* система неопходно је правилно међусобно повезивање ова два уређаја.

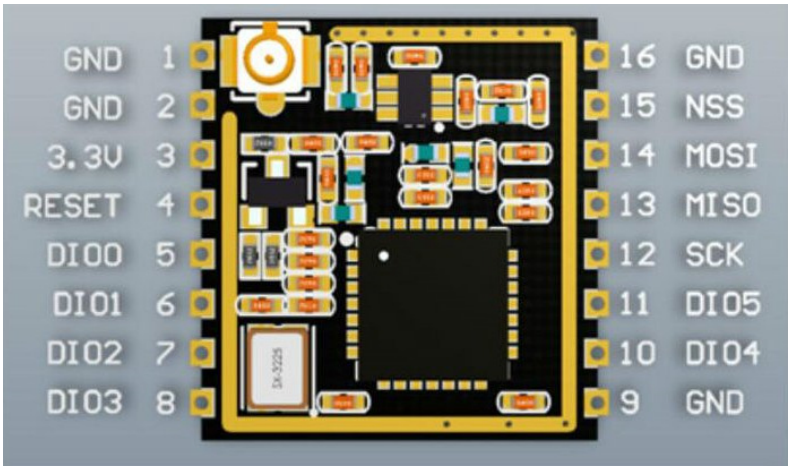
Пинови 12, 13 и 14 *LoRa* модула (слика 4) редом се повезују на пинове 19, 18 и 17 микроконтролера ATmega328P (слика 3). Пинови NSS и RESET се везују на било која два дигитална пина микроконтролера назначена светло плавом бојом на слици 3. Пин DIO0 (енг. *digital input-ouput*) се мора повезати на пинове 4 или 5 микроконтролера, тј. мора бити повезан на било који пин који је предодређен као INTERRUPT. Ово омогућава додатне опције као што су буђење из SLEEP режима рада, прекидање тренутних команди зарад извршавања неких других операција и томе слично. Пинови који су назначени као DIO*X*, где је *X* било који број осим 0, нису неопходни за повезивање нити су разматрани у овом раду.

Важно је напоменути да је *LoRa* модул уређај који се искључиво напаја са 3,3 V, као и да користи логичке нивое који су предвиђени за овај напон напајања (напони за дигиталне сигнале су у опсегу од 0 до 3,3 V). Микроконтролер ATmega328P се може напајати са 5 V или 3,3 V, од чега директно зависе и нивои логичких нивоа које ће користити. У овом раду

микроконтролер се напаја са 5 V, тако да користи логичке нивое прилагођене том напону. Овим се ствара потреба за коришћењем посебног дигиталног кола LLS (енг. *Logic Level Shifter*) које има за улогу да логичке нивое са 5 V спусти на 3,3 V, односно да логичке нивое са 3,3 V подигне на 5 V. За потребе овог рада коришћен је бидирекциони LLS модул TX0108.



Слика 3. Конфигурација пинова на микроконтролеру ATmega328P [9].



Слика 4. Конфигурација пинова на модулу LoRa RA-02 [10].

3.4. SPI комуникација

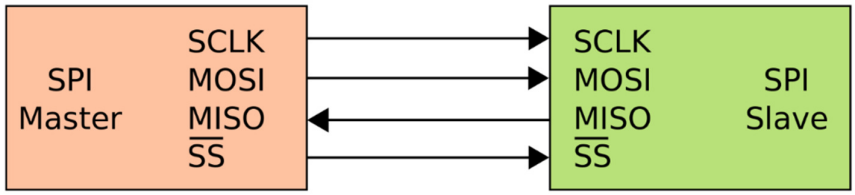
SPI [11] (енг. *Serial Peripheral Interface*) је синхрони серијски комуникациони интерфејс који се користи за комуникацију уређаја на кратким раздаљинама, превасходно у *embedded* системима и представља *de facto* стандард.

SPI уређаји су засновани на *master-slave* архитектури, при чему користе само један *master* и један или више *slave* уређаја, који се бирају на основу *slave-select*, односно *chip-select* линија. За разлику од SSI [12] (енг. *Synchronous Serial Interface*) који је, такође, синхрони серијски комуникациони протокол и врши комуникацију у једном симплекс режиму, SPI

врши комуникацију у потпуном дуплекс режиму(енг. *full-duplex*), што дозвољава комуникацију у оба правца симултано.

SPI магистрала се састоји од четири логичка канала:

- SCKL (енг. *serial clock*);
- MOSI (енг. *master-out slave-in*);
- MISO (енг. *master-in slave-out*);
- SS (енг. *slave-select*).



Слика 5. Стандардна SPI конфигурација једног *master* и једног *slave* уређаја [13].

3.5. I^2C комуникација

Други стандард комуникације међу модулима у оквиру једне штампане плочице је I^2C [14] (енг. *Inter-Integrated Circuit*) који представља синхрону, небалансирану, серијску комуникациону магистралу која подржава више *master* и више *slave* уређаја. Примарно се користи за комуникацију на кратким раздаљинама или, такозвану, *intra-board* комуникацију, између интегрисаних кола, процесора, односно микроконтролера. Ова магистрала омогућава комуникацију ниским брзинама.

I^2C користи само две бидирекционе линије које раде у конфигурацији са отвореним колектором (енг. *open collector*) или отвореним дрејном (енг. *open drain*). Прва линија је SDA (енг. *Serial Data Line*), а друга линија је SCL (енг. *Serial Clock Line*). Обе ове линије се преко такозваних *pull-up* отпорника доводе на логичку јединицу. Референтни I^2C дизајн представља магистралу са *clock* (SCL) и *data* (SDA) линијама уз 7-битно адресирање. Магистрала има две улоге: *master* - који генерише *clock* и иницијализује комуникацију са *slave* уређајем или уређајима; *slave* - који прихвата *clock* и одговара на позив *master* уређаја. Постоје четири потенцијална режима рада за сваки уређај: *master* пренос, *master* пријем, *slave* пренос и *slave* пријем. Већина уређаја има једну улогу и највише два режима рада. У овом раду ћемо користи оба наведена комуникациона стандарда.

4. Сензорски систем

У овом поглављу ћемо описати целокупно хардверско решење овог рада. Поћи ћемо од прегледа сензорских и других модула који се користе, као и од прегледа микроконтролера. Такође ћемо приказати и дизајн шема и штампаних плоча (PCB, енг. *printed circuit board*) креираних у софтверском пакету *Altium Designer*. Детаљније ћемо описати неке од битнијих делова дизајна, као и неке од практичних недостатака који су примећени приликом тестирања и начин на који су те потешкоће превазиђене.

4.1. Преглед сензора

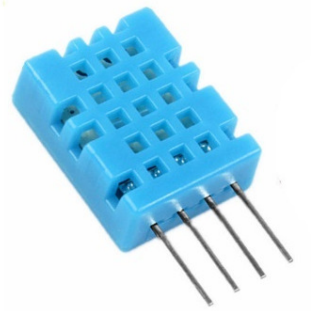
У овом потпоглављу ћемо описати све сензоре који су коришћени за реализацију овог пројекта. Биће изложено како се повезују са микроконтролером, као и које библиотеке свака од њих користи.

Важно је напоменути да су неки од модула физички измењени у односу на фабричко стање. О томе неће бити речи у овом поглављу, већ у једном од наредних где ћемо детаљно објаснити шта је измењено и из којег разлога.

4.1.1. Сензор температуре и влажности ваздуха DHT11

На слици 6 је приказан сензор температуре и влажности ваздуха DHT11. Конфигурација пинова са лева на десно је редом, VCC, DATA, NC (енг. *not connected*) и GND. Модул се може напајати са 3,3 V или са 5 V. У нашој примени, модул се напаја са 5 V.

DATA пин служи за комуникацију са микроконтролером и препоручљиво је да буде спојен са VCC пином преко отпорника минималне отпорности 4,7 kΩ. Отпорник је препоручен да би излаз из DATA показивао логичку јединицу када се не врше мерења. Овим се решава проблем такозваног „лебдећег“ пина (енг. *floating pin*).



Слика 6. Сензор температуре и влажности ваздуха DHT11 [15].

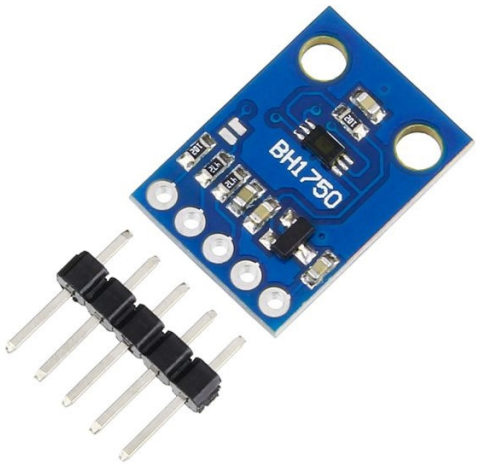
Овај модул је погодан за мерења температуре у опсега од 0°C до 50°C, уз максималну грешку приликом мерења од ±2 °C, и за мерење влажности ваздуха у опсегу од 20% од 80%, уз максималну грешку мерења од ±5 %.

За комуникацију са микроконтролером ATmega328P помоћу *Arduino* платформе коришћена је *DHT-sensor-library* од компаније *Adafruit*.

4.1.2. Сензор нивоа осветљења BH1750

На слици 7 је приказан сензор нивоа осветљења BH1750. Конфигурација пинова са лева на десно је редом, ADDR, SDA, SCL, GND и VCC. Модул се може напајати са 3,3 V или 5 V. У оквиру овог рада напаја се са 5 V.

ADDR пин се не користи у нашој примени, док се SDA и SCL пинови примењују при I^2C комуникацији.



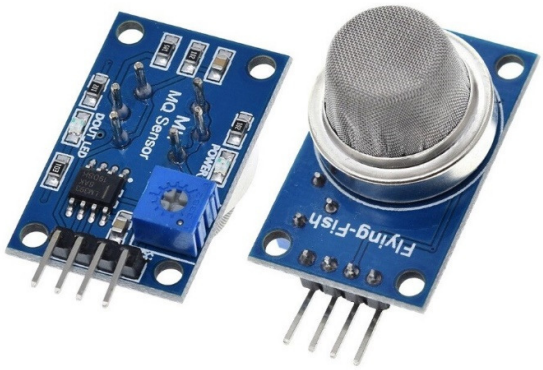
Слика 7. Сензор нивоа осветљења BH1750 [16].

Сензор нивоа осветљења BH1750 конвертује ниво осветљења у дигитални ниво у опсегу од 0 до 65534 lx, за разлику од фотоотпорника који ниво осветљења представљају аналогно у опсегу од 0 до VCC. Резолуцију очитавања је могуће подесити софтверски у ниско-резолуциони режим, са корацима од по 4 lx и високо-резолуциони режим, са корацима од по 1 lx. За рад са овим уређајем коришћена је библиотека BH1750 од корисника *claws* која се налази на *GitHub* сервису.

4.1.3. Сензор присутности штетних гасова MQ2

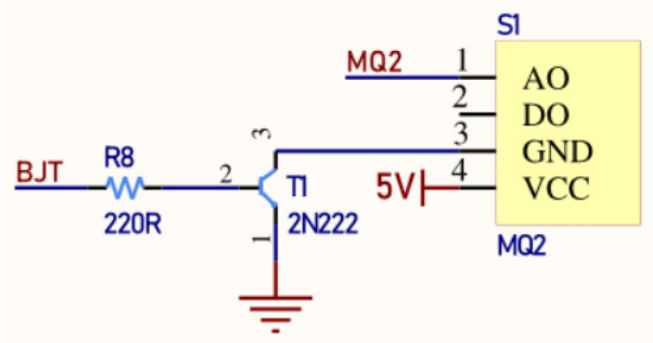
На слици 8 је приказан сензор присутности штетних гасова MQ2. Модул садржи четири пина: два за напајање, два за излазне податке. Могуће је читање података преко аналогног или дигиталног интерфејса. У нашем случају, читање смо вршили помоћу аналогног интерфејса, што нам је омогућило да избегнемо коришћење готових библиотека, већ да очитавање спроведемо директно. На тај начин, уз не тако велики додатни труд, избегли смо потенцијалне проблеме у вези са некомпатибилностима софтверских решења.

Када се очитавање врши помоћу аналогног интерфејса, резултати се бележе у вредностима од 0 до 5 V, где 0 представља ваздух без штетних гасова, док 5 V представља изузетно високу присутност штетних гасова. Модул такође садржи потенциометар чијом конфигурацијом се подешава осетљивост мерења.



Слика 8. Сензор присуства штетних гасова MQ2 [17].

Због релативно високе потрошње електричне енергије и немогућности једноставног пребацивања овог модула у такозвани *sleep* режим рада, напајање је вршено помоћу логички контролисаног биполарног транзистора (BJT), где је GND модула спојен на колектор транзистора 2N222, емитер је спојен на GND система, а база транзистора је спојена на један од дигиталних пинова микроконтролера, као што је приказано на слици 9. Променом стања на том дигиталном пину микроконтролера, односно променом напона база-емитер транзистора, могуће је укључити модул само када је неопходно вршити мерења. На овај начин значајно је смањена потрошња електричне енергије комплетног система.



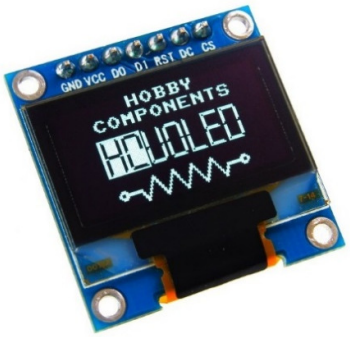
Слика 9. Шема повезивања модула MQ2 и модула Arduino којом се смањује потрошња електричне енергије.

4.2. Преглед осталих модула

4.2.1. OLED екран SSD1306

На слици 10 је приказан OLED екран SSD1306 који је коришћен за приказивање тренутних података које су достављене *Master* систему. Овај модул има резолуцију од 64 x 128 пиксела, а постоје две врсте ових модула које су функционално исте, али користе другачије комуникационе интерфејсе. Први користи I^2C интерфејс, док други користи SPI интерфејс. У нашем случају је коришћен дисплеј модул са SPI интерфејсом. Ово решење је доста сложеније и, као што ћемо видети касније, доводи до нежељених ометања у целокупном систему. Ми смо се определили за овај модул, пошто нам је само он био на

располагању, а потешкоће у вези са његовим функционисањем превазишли смо у оквиру овог рада.



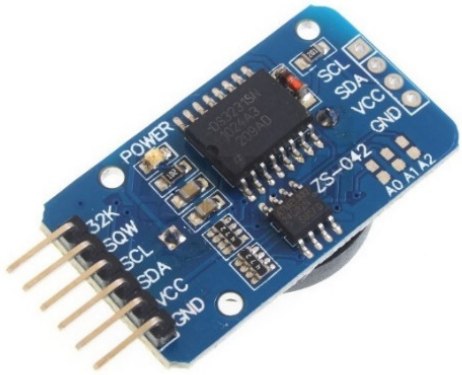
Слика 10. OLED екран SSD1306 [18].

Постоје две основне библиотеке које су распрострањене за рад са овим модулом. Прва је *Adafruit_SSD1306*, заједно са *Adafruit_GFX* библиотеком, док је друга *SSD1306Ascii* библиотека, од корисника *greiman* са сервиса *GitHub*. Прва библиотека је погодна за приказ различитих фонтова, цртање минијатурних графика, приказ слика и томе слично. Све ове могућности са собом носе високу цену у виду заузећа меморијског простора. Друга библиотека корисна је у случају када је потребно приказати искључиво текст, и са становишта заузећа рачунарских ресурса значајно је ефикаснија од претходне. Обе библиотеке је могуће наћи на *GitHub* сервису.

Овај модул се може напајати са најмање 3,3 V, или са највише 12 V. Независно од вредности напона напајања користи искључиво 3,3 V логику, тако да је неопходно користити LLS модул, или у нашем случају, отпорнички делитељ направљен за повезивања овог уређаја са микроконтролерима који користе 5 V логику.

4.2.2. Дигитални сат DS3231

На слици 11 је приказан дигитални сат DS3231. Осим пинова за напајање и пинова за I^2C комуникацију постоје још два пина, од којих је само један од интереса у овом раду, а пин 32K није коришћен у овом раду. Пин SQW се користи код *Node*-ова као *interrupt* за буђење микроконтролера. Овај модул се користи као временска референца и као модул који ће укључивати *Node*-ове само у оним тренуцима времена када је потребно да измере и пренесу податке. Модул поседује независно напајање у виду литијум-јонске дугмасте батерије, тако да задржава подешене вредности и након уклањања основног напајања система.

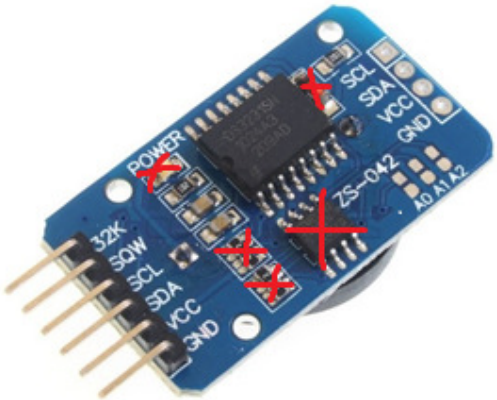


Слика 11. Дигитални сат DS3231 [19].

Напајање се може вршити са 3,3 V или са 5 V. У нашем случају, напајање је помоћу 3,3 V због мање потрошње.

Овај модул на полеђини садржи прикључак за литијум-јонске батерије. Овим се постиже независно напајање у односу на систем, што додатно умањује потрошњу и дозвољава константно бележење времена чак и када сам систем изгуби напајање или дође до било каквог прекида. Важно је напоменути да је овај модул коришћен искључиво због његове способности генерисања аларма, што није доступно на свим комерцијално доступним дигиталним сатовима.

Библиотека коришћена за управљање модулом *DS3232RTC* развијена је од стране корисника *JChristensen* и може се пронаћи на сервису *GitHub*. Библиотека подржава и друге модуле дигиталних сатова и дозвољава коришћене напредних функција, попут аларма, уз минималан број линија рачунарског кода.



Слика 12. Уклоњене компоненте на модулу DS3231 ради уштеде електричне енергије.

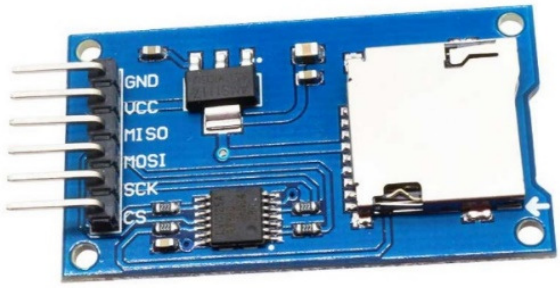
Да би модул могао да се користи у *sleep* режиму што ефикасније, односно како би се додатно смањила потрошња електричне енергије, неопходно је било направити одређене измене уклањањем неколико компоненти, међу којима су EEPROM, LED за приказ прикљученог напајања, диода преко које се пуни батерија на полеђини и два отпорника

отпорности 4,7 kΩ. Уклоњене компоненте немају никакав утицај на функционалност модула, већ само побољшавају енергетску ефикасност, а означене су на слици 12.

4.2.3. Читач SD картица

За снимање података које од разних *Node*-ова прикупља *Master*, користи се читач SD картица који је приказан на слици 13. Овај модул користи SPI комуникацију за повезивање са микроконтролером.

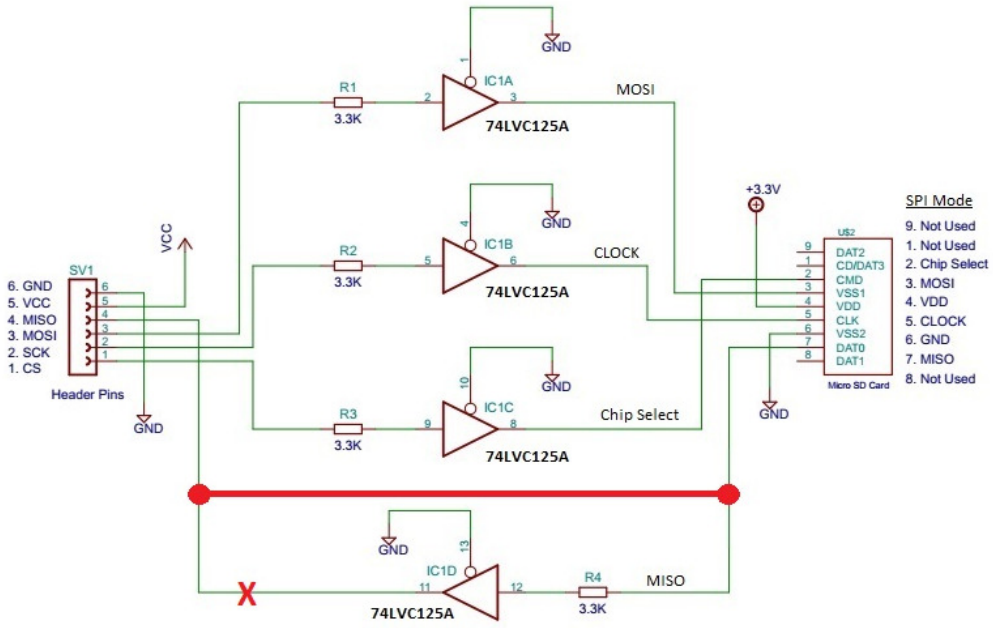
На модулу се налазе два интегрисана кола и место за SD картицу. На слици 13 интегрисано коло које се види у горњој половини плочице је фиксни линеарни напонски регулатор, а интегрисано коло које се види у доњој половини плочице је LLS модул. Ова два кола су неопходна зато што конектор за SD картицу захтева 3,3 V за напајање као и за логичке нивое. Дизајн овог модула је заснован на старом дизајну компаније *SparkFun* који има грешку у повезивању на самом модулу, тако да смо морали да модификујемо модул према шеми датој на слици 14.



Слика 13. Читач SD картица [20].

Када је модул једини уређај који користи SPI комуникацију са микроконтролером, нема проблема у виду комуникације, због чега се привидно чини да модул ради као што је предвиђено, али приликом коришћења још једног уређаја повезаног паралелно настају проблеми немогућности комуникације са било којим уређајем. Као што је могуће видети на слици 14, то се решава тако што се MISO пин директно споји на микроконтролер без коришћења уграђеног LLS модула. С обзиром да је MISO пин дефинисан за слање података са модула на микроконтролер, неопходно је екстерно подићи логички ниво са 3,3 V на 5 V.

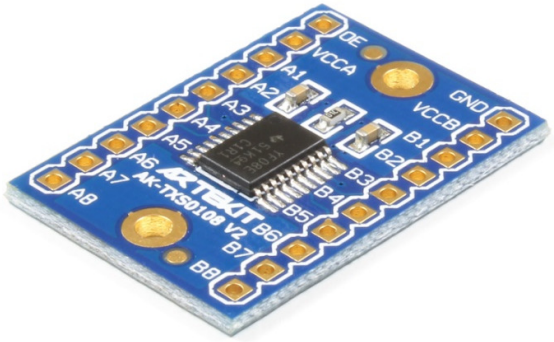
Са слике 14 се такође види због чега долази до проблема повезивања више SPI уређаја паралелно. У стандардној конфигурацији MISO пин константно држи SPI интерфејс „заробљеним“. Последица овога је немогућност других уређаја да врше комуникацију са микроконтролером, а овај проблем превазиђен је изменом назначеном на слици 14.



Слика 14. Измена дизајна читача SD картица који омогућава рад са другим SPI уређајима [21].

4.2.4. LLS модул TXS0108

До сада смо споменули неколико пута LLS модуле и због чега се користе у овом раду. На слици 15 је приказан LLS модул TXS0108 који смо користили. Определили смо се за овај модул због великог броја улазних, односно излазних пинова декларисаних од A1 до A8, односно B1 до B8, као и због тога што може да ради у бидирекционој конфигурацији.

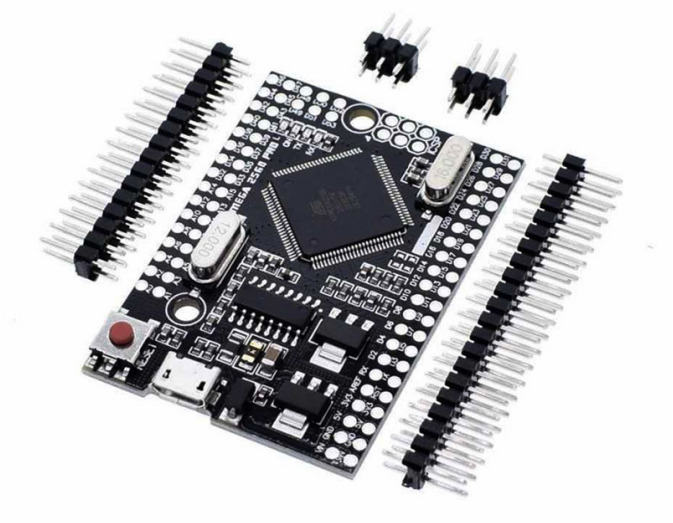


Слика 15. Померач логичких нивоа TXS0108 [22].

Овом модулу је неопходно обезбедити два типа напајања VCCA и VCCB, где је важно да VCCB има већу вредност од VCCA, како би било могуће вршити логичку конверзију. На модулу се, такође, налази пин OE којем се доводи напајање VCCA. У нашем случају на пин VCCB доводимо напајање од 5 V, док на пинове VCCA и OE доводимо напајање од 3,3 V.

4.3. Arduino Mega Pro

Уређај који представља главни део *Master* дела нашег система је *Arduino* MEGA који је приказан на слици 16. Конкретно је коришћена линија *Arduino* MEGA PRO модула због мањих димензија од оригиналног MEGA уређаја. Главна микропроцесорска јединица овог уређаја је микроконтролер ATmega2560 од компаније *Atmel* и представља један од њихових најјачих микроконтролера.



Слика 16. *Arduino* MEGA PRO [23].

Како је било неопходно обезбедити обраду великог броја података, повезивање са меморијски захтевним уређајима и исто развојно окружење као и за *Node*-ове, *Arduino* MEGA PRO је био најпрактичније и најприступачније решење. Иако постоје јачи микроконтролери мањих димензија њихова цена је доста висока, а могућност набавке у Србији је јако ограничена.

У табели 2 су наведене упоредне спецификације *Arduino* MEGA PRO, базираном на микроконтролеру ATmega2560 и *Arduino* UNO, базираном на микроконтролеру ATmega328P. У иницијалном дизајну пошло се од *Arduino* UNO, а касније прешло на *Arduino* MEGA PRO микроконтролер.

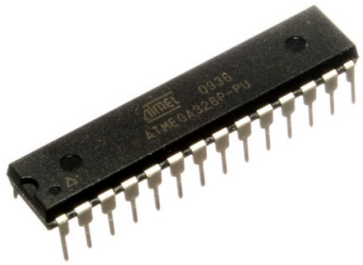
Табела 2. Приказ поређења спецификација *Arduino* MEGA PRO и *Arduino* UNO уређаја.

Микроконтролер	***Arduino* UNO**	***Arduino* MEGA PRO**
Радни напон	5 V	5 V
Број дигиталних пинова	14 (од којих 6 може користити ИКМ)	54 (од којих 16 може користити ИКМ)
Број аналогних пинова	6	16
Број *interrupt* пинова	2	6
***Flash* меморија**	32 KB (од којих је 0,5 KB резервисано за *bootloader*)	256 KB (од којих је 8 KB резервисано за *bootloader*)
SRAM	2 KB	8 KB
EEPROM	1 KB	4 KB
Радни такт	16 MHz	16 MHz

4.4. Микроконтолер ATmega328P

На слици 17 је дат приказ микроконтролера ATmega328P који је коришћен као главна процесорска јединица *Node*-ова система. Микроконтролер ATmega328P је, такође главна микропроцесорска јединица *Arduino* UNO уређаја, чије су спецификације дате у табели 2.

За разлику од *Master* система, где смо користили готов уређај, код *Node* система смо се определили за сам микроконтролер и дизајнирали цео систем око њега [24]. Ово је уређено из два главна разлога; први је простор и распоред истог на штампаној плочи, док је други енергетска ефикасност. Коришћење само микроконтролера нам дозвољава додатан степен слободе код оријентације сензорских модула око микроконтролера као и њиховог повезивања, а омогућава дизајнирање штампане плочице која је димензионо слична комплетном *Arduino* UNO уређају. Са тачке енергетске ефикасности допринос је у томе што се на овај начин искључује допринос регулатора и осталих пасивних компоненти на готовим микроконтролерским уређајима, а омогућено нам је и да сами бирамо те компоненте у циљу што веће енергетске ефикасности. На слици 18 можемо видети шему детаљног повезивања микроконтролера ATmega328P са остатком система, а оваква веза омогућава коришћење овог микроконтролера у *Arduino* развојном окружењу, према конфигурацији пинова са слике 3.

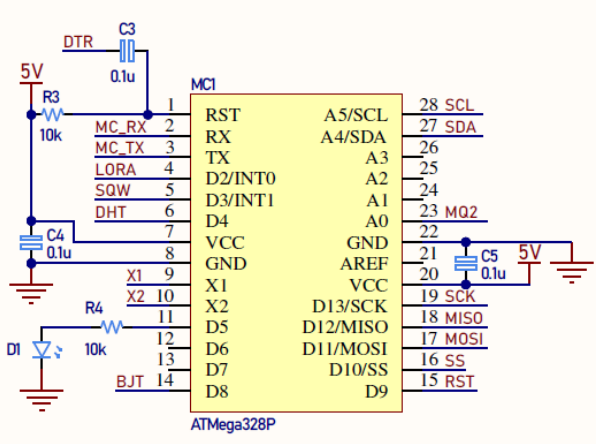


Слика 17. Микроконтролер ATmega328P [8].

Пинови 1, 2 и 3, односно RST, RX и TX, служе за повезивање микроконтролера са програматором, тако да је могуће *upload*-овати софтвер на сам микроконтролер. Приметићемо да је кондензатор C3, који има вредност 0,1 μF паралелно повезан преко отпорника од 10 kΩ на 5 V (слика 18). Ресетовање микроконтролера је неопходно како би се софтвер *upload*-овао, а то се врши тако што се RST пин повеже на масу, односно GND. Да бисмо избегли „лебдећи“ пин и случајан ресет ми везујемо RST на 5 V преко, такозваног, *pull-up* отпорника. Пошто је RST пин изузетно високе импедансе можемо сматрати ту везу као кратак спој. Програматор преко кондензатора C3 и импулса спушта напон на RST пину на привидну масу и тиме се микроконтролер ресетује. Када се микроконтролер ресетује преко пинова 2 и 3, односно RX и TX, врши се комуникација између микроконтролера и програматора.

Пинови 7 и 20 су напојни пинови. На њих се доводи 5 V преко оточно везаног кондензатора капацитивности 0,1 μF. Овај кондензатор служи да сузбије брзе транзијенте у промени напајања, као и да сузбије шум на линији за напајање.

Преко пинова 9 и 10 доводи се референтни такт од 16 MHz који потиче од кристалног осцилатора. Оточно овим пиновима су везани кондензатори од 22 pF, према упутству произвођача.



Слика 18. Шема повезивања микроконтролера ATmega328P у остатак система.

Ако бисмо се определили да овај систем користимо са напајањем од 3,3 V, било би неопходно користити уграђени осцилатор од 8 MHz, што није препоручено од стране произвођача јер је потребно оспособљавање *brown-out* детектора на самом чипу, што у неким случајевима није могуће.

Пин 21 је референтни ниво за аналогне сигнале и има вредност од 1,1 V са грешком ±10%. На овај пин није ништа повезано, иако је могуће повезати екстерну референцу, али је неопходан за даљи рад када будемо приказивали прорачун нивоа напајања на улазу микроконтролера.

4.5. Шеме комплетног система

У овом потпоглављу ћемо анализирати шеме *Node* и *Master* система. Објаснићемо како су изабране компоненте и како су повезани горе наведени модули.

Све шематске компоненте су цртане од почетка и подешене заједно са одговарајућим *footprint*-овима. Вредности и димензије компоненти су прикупљене из њихових *datasheet*-ова.

4.5.1. Шема Node система

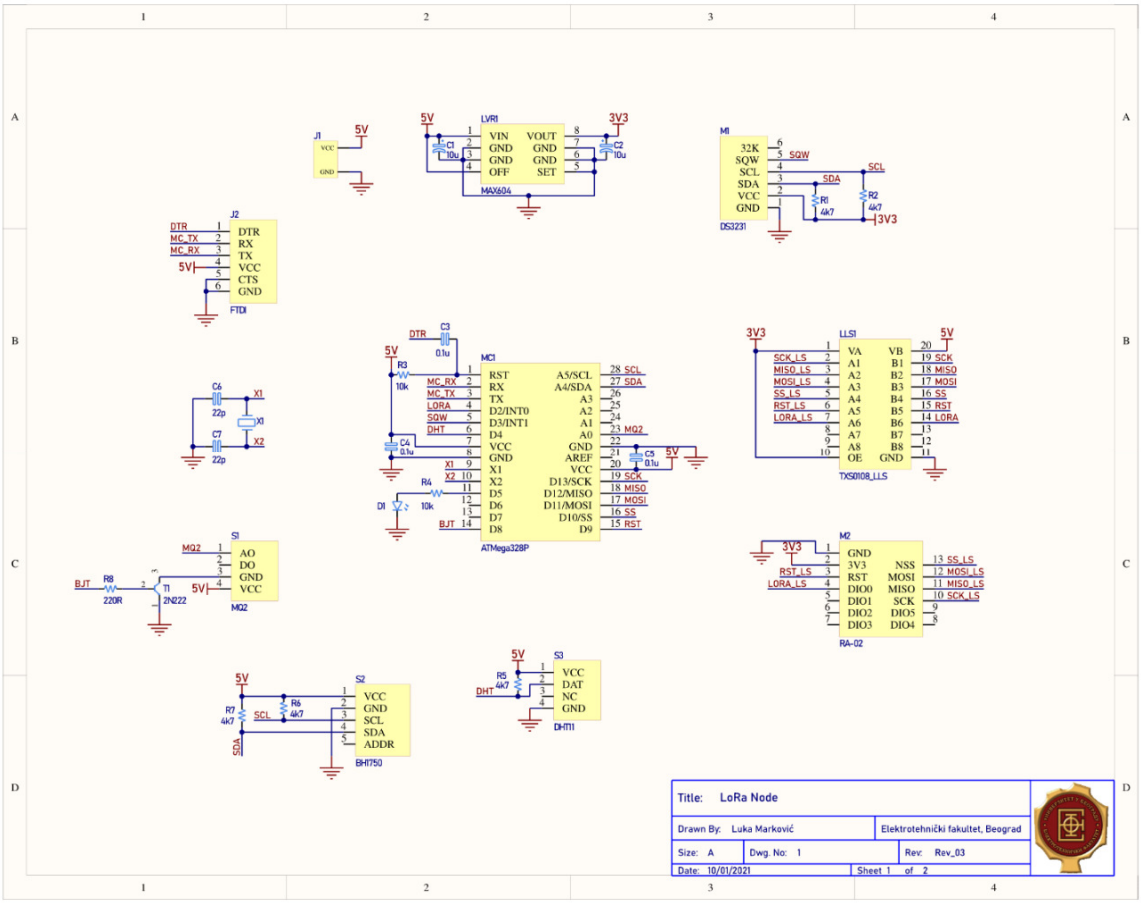
Анализу шеме *Node* система, која је приказана на слици 19, ћемо започети од напајања. Главно напајање система се доводи преко компоненте J1 која представља два 2,54 mm мушка пина преко којих се доводи екстерно напајање од 5 V у целокупни систем. Компоненте S1, S2 и S3 се напајају директно са 5 V, као и сам микроконтролер ATmega328P. Сваки модул на својој штампаној плочи садржи кондензаторе капацитивности 0,1 µF, тако да то нисмо морали засебно да додајемо. Напајање од 3,3 V смо добили преко фиксног линеарног напонског регулатора MAX604. Овај регулатор се користи због велике јачине струје коју може доставити, реда величине 250 mA, као и због изузетно мале потрошње када је систем у *sleep* режиму рада, од око 10 µA до 14 µA. Ово је енергетски врло ефикасно, посебно када се

узме у обзир да стандардни регулатори имају потрошњу од око 15 mA. Ова величина се наводи као *quiescent current*.

Регулатор је конфигурисан тако да на улазу и на излазу има оточно везане тантал кондензаторе капацитивности 10 µF што омогућава равну карактеристику напона на улазу, односно излазу. Иако систем функционише овакав какав јесте, генерално је лоша пракса не ставити и оточно кондензаторе капацитивности 0,1 µF за сузбијање брзих транзијената, као и оточно везивање отпорника, отпорности 4,7 kΩ такозваног *bleeder* отпорника, који служи за пражњење кондензатора када се систему укине напајање.

Дигитални сат DS3231 и сензор нивоа осветљења BH1750 повезани су I^2C пиновима микроконтролера, односно пиновима 27 и 28 микроконтролера ATmega328P. По препоруци произвођача на ове пинове, са стране модула, а не са стране микроконтролера, везани су оточно *pull-up* отпорници на SDA и SCL пинове. Један овакав отпорник је такође повезан код сензора температуре и влажности ваздуха DHT11 иако он користи други вид комуникације.

Како је *LoRa* једини модул који користи SPI протокол комуникације, SPI пинови микроконтролера, односно пинови 17, 18 и 19 се преко LLS-а повезују са *LoRa* модулом.

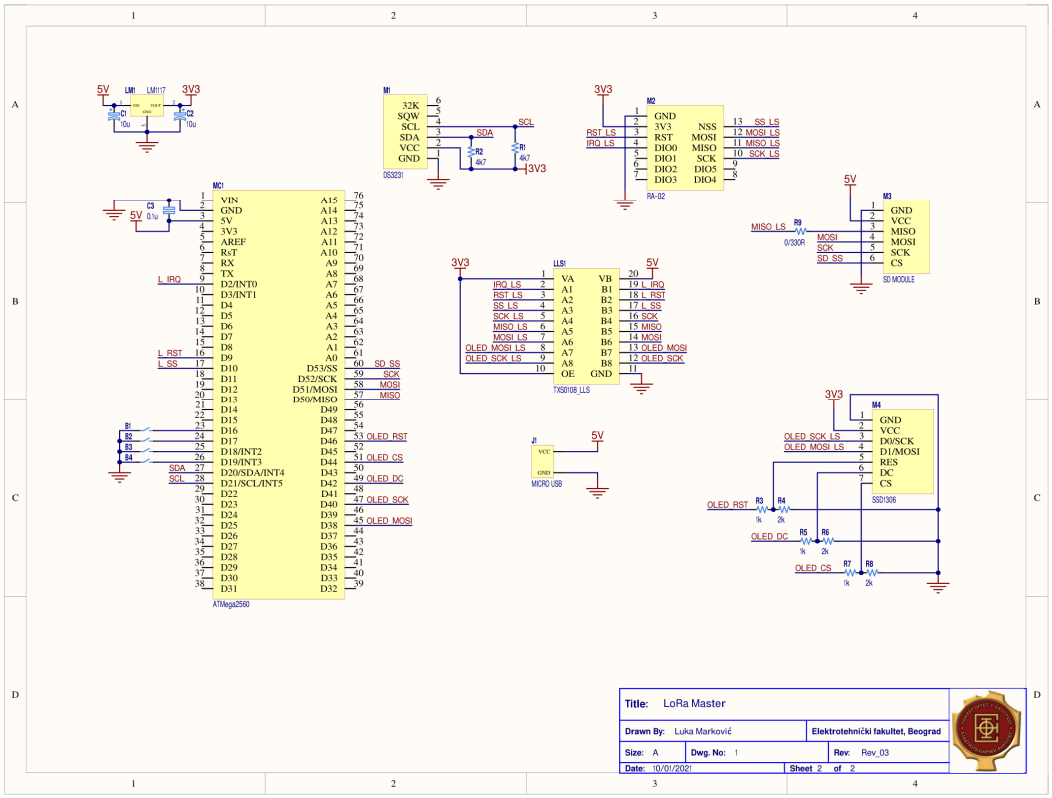


Слика 19. Шема *Node* система.

Пинови 2 и 3 на микроконтролеру су *interrupt* пинови. Они су редом везани за *LoRa* IRQ пин и DS3231 SQW пин. Они омогућавају буђење система из *sleep* режима рада по потреби. Пин 4 је повезан на DATA пин DHT11, али је могуће користити било који други дигитални пин. Пин број 11, дигитални пин број 5 се користи за контролисање LED и служи искључиво као *feedback* провере рада система. Пин број 14, односно дигитални пин број 8, користи се за укључивање и искључивање MQ2 сензора, док се пин 23 користи за читање података са истог, као што је претходно наведено. За крај, пинови 15 и 16 се користе за контролисање LoRa модула.

4.5.2. Шема Master система

На слици 20 је приказана комплетна шема *Master* система. Анализу ћемо поново започети од напајања, преко *micro-USB* конектора се доводи 5 V у систем, што се даље води на одговарајуће модуле, док се преко LMP1117-3.3 фиксног линеарног напонског регулатора доводи 3,3 V. За разлику од *Node* система, потрошња *Master* система није од интереса јер се напајање доводи из рачунара или, још боље, преко пуњача за мобилни телефон/таблет, а не из батерије.



Слика 20. Шема *Master* система.

Коришћењем LLS-а вршена је комуникација између *Arduino* MEGA PRO и *LoRa* модула, као и комуникација са SD и SSD1306 OLED модулима. Како смо већ навели да SD модул има уграђен регулатор и LLS неопходно је само повезати његов MOSI пин на LLS преко отпорника отпорности 330 Ω. Ово је урађено ради симетрије са осталим пиновима који су,

такође, на самом модулу повезани преко отпорника отпорности 330 Ω, што решава потенцијалне проблеме кашњења. Као што можемо видети на шеми са слике 20, пинови 57, 58 и 59, односно дигитални пинови 50, 51 и 52 представљају редом SPI пинове. Ово су хардверски SPI пинови које користимо за комуникацију са *LoRa* и SD модулима, док за комуникацију са SSD1306 OLED модулом користимо дефинисане софтверске SPI пинове које је могуће подесити на било који дигитални пин микроконтролера. Разлог овој измени, за разлику од везивања сва три уређаја у паралелу, је тај што је раније долазило до грешке при комуникацији, односно што је била могућа грешка у конфликту међу библиотекама које се користе. Да не бисмо користили још један LLS и уштедели на месту установили смо да су DC, RST и CS пинови OLED модула *input* пинови, као и да се преко њих не шаљу комуникациони сигнали, тако да је могуће користити стандардни отпорнички делитељ са отпорницима отпорности 1 kΩ и 2 kΩ.

Како не постоји потреба за аларм системом помоћу дигиталног сата DS3231, пин SQW није неопходно повезати, већ нам само требају пинови за међусобну комуникацију читања и уписа времена.

На пинове 23, 24, 25 и 26, односно дигиталне пинове 16, 17, 18 и 19, су повезани тастери за контролисање мени система. Приметимо да су пинови 18 и 19 заправо *interrupt* пинови, што нам омогућава да приступимо мени систему када код желимо, без обзира на функцију која се тренутно извршава.

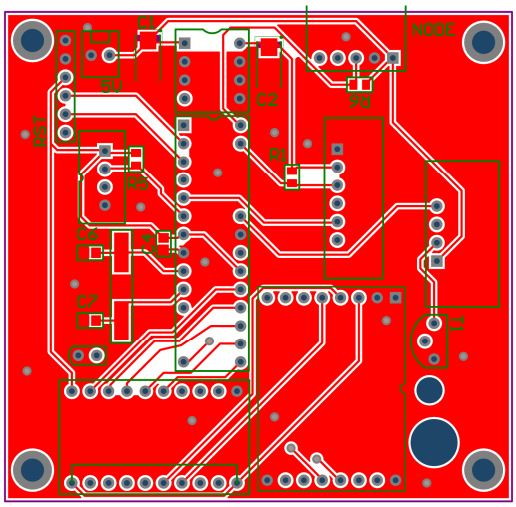
4.6. PCB дизајн

У овом потпоглављу ћемо врло кратко описати процес дизајна штампаних плоча оба система, *layout* и оријентацију модула. Осим кондензатора и отпорника који су SMD величине 0805, све компоненте су такозване *through hole* што значајно олакшава лемљење у кућној варијанти.

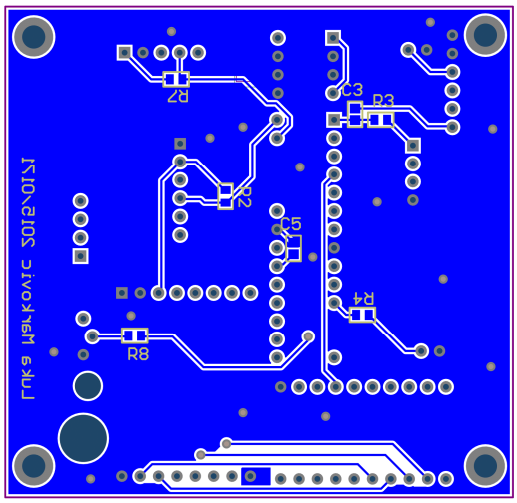
4.6.1. PCB дизајн Node и Master система

На сликама 21 и 22 је су приказани, редом, горњи и доњи слој штампане плоче Node система, док су на сликама 23 и 24 приказани горњи и доњи слој штампане плоче Master система. Штампане плоче су дизајниране у двослојној технологији, где се рутирање вршило у оба слоја. За овако модуларан дизајн двослојна штампа је и више него адекватна, али ако би се овакав систем пројектовао без модула онда би вероватно било потрбно користити четворослојну штампу где би се рутирање сигнала вршило на горњем и, евентуално, доњем слоју, док би међуслојеви били за GND и напајање.

Добра пракса је да се сви трагови којима се преносе сигнали рутирају контролисаном импеданом, генерално 50 Ω у границама од ± 5%, а трагови којима се спроводи напајање буду од 0,762 mm (30 mils) до 1,27 mm (50 mils). Како смо ми пројектовали штампу на витропласту (FR-4) који има релативну ефективну пермитивност $\varepsilon_r = 4{,}4$ и дебљину 1,6 mm, није било могуће вршити рутирање траговима контролисане импедансе јер су ширине 50 омских водова, на витропласту, приближно у опсегу $[1{,}8h, 2h]$, где је h дебљина супстрата, а ово би било прешироко за наш дизајн.

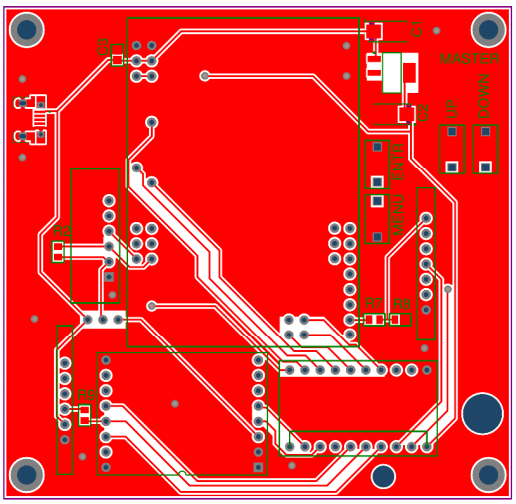


Слика 21. Горња страна плочица *Node* система.

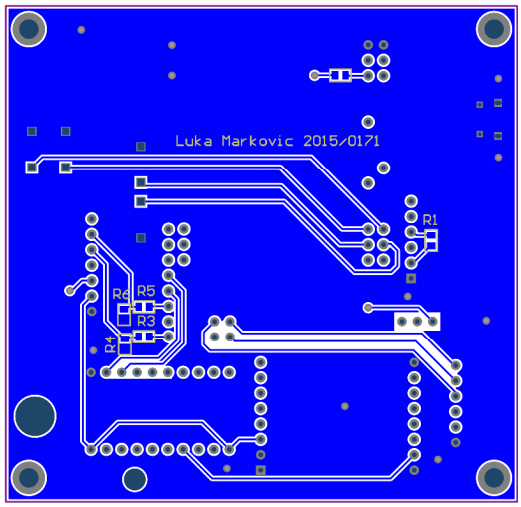


Слика 22. Доња страна плочице *Node* система.

Како се овде не преносе брзи сигнала, нити користи велика количина струје, определили смо се да сви трагови буду исте ширине 0,254 mm. Ово нам је омогућило поуздану везу између елемената кола, као и мање димензије штампаних плоча. Такође, са ове четири слике може се приметити да су кондензатори капацитивности 0,1 μF постављени што је ближе могуће пиновима за напајање, из разлога образложених у претходном поглављу.



Слика 23. Горњи страна плочице *Master* система.

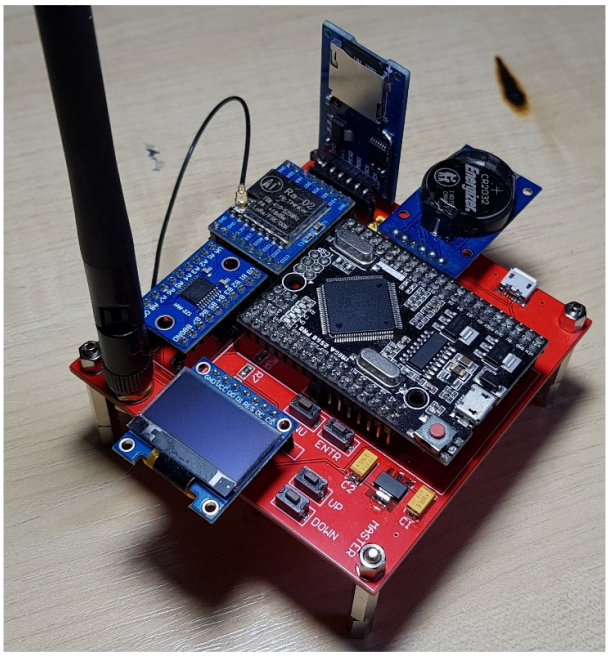


Слика 24. Доња страна плочице *Master* система.

4.6.2. Приказ израђених штампаних плоча

На сликама 25 и 26 су приказане произведене штампане плоче *Master* и *Node* система, респективно, док су на сликама 27 и 28 приказани 3D модели истих система у програмском пакету *Altium Designer*. За разлику од 3D приказа, ми нисмо директно лемили модуле на плочу, већ смо користили женске *header* конекторе. Ово нам дозвољава лако тестирање и

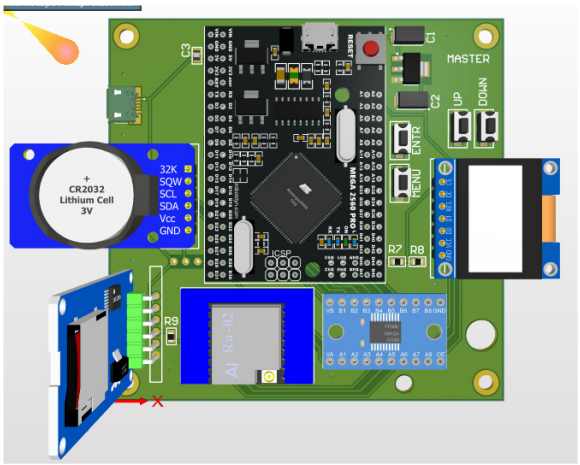
замену неисправних модула, као и довољно добар механички контакт уз задржавање високе модуларности читавог система.



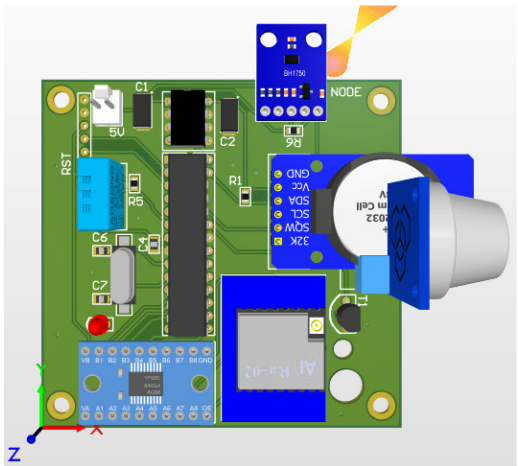
Слика 25. Штампана плоча *Master* система.

Слика 26. Штампана плоча *Node* система.

Како је 3D модел дигиталног сата у фиксном положају и није га било лако едитовати односно изменити, на слици 28 долази до преклапања са модулом детекције штетних гасова, али то ни на који начин није утицало на финалну верзију штампе. Једини проблем са којим смо се сусрели је што смо рупу за антену поставили превише близу *LoRa* модула, не узимајући у обзир димензију антене. Ово смо накнадно кориговали, тако што смо направили још једну рупу у, гледајући слику 28, горњем десном углу, што није утицало на перформансе система, већ само естетику.



Слика 27. 3D приказ *Master* штампане плоче у програмском пакету *Altium Designer*.



Слика 28. 3D приказ *Node* штампане плоче у програмском пакету *Altium Designer*.

5. Резултати

5.1. Опис рада целокупног система

Пре него што будемо приказали резултате, неопходно је да наведемо принцип рада целокупног система, тачније, да размотримо принципе рада *Master* система и *Node* система, односно услове под којима је комплетан систем тестиран.

Master систем има главну улогу у бележењу и чувању података свих *Node* система. OLED дисплеј који је саставни део *Master* система, као и дугмићи за навигацију служе за лак и брз приступ најновијим подацима *Node* система, као и подешавање времена дигиталног сата, које служи за тачно бележење података. Подаци се бележе у формату приказаном на слици 29, чиме је могућа њихова даља обрада на рачунару.

Node систем може бити сингуларни *Node* или мрежа више *Node*-ова који се разликују по њиховом јединственом кључу, а који помаже при аутентификацији приликом слања података *Master* систему. *Node* систем је већину времена у *sleep* режиму ради очувања нивоа батерије, а буди се из овог режима рада помоћу дигиталног сата ради мерења података и слања истих помоћу *LoRa* модула *Master* систему. Када је *Node* систем у *sleep* режиму, дигитални сат води евиденцију о томе да ли је неопходно прекинути овај режим у зависности од изабраног аларм система. Основно подешавање је да се врши буђење сваког сата, али ако је ниво гаса превише висок учесталост буђења се смањује на сваки минут. Том приликом се прикупљају мерења са свих сензора, пакују у одговарајућу структуру и врши се слање целокупне структуре. Након тога, систем поново улази у *sleep* режим и чека поновно буђење.

012021.txt - Notepad

File Edit Format View Help

```
4:22:34 9/1/2021 179 Node 1 17.20 50.00 0.00 0.00 4910 10.25 -72.00
4:23:34 9/1/2021 179 Node 1 17.50 50.00 0.00 0.00 4954 9.75 -72.00
4:24:34 9/1/2021 179 Node 1 17.50 50.00 0.00 0.00 4954 9.50 -72.00
4:25:34 9/1/2021 179 Node 1 17.20 50.00 0.83 0.00 4910 10.25 -72.00
4:26:34 9/1/2021 179 Node 1 17.20 50.00 0.00 0.00 5115 9.50 -72.00
4:27:34 9/1/2021 179 Node 1 17.40 50.00 0.00 0.00 4932 10.00 -72.00
4:28:34 9/1/2021 179 Node 1 17.40 50.00 0.83 0.00 4910 9.75 -71.00
4:29:34 9/1/2021 179 Node 1 17.40 50.00 0.83 0.00 4932 9.50 -71.00
4:30:34 9/1/2021 179 Node 1 17.40 50.00 0.00 0.00 4954 9.50 -71.00
4:31:34 9/1/2021 179 Node 1 17.30 50.00 0.00 0.00 4954 10.00 -71.00
4:32:34 9/1/2021 179 Node 1 17.30 50.00 0.00 0.00 5091 10.00 -71.00
4:33:34 9/1/2021 179 Node 1 17.40 50.00 0.83 0.00 5115 9.75 -71.00
```

Слика 29. Формат уписивања података на SD картицу.

5.2. Мерење потрошње Node система

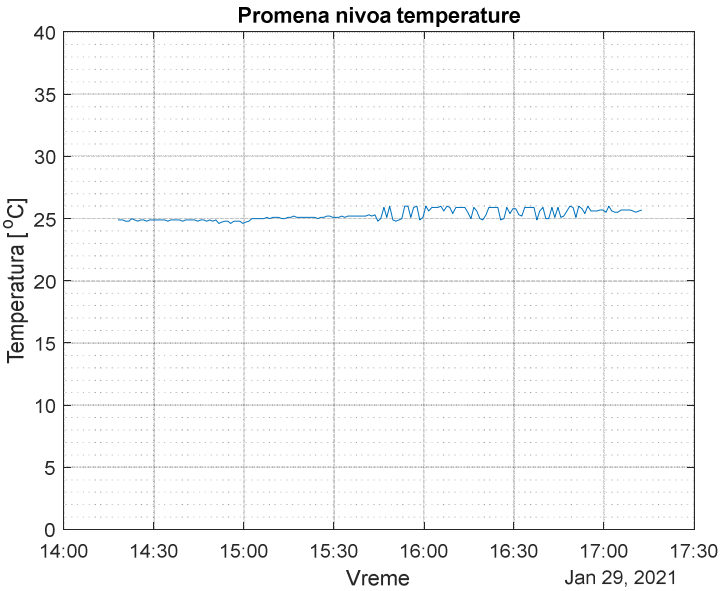
На слици 30 се може видети потрошња струје *Node* система када се систем налази у *sleep* режиму рада. Као што можемо видети, када је систем у *sleep* режиму потрошња је свега 0,53 mA, односно 530 μA. Како је приликом слања податак за *LoRa* модул потребна јачина струје 120 mA, а за мерење нивоа гаса јачине 180 mA, потрошња у тренутку мерења и слања података превазилази 300 mA све укупно. То јесте релативно висока вредност струје коју захтевају коришћени сензори, али пошто се слање обавља врло брзо, усредњено се може сматрати да систем у просеку троши око 600 μA ако се слање врши сваког сата.

Слика 30. Потрошна струја *Node* система у *sleep* режиму изражена у µA.

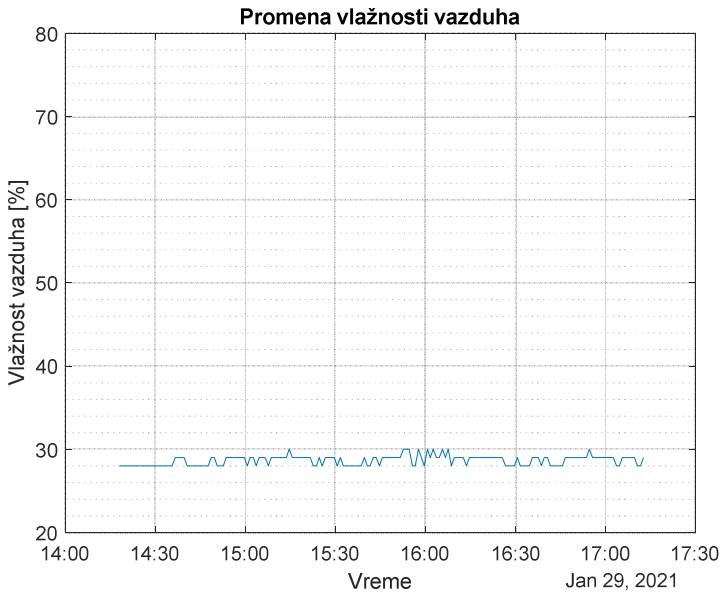
5.3. Резултати мерења

На следећим сликама ће бити приказани резултати добијени мерењем сваког минута у периоду од три сата, изузев резултата о присутности штетних гасова који је намерно изостављен због константног очитавања нулте вредности, што је био жељени резултат. Резултати су уписивани у текст документ на SD картици (слика 29), где колоне представљају, редом: сат, минут и секунду; дан, месец и годину; кључ одређеног *Node*-а, његов назив, забележену температуру, влажност ваздуха, ниво осветљења, присуство штетних гасова, ниво батерије изражен у mV, SNR и RSSI. Подаци су потом обрађивани помоћу програмског пакета MATLAB. Сва мерења су вршена у просторији од 10 m^2 која има добру топлотну изолацију, али је слабо изложена светлости.

На слици 31 је приказана варијација температуре просторије у периоду од три сата. Као што можемо видети, мерења су поприлично стабилна и у очекиваним границама, док мерења влажности ваздуха, која су приказана на слици 32, имају благе варијације, али откривају нездраво ниску вредност влажности ваздуха, која би идеално требала да се креће у опсегу од 40 % до 60 %.

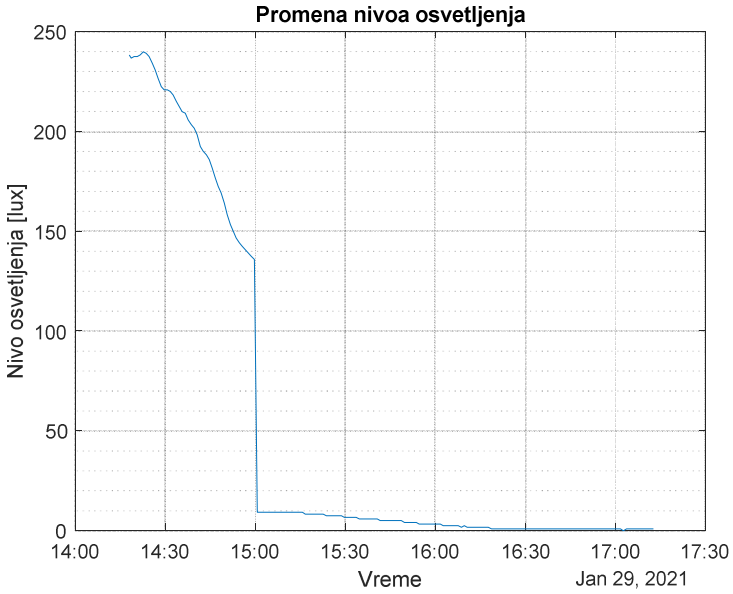


Слика 31. Температура измерена у периоду од 3 сата.



Слика 32. Влажност ваздуха измерена у периоду од 3 сата.

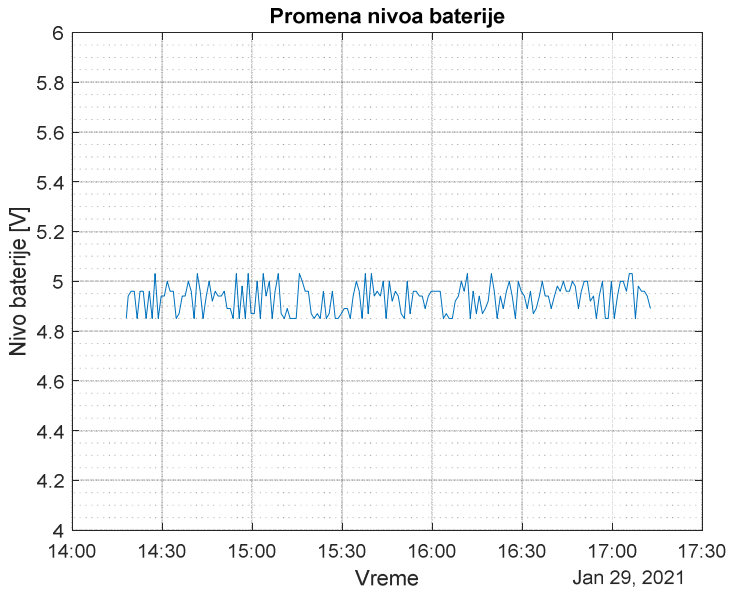
На слици 33 је приказана промена нивоа осветљења у периоду од три сата. Јасно се може видети нагли пад у 15:00 када је завеса пребачена преко прозора поред којег је *Node* систем стајао.



Слика 33. Промена нивоа осветљења у периоду од 3 сата.

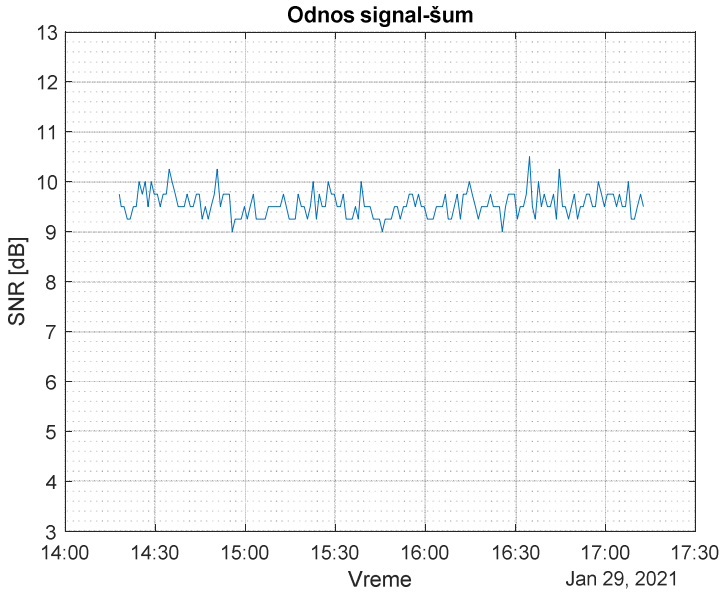
На слици 34 се виде резултати мерења напона батерије који значајно варирају око 5 V. Претпостављамо да је ово последица коришћеног модула за Li-ion батерије, који има у себи уграђен прекидачки DC-DC конвертер, а који на свом извору не даје идеално стабилан напон. Мерењем овог напона волтметром очитавали смо само две вредности, и то 5,07 V и 5,1 V, а када смо пребацили цео систем на 3,3 V напајање, очитавали смо константних 3,4 V.

Наравно, при потрошњи електричне енергије о којој смо говорили на почетку поглавља, време тестирања је недовољно да би се са ових графика видео ефекат пражњења батерија.

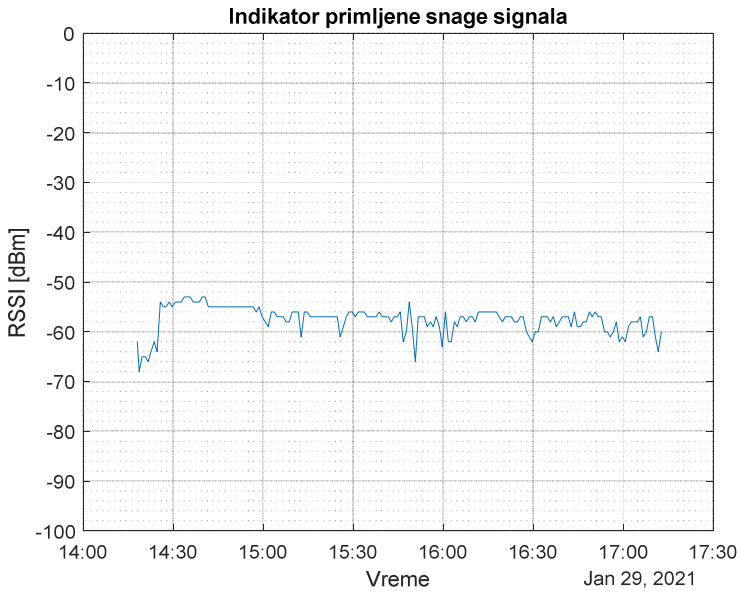


Слика 34. Промене нивоа батерије у периоду од 3 сата.

Процењени однос сигнал-шум, SNR је приказан на слици 35. Јасно се види да се вредности крећу од 9 dB до 10 dB, док су вредности RSSI приказани на слици 36 и крећу се у опсегу од −70 dBm до −58 dBm. У нашим ранијим тестирањима, која нису приказана овде, успели смо да добијемо веродостојан пренос када је SNR процењен на само 3,5 dB, што говори доста о робусности *LoRa* модулационе технологије.

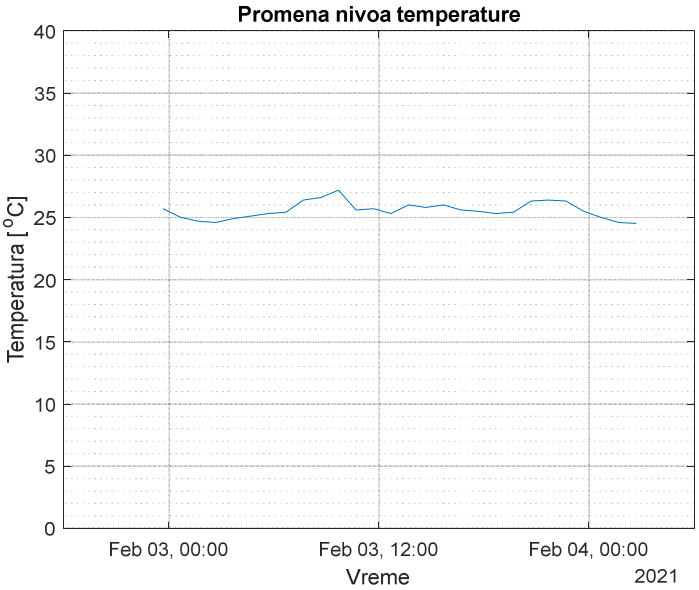


Слика 35. Процењена вредност SNR за сваки пакет.

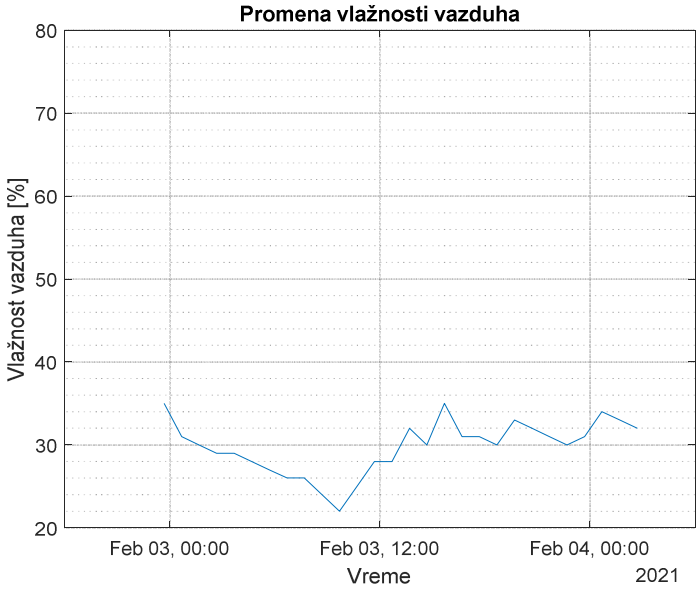


Слика 36. Измерена вредност RSSI за сваки пакет.

На следећим сликама ће бити приказани резултати мерени у истој просторији али са мерењем сваког сата у периоду од 26 сата. Приликом овог теста није било никаквих назнака прекида рада система, а овим тестом извршена је потврда да систем може стабилно да ради и у дужем временском периоду. Као и у претходним резултатима, мерење нивоа штетних гасова је приказивало вредност 0, тако да ћемо та мерења изоставити.

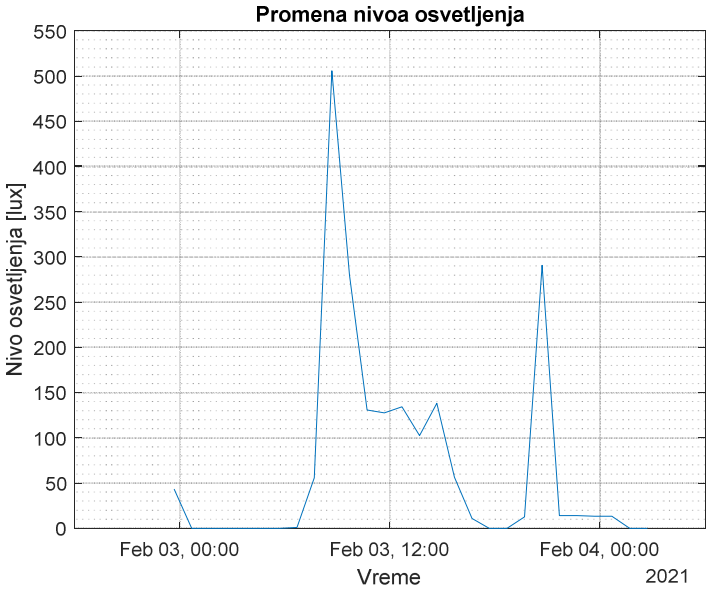


Слика 37. Температура измерена у периоду од 26 сати.

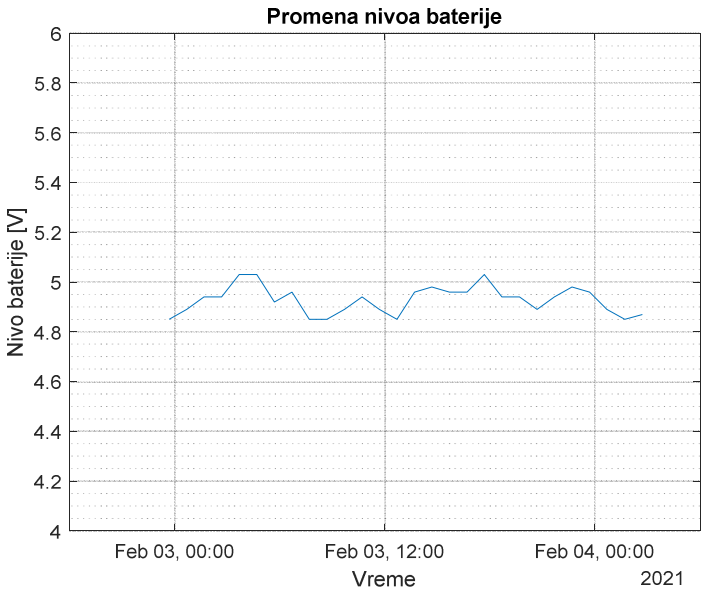


Слика 38. Влажност ваздуха измерена у периоду од 26 сати.

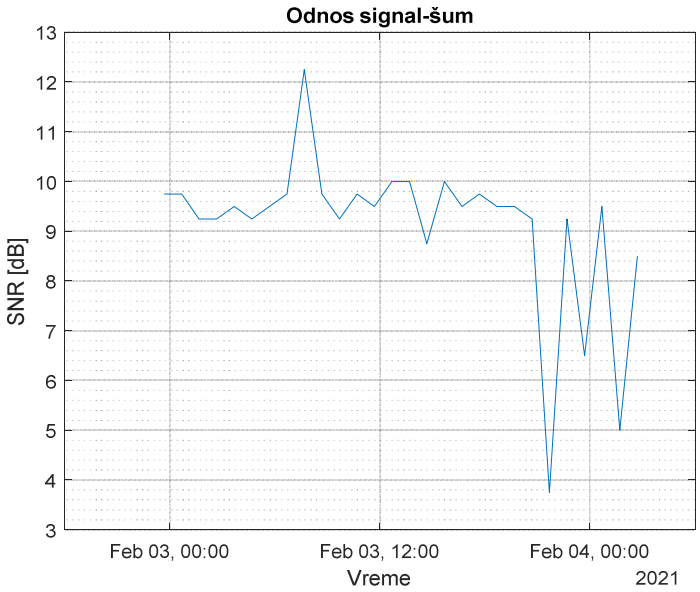
Иако је осветљење просторије у којој се налази *Node* систем релативно лоше, са слике 39 можемо приметити јасно када долази до пораста нивоа светлости, а по интензитету можемо закључити да се ради о дневној светлости. Касније, након заласка сунца, са слике 39 можемо препознати тренутке времена када се укључује вештачко осветљење у непосредној близини *Node* система.



Слика 39. Ниво осветљења измерен у периоду од 26 сати.

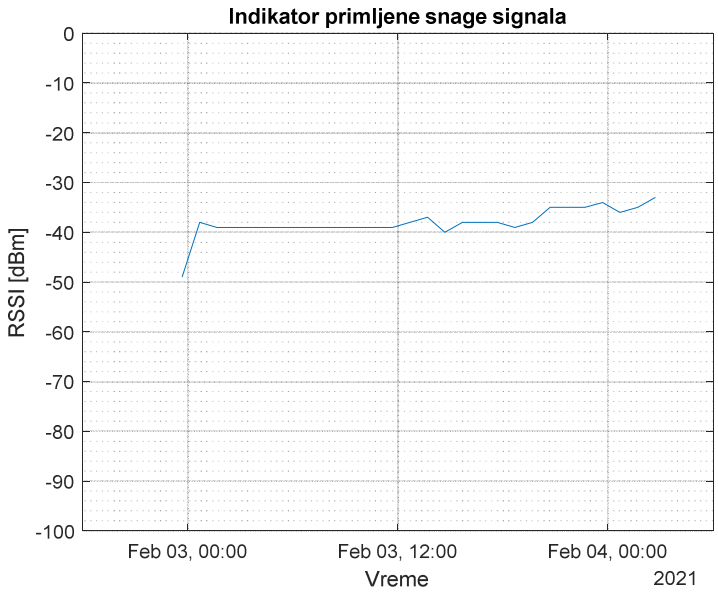


Слика 40. Промена нивоа батерије у периоду од 26 сати.



Слика 41. Процењена вредност SNR за сваки пакет.

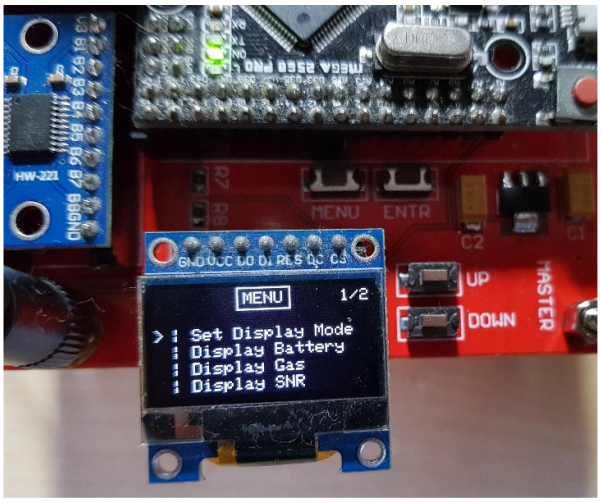
Како се права вредност SNR не мери, већ се само процењује, не можемо изнети неки значајни закључак са слике 41.



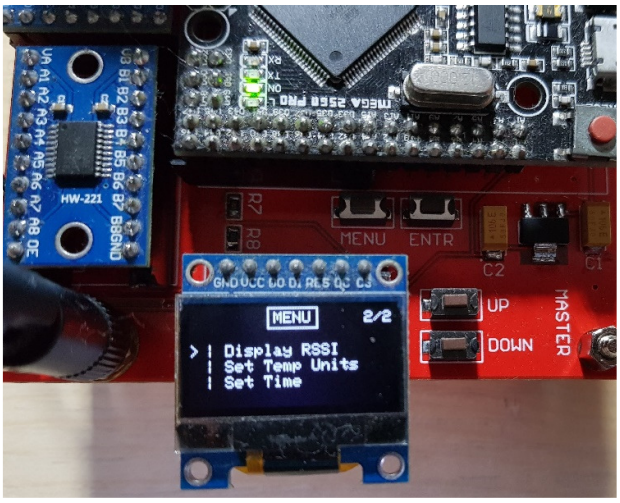
Слика 42. Измерена вредност RSSI за сваки пакет.

5.4. Приказ мени система

На сликама 43 и 44 је приказан мени систем који је осмишљен за *Master* систем. Као што можемо видети мени систем је подељен на две стране. На првој страни мени система могућ је приказ података одређених *Node* уређаја, али је такође могуће подесити циклично приказивање података више уређаја кроз подмени. Затим је могуће погледати ниво батерије, ниво гаса и процењену вредност SNR за сваки уређај. На другој страни мени система је могуће погледати ниво RSSI за сваки уређај, подесити у којим јединицама се приказује температура и може се подесити време *Master* система. Оваквим системом мерења уређај је у потпуности независан и једноставан за коришћење, без потребе за управљањем раније поменутим рачунарским библиотекама.



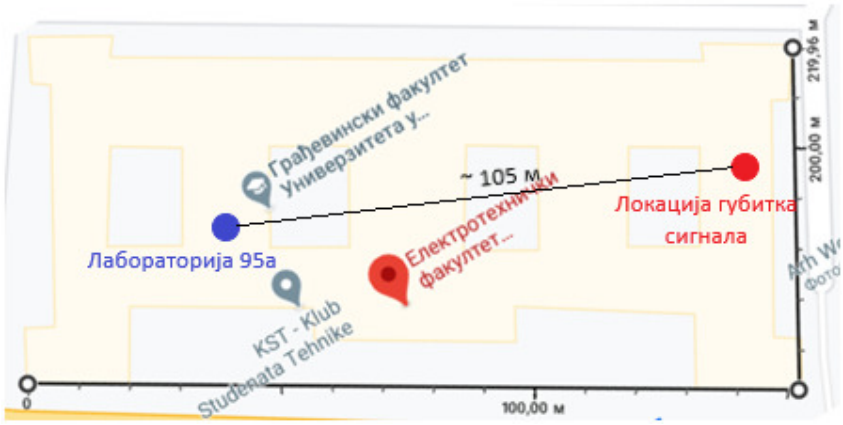
Слика 43. Прва страна мени система.



Слика 44. Друга страна мени система.

5.5. Процена максималног домета

Као што смо нагласили на почетку рада, једна од кључних особина *LoRa* система је њен велики домет. Како нисмо могли да меримо домет директно, за шта нам је био потребан врло велики простор, пожељно у руралној средини, за процену максималног домета искористили смо чињеницу да сигнал доста слаби при проласку кроз дебеле зидове, тако да смо мерење вршили у згради Електротехничког факултета. Сигнал смо слали из лабораторије 95а, у приземљу, док смо пријемник шетали кроз целу зграду Факултета и напокон изгубили конекцију на супротној страни зграде на последњем, односно трећем спрату. На слици 45 приказане су димензије зграде Електротехничког факултета, где су јасно назначена места на којима се налазе лабораторија 95а и локација губитка сигнала, респективно. Као што можемо видети, комуникација је остварена кроз шест спољашњих фасадних зидова зграде, неколицину унутрашњих преградних зидова и три међуспратне конструкције.



Слика 45. Димензије зграде Електротехничког факултета [25].

6. Закључак

У овом раду осмишљен је, дизајниран и фабрикован комплетан *LoRa-Arduino* бежични систем, укључујући и хардверски и софтверски део, а један од циљева био је и да систем буде што повољнији и енергетски ефикаснији. Овим системом се бежичним путем на удаљеним локацијама мери температура, влажност ваздуха, ниво светлости и присутност штетних гасова. Систем је замишљен као модуларан, тако да се релативно једноставно може надоградити за мерење и неких других физичких величина.

Након теоријског увода, где је, између осталог разматран и *LoRa* протокол комуникације, размотрене су тренутне и потенцијалне будуће примене комуникације помоћу *LoRa* протокола. Размотрени су и разни хардверски аспекти овог система, као што су комуникација између *LoRa* модула и микроконтролера ATmega328P, као и начин реализације минималистичког дизајна помоћу микроконтролера у циљу минимизирања потрошње електричне енергије тако да се перформансе комплетног система не деградирају. Описани су и сви коришћени сензори, као и недостаци које су оригинални сензори имали и начини на које смо те недостатке исправили. На крају је помоћу CAD алата направљен тродимензионални изглед штампаних плочица које су израђене. На овај начин, кроз CAD дизајн оптимизоване су димензије штампаних плочица, као и међусобни положај коришћених сензора. Израђени су и прототипови овог система, а помоћу тих прототипова извршена су и мерења физичких величина од интереса.

Резултате мерења смо приказали двојако: у јако кратком временском периоду, али са честим мерењима; као и резултате мерења у доста дужем временском периоду уз ређе мерење, као провера поузданости система и могућности исправног рада у оба случаја. Иако нисмо тестирали домет система у идеалним условима, показали смо да је могућа комуникација на великим раздаљинама (~1 km) без било какве додатне опреме. Направљени систем може се сматрати практично комплетним производом, а након убацивања уређаја у робусно кућиште које би било отпорно на влагу, систем би могао успешно да се користи и у спољашњим временским условима.

Литература

[1] Semtech, „What is LoRa?,“ [На мрежи]. Available: https://www.semtech.com/lora/what-is-lora. [Последњи приступ 21 јануар 2021].

[2] S. Chin, „LoRA Technology Comes to Aid of Australian Cotton Farmers,“ 5 децембар 2018. [На мрежи]. Available: https://www.eeworldonline.com/lora-technology-comes-to-aid-of-australian-cotton-farmers/.

[3] LoRa Alliance, „Why LoRaWAN is the Logical Choice for Crop Production Monitoring Solutions,“ LoRa Alliance, 2020.

[4] LoRa Alliance, „Preserving Tree Health With a Simple LoRaWAN Sensor,“ LoRa Alliance, 2020.

[5] K. Mishra, „Automated Illegal Parking Management LoRaWAN Whitepaper,“ p. 14, 29 новембар 2018.

[6] Makerfabs, „SX1278 LoRa Module 433M 10KM Ra-02,“ [На мрежи]. Available: https://www.makerfabs.com/sx1278-lora-module-433m-10km-ra-02.html.

[7] Arduino, „Introduction,“ Arduino LLC, [На мрежи]. Available: https://www.arduino.cc/en/guide/introduction.

[8] Wikipedia, „ATmega328P,“ 22 јануар 2021. [На мрежи]. Available: https://en.wikipedia.org/wiki/ATmega328.

[9] Microcontrollers Lab, „ATMEGA328P Microcontroller,“ [На мрежи]. Available: https://microcontrollerslab.com/atmega328p-microcontroller-pinout-prograamming-features-datasheet/.

[10] Smart Prototyping, „LoRa Ra-02 433MHz Long Range Wireless Transreceiver - SX1278,“ [На мрежи]. Available: https://www.smart-prototyping.com/LoRa-SX1278-10KM-433M-long-range-wireless-module-Ra-02.

[11] Corelis, „SPI Tutorial,“ Corelis, [На мрежи]. Available: https://www.corelis.com/education/tutorials/spi-tutorial/#Summary.

[12] National Instruments, „What Is Serial Synchronous Interface (SSI)?,“ National Instruments, 8 јануар 2019. [На мрежи]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019MgLSAU&l=en-RS.

[13] en:User:Cburnett, „File:SPI single slave.svg,“ 19 децембар 2006. [На мрежи]. Available: https://commons.wikimedia.org/wiki/File:SPI_single_slave.svg.

[14] NXP, „I2C-bus specification and user manual,“ 4 април 2014. [На мрежи]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[15] „Digital DHT11 DHT-11 Sensitivity Temperature and Humidity Sensor For Arduino,“ [На мрежи]. Available: https://www.ebay.com/itm/Digital-DHT11-DHT-11-Sensitivity-Temperature-and-Humidity-Sensor-For-Arduino-/253376104535.

[16] „BH1750 Light Intensity Sensor,“ [На мрежи]. Available: https://www.addicore.com/BH1750FVI-p/ad290.htm.

[17] „TZT MQ-2 MQ2 Smoke Gas LPG Butane Hydrogen Gas Sensor Detector Module For Arduino,“ [На мрежи]. Available: https://www.aliexpress.com/item/32727143285.html.

[18] „SSD1306 128x64 Pixel OLED Display Module (White),“ [На мрежи]. Available: https://hobbycomponents.com/displays/400-ssd1306-128x64-pixel-oled-display-module-white.

[19] „DS3231 RTC Module Real Time Clock Module AT24C32 High Accuracy,“ [На мрежи].

Available: https://www.digitspace.com/ds3231-rtc-module-real-time-clock-module-at24c32-high-accuracy.

[20] „Micro SD Card Module,“ [На мрежи]. Available: https://www.makerlab-electronics.com/product/micro-sd-card-module/.

[21] ShermanP, „SPI SD card reader & other SPI device not working together,“ 19 март 2020. [На мрежи]. Available: https://forum.arduino.cc/index.php?topic=465101.0.

[22] „AK-TXS0108 V2 - Bidirectional Level Converter Breakout,“ [На мрежи]. Available: https://www.artekit.eu/products/breakout-boards/io/ak-txs0108-v2-bidirectional-level-converter-breakout/.

[23] „Mega 2560 PRO MINI ATmega2560-16AU CH340G,“ [На мрежи]. Available: https://www.makerlab-electronics.com/product/arduino-mega-2560-pro-mini-atmega2560-16au-ch340g/.

[24] Arduino, „From Arduino to a Microcontroller on a Breadboard,“ Arduino LLC, 5 февруар 2018. [На мрежи]. Available: https://www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoToBreadboard.

[25] „Google Maps,“ [На мрежи]. Available: https://www.google.com/maps/place/Elektrotehni%C4%8Dki+fakultet,+73,+Beograd/@44.8057991,20.4742455,17z/data=!3m1!4b1!4m5!3m4!1s0x475a7a9f59af917f:0xf9b50e9f0114cb72!8m2!3d44.8057991!4d20.4764342.

Додатак

У овом поглављу ће бити приказани кодови *Node* и *Master* система, за случај да заинтересованом читаоцу могу бити од користи. Код за сваки *Arduino* програм је подељен у четири целине: хедер кода у којем се позивају све неопходне библиотеке за коришћење у коду; *setup* који се извршава само једном после сваког ресетовања микроконтролера; *loop* који се непрекидно извршава, као и остале разне функције за помоћ при прорачуну, исписивању резултата и томе слично.

Табела 3. Комплетан код *Node* ситема.

```
#include <SPI.h>
#include <LoRa.h>
#include <DS3232RTC.h>
#include <BH1750.h>
#include <DHT.h>
#include <LowPower.h>

// ========== LORA ========== //
const int8_t lora_cs = 10;
const int8_t lora_rst = 9;
const int8_t lora_irq = 2;
const long frequency = 433E6;

// ========== RTC ========== //
const int SQW_PIN(3);
volatile bool alarmIsrWasCalled = false;

// ========== BH1750 ========== //
BH1750 lightMeter;

// ========== DHT11 ========== //
#define DHTTYPE DHT11
const int8_t dhtPin = 4;
DHT dht(dhtPin, DHTTYPE);

// ========== LED ========== //
const int8_t ledPin = 5;

// ========== MQ2 ========== //
const int8_t mqPin = 8;
const int8_t gasPin = A0;

// ========== DATA ========== //
typedef struct
{
  byte key;
  float temp;
  float humi;
  float lux;
  float gas;
  int battery;
} Data;
Data toSend = {0xB3, 0, 0, 0, 0, 0};

// ========== ROUTINES ========== //
// ISR ROUTINE
```

```
47  void wakeUp()
48  {
49    alarmIsrWasCalled = true;
50  }
51
52  // LED ROUTINE
53  void blinkLED(int8_t times, int del)
54  {
55    for (int8_t i = 0; i < times; ++i)
56    {
57      digitalWrite(ledPin, HIGH);
58      delay(del);
59      digitalWrite(ledPin, LOW);
60      delay(del);
61    }
62  }
63
64  // BATTERY LEVEL ROUTINE
65  // http://www.gammon.com.au/forum/?id=11497
66  const long InternalReferenceVoltage = 1080L;
67
68  int getBandGap()
69  {
70    ADMUX = (0<<REFS1) | (1<<REFS0) | (0<<ADLAR) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (0<<MUX0);
71    delay(50);
72    ADCSRA |= _BV(ADSC);
73    while(((ADCSRA & (1<<ADSC)) != 0));
74    int results = (((InternalReferenceVoltage * 1024L) / ADC) + 5L) / 10L;
75    return results;
76  }
77
78  // RTC ROUTINE
79  time_t compileTime()
80  {
81    const time_t FUDGE(10);
82    const char *compDate = __DATE__, *compTime = __TIME__, *months =
83  "JanFebMarAprMayJunJulAugSepOctNovDec";
84    char compMon[3], *m;
85
86    strncpy(compMon, compDate, 3);
87    compMon[3] = '\0';
88    m = strstr(months, compMon);
89
90    tmElements_t tm;
91    tm.Month = ((m - months) / 3 + 1);
92    tm.Day = atoi(compDate + 4);
93    tm.Year = atoi(compDate + 7) - 1970;
94    tm.Hour = atoi(compTime);
95    tm.Minute = atoi(compTime + 3);
96    tm.Second = atoi(compTime + 6);
97
98    time_t t = makeTime(tm);
99    return t + FUDGE;
100 }
101
102 void RTCsetup(byte alarmType)
103 {
104   RTC.setAlarm(ALM1_MATCH_DATE, 0, 0, 0, 1);
105   RTC.setAlarm(ALM2_MATCH_DATE, 0, 0, 0, 1);
```

```
106   RTC.alarm(ALARM_1);
107   RTC.alarm(ALARM_2);
108   RTC.alarmInterrupt(ALARM_1, false);
109   RTC.alarmInterrupt(ALARM_2, false);
110   RTC.squareWave(SQWAVE_NONE);
111
112   if (alarmType == 0)
113   {
114     RTC.setAlarm(ALM1_EVERY_SECOND, 0, 0, 0, 1);
115     RTC.alarm(ALARM_1);
116     RTC.alarmInterrupt(ALARM_1, true);
117   }
118   else
119   {
120     RTC.setAlarm(ALM2_EVERY_MINUTE, 0, 0, 0, 1);
121     RTC.alarm(ALARM_2);
122     RTC.alarmInterrupt(ALARM_2, true);
123   }
124
125   pinMode(SQW_PIN, INPUT_PULLUP);
126 }
127
128 // DATA ROUTINE
129 void updateData()
130 {
131   for (auto i = 0; i < 3; ++i) {
132     toSend.temp = dht.readTemperature();
133     toSend.humi = dht.readHumidity();
134
135     while (!lightMeter.measurementReady(true)) {
136      yield();
137     }
138     toSend.lux = lightMeter.readLightLevel();
139     lightMeter.configure(BH1750::ONE_TIME_HIGH_RES_MODE);
140   }
141
142   pinMode(mqPin, OUTPUT);
143   digitalWrite(mqPin, HIGH);
144   delay(200);
145   for (auto i = 0; i < 3; ++i) {
146     toSend.gas = analogRead(gasPin) * 5L / 1023L;
147   }
148   digitalWrite(mqPin, LOW);
149
150   for (auto i = 0; i < 5; ++i) {
151     toSend.battery = getBandGap();
152   }
153
154   if (toSend.gas > 2.5)
155   {
156     RTCsetup(0);
157   }
158   else
159   {
160     RTCsetup(1);
161   }
162 }
163
164 // LORA ROUTINES
```

```
165 void LoRa_sendData()
166 {
167   blinkLED(1, 500);
168   LoRa.beginPacket();
169   LoRa.write((uint8_t *)&toSend, sizeof(toSend));
170   LoRa.endPacket();
171 }
172
173 void setup()
174 {
175 //  Serial.begin(9600);
176   Wire.begin();
177
178   pinMode(ledPin, OUTPUT);
179   digitalWrite(ledPin, LOW);
180   pinMode(mqPin, OUTPUT);
181   digitalWrite(mqPin, LOW);
182   pinMode(lora_cs, OUTPUT);
183   digitalWrite(lora_cs, HIGH);
184
185   // LORA SETUP
186   LoRa.setPins(lora_cs, lora_rst, lora_irq);
187   while (!LoRa.begin(frequency))
188   {
189     blinkLED(1, 500);
190   }
191   digitalWrite(ledPin, LOW);
192
193   lightMeter.begin(BH1750::ONE_TIME_HIGH_RES_MODE); // BH1750 SETUP
194   dht.begin(); // DHT SETUP
195
196   // RTC SETUP
197   RTCsetup(1);
198   RTC.set(compileTime());
199
200 //  updateData();
201
202   blinkLED(4, 300);
203 }
204
205 void loop()
206 {
207 //  Serial.print(F("Temperature: "));     Serial.println(toSend.temp);
208 //  Serial.print(F("Humidity: "));        Serial.println(toSend.humi);
209 //  Serial.print(F("Light level: "));     Serial.println(toSend.lux);
210 //  Serial.print(F("Gas level: "));       Serial.println(toSend.gas);
211 //  Serial.print(F("Battery level: "));   Serial.println(toSend.battery);
212 //  Serial.println();
213
214   // SLEEP
215   attachInterrupt(digitalPinToInterrupt(SQW_PIN), wakeUp, LOW);
216   LoRa.sleep();
217   delay(200);
218   LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);
219
220   // WAKE UP AND SEND DATA
221   detachInterrupt(digitalPinToInterrupt(SQW_PIN));
222   LoRa.idle();
223   delay(200);
```

```
  if (alarmIsrWasCalled) {
    if (RTC.alarm(ALARM_1) || RTC.alarm(ALARM_2)) {
      updateData();
    }
    alarmIsrWasCalled = false;
  }
  pinMode(ledPin, OUTPUT);
  LoRa_sendData();

  // SET ALL PINS TO INPUT AND LOW
  pinMode(ledPin, INPUT);
  digitalWrite(ledPin, LOW);
  pinMode(mqPin, INPUT);
  digitalWrite(mqPin, LOW);
}
```

Табела 4. Главни код *Master* система.

```
#include <SPI.h>
#include <SD.h>
#include <LoRa.h>
#include <Wire.h>
#include <DS3232RTC.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// === SD === //
File myFile;
const int8_t sd_cs = SS;

// === LORA === //
const int8_t lora_cs = 10;
const int8_t lora_rst = 9;
const int8_t lora_irq = 2;
const long freq = 433E6;

// === OLED === //
const uint8_t SCREEN_WIDTH = 128;
const uint8_t SCREEN_HEIGHT = 64;
const int8_t oled_mosi = 38;
const int8_t oled_sck = 40;
const int8_t oled_dc = 42;
const int8_t oled_cs = 44;
const int8_t oled_rst = 46;
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT,
                      oled_mosi, oled_sck, oled_dc, oled_rst, oled_cs);
/*
   The default hardware SPI was overwritten to software SPI
   because of conflicts with the SD board
*/

// === BUTTONS === //
const int8_t UP = 17;
const int8_t DOWN = 16;
const int8_t ENTR = 18;
const int8_t MENU = 19;
const uint8_t DEBOUNCE = 250;
```

```
volatile byte menuPress = 0;

// === DATA === //
const int MAX_NODES = 3;
const byte securityKeys[MAX_NODES] = {0xB3, 0x4E, 0x38};  // Used to differentiate between
nodes

typedef struct {
  byte key;
  float temp;
  float humi;
  float lux;
  float gas;
  int battery;
} nodeData;

const char* names[] {
  "Node 1",
  "Node 2",
  "Node 3"
};

typedef struct {
  nodeData data;
  float rssi;
  float snr;
  const char* nodeName;
} nodeInfo;
nodeInfo nodes[MAX_NODES];

byte celsius = 1; // Display temperature in C
byte displayMode = 1; // Display single node
int8_t nodeToDisplay = 0; // Display the first node
bool updated = false; // Set the display to NOT_UPDATED

// === ROUTINES === //
void (* resetFunc) (void) = 0;
// Interrupt routine
void func() {
  menuPress = 1;
}

// Write names of all nodes to a default value
void writeNames() {
  for (size_t i = 0; i < MAX_NODES; ++i) {
    nodes[i].nodeName = names[i];
    nodes[i].data.key = securityKeys[i];
  }
}

// How to write data to text file
void writeData(byte i) {
  char buff[16];
  sprintf(buff, "%d%d.txt", month(RTC.get()), year(RTC.get()));
  const char* fileName = buff;
  if (myFile = SD.open(fileName, FILE_WRITE)) {
    myFile.print(hour(RTC.get()));
    myFile.print(':');
    myFile.print(minute(RTC.get()));
```

```
    myFile.print(':');
    myFile.print(second(RTC.get()));
    myFile.print(' ');
    myFile.print(day(RTC.get()));
    myFile.print('/');
    myFile.print(month(RTC.get()));
    myFile.print('/');
    myFile.print(year(RTC.get()));
    myFile.print(' ');
    myFile.print(nodes[i].data.key);
    myFile.print(' ');
    myFile.print(nodes[i].nodeName);
    myFile.print(' ');
    myFile.print(nodes[i].data.temp);
    myFile.print(' ');
    myFile.print(nodes[i].data.humi);
    myFile.print(' ');
    myFile.print(nodes[i].data.lux);
    myFile.print(' ');
    myFile.print(nodes[i].data.gas);
    myFile.print(' ');
    myFile.print(nodes[i].data.battery);
    myFile.print(' ');
    myFile.print(nodes[i].snr);
    myFile.print(' ');
    myFile.println(nodes[i].rssi);

    myFile.flush();
    myFile.close();
  }
  else {
    resetFunc();
    // Serial.println(F("Something went wrong while writing..."));
  }
}

volatile bool written = true;
byte toWrite = 0;

// What to do when receiving a message
void onReceive(int packetSize) {
  // Serial.println(F("Packet..."));
  if (packetSize) {
    nodeData tmp;
    LoRa.readBytes((uint8_t*)&tmp, packetSize);
    float rssi_temp = LoRa.packetRssi();
    float snr_temp = LoRa.packetSnr();
    for (size_t i = 0; i < MAX_NODES; ++i) {
      if (nodes[i].data.key == tmp.key) {
        nodes[i].data = tmp;
        nodes[i].rssi = rssi_temp;
        nodes[i].snr = snr_temp;

        updated = false;
        written = false;
        toWrite = i;
      }
    }
  }
```

```
158     else {
159       resetFunc();
160     }
161   }
162
163   // Display node specific information
164   void displayNode(int8_t n) {
165     displayHeader(nodes[n].nodeName);
166
167     oled.setCursor(3, 24);
168     oled.print(F("Temp: "));
169     if (celsius) {
170       oled.print(nodes[n].data.temp);
171       oled.print(" ");
172       oled.print((char)223);
173       oled.print(F("C"));
174     }
175     else {
176       oled.print(nodes[n].data.temp * 9 / 5 + 32);
177       oled.print(" ");
178       oled.print((char)223);
179       oled.print(F("F"));
180     }
181     oled.setCursor(3, 34);
182     oled.print(F("Humi: "));
183     oled.print(nodes[n].data.humi);
184     oled.print(F(" %"));
185     oled.setCursor(3, 44);
186     oled.print(F("Light: "));
187     oled.print(nodes[n].data.lux);
188     oled.print(F(" lux"));
189
190     time_t t = RTC.get();
191     oled.setCursor(94, 54);
192     if (hour(t) < 10) {
193       oled.print(F("0"));
194       oled.print(hour(t));
195       oled.print(F(":"));
196     }
197     else {
198       oled.print(hour(t));
199       oled.print(F(":"));
200     }
201
202     if (minute(t) < 10) {
203       oled.print(F("0"));
204       oled.print(minute(t));
205     }
206     else {
207       oled.print(minute(t));
208     }
209
210     oled.display();
211   }
212
213   void updateDisplay() {
214     oled.cp437(true);
215
216     if (displayMode == 1) {
```

```
217       displayNode(nodeToDisplay);
218     }
219
220     if (displayMode == 2) {
221       if (second(RTC.get()) % 5 == 0) {
222         displayNode(nodeToDisplay);
223         if (++nodeToDisplay > 2) {
224           nodeToDisplay = 0;
225         }
226         while (second(RTC.get()) % 5 == 0);
227       }
228     }
229   }
230
231   void setup() {
232     // Serial.begin(9600);
233     pinMode(lora_cs, OUTPUT);
234     pinMode(oled_cs, OUTPUT);
235     pinMode(sd_cs, OUTPUT);
236     digitalWrite(lora_cs, HIGH);
237     digitalWrite(oled_cs, HIGH);
238     digitalWrite(sd_cs, HIGH);
239
240     while (!SD.begin(sd_cs)) {
241       // Serial.print(F("."));
242       delay(500);
243     }
244     // Serial.println(F("SD initialization complete."));
245
246     LoRa.setPins(lora_cs, lora_rst, lora_irq);
247     while (!LoRa.begin(freq)) {
248       // Serial.print(F("."));
249       delay(500);
250     }
251     // Serial.println(F("LoRa initialization complete."));
252     LoRa.onReceive(onReceive);
253     LoRa.receive();
254     writeNames();
255
256     while (!oled.begin(SSD1306_SWITCHCAPVCC)) {
257       // Serial.print(F("."));
258       delay(500);
259     }
260     // Serial.println(F("Display initialization complete."));
261     oled.clearDisplay();
262
263     pinMode(UP, INPUT_PULLUP);
264     pinMode(DOWN, INPUT_PULLUP);
265     pinMode(ENTR, INPUT_PULLUP);
266     pinMode(MENU, INPUT_PULLUP);
267     attachInterrupt(digitalPinToInterrupt(MENU), func, FALLING);
268
269     RTC.setAlarm(ALM1_MATCH_DATE, 0, 0, 0, 1);
270     RTC.setAlarm(ALM2_MATCH_DATE, 0, 0, 0, 1);
271     RTC.alarm(ALARM_1);
272     RTC.alarm(ALARM_2);
273     RTC.alarmInterrupt(ALARM_1, false);
274     RTC.alarmInterrupt(ALARM_2, false);
275     RTC.squareWave(SQWAVE_NONE);
```

```
}

void loop() {
  if (second(RTC.get()) == 0) {
    LoRa.receive();
    updated = false;
    while (second(RTC.get()) == 0);
  }

  if (!updated) {
    updateDisplay();
    if (displayMode == 1) {
      updated = true;
    }
  }

  if (menuPress) {
    menu();
    menuPress = 0;
  }

  if (!written) {
    writeData(toWrite);
    written = true;
  }
}
```

Табела 5. Код мени система.

```
// Used for calculating the width and height of the header
int16_t x, y;
uint16_t w, h;

// Setting how many menu pages and rows per page there are
int8_t page = 1;
const int8_t pages = 2;
int8_t row = 1;
const int8_t maxRows[2] = {4, 3};

// Menu items as they will appear on each page
const char* menuItems[] = {
  "Set Display Mode",
  "Display Battery",
  "Display Gas",
  "Display SNR",
  "Display RSSI",
//  "Set Node Name",
  "Set Temp Units",
  "Set Time",
};

// Displaying the header for the menu/sub-menu in a rectangular box
void displayHeader(const char* header) {
  oled.clearDisplay();
  oled.setTextSize(1);
  oled.setTextColor(WHITE);
  oled.getTextBounds(header, 0, 0, &x, &y, &w, &h);
```

```
29    oled.setCursor((oled.width() - w) / 2, 3);
30    oled.print(header);
31    oled.drawRect((oled.width() - w - 8) / 2, 0, w + 8, h + 6, WHITE);
32    x = 0;
33    y = 0;
34    w = 0;
35    h = 0;
36    oled.display();
37  }
38
39  // Displaying menu items
40  void displayMenuItems(int8_t p) {
41    oled.fillRect(10, 24, oled.width() - 10, oled.height() - 24, BLACK);
42    oled.fillRect(110, 3, oled.width() - 110, 10, BLACK);
43
44    if (p == 1) {
45      for (size_t i = 0; i < maxRows[0]; ++i) {
46        oled.setCursor(10, 24 + 10 * i);
47        oled.print(F("| "));
48        oled.print(menuItems[i]);
49      }
50      oled.setCursor(110, 3);
51      oled.print(F("1/2"));
52      oled.display();
53    }
54    else {
55      for (size_t i = 0; i < maxRows[1]; ++i) {
56        oled.setCursor(10, 24 + 10 * i);
57        oled.print(F("| "));
58        oled.print(menuItems[4 + i]);
59      }
60      oled.setCursor(110, 3);
61      oled.print(F("2/2"));
62      oled.display();
63    }
64  }
65
66  // Helper function for displaying arrows when selecting items within a menu
67  void displayArrow(int8_t x) {
68    oled.fillRect(0, 24, 9, oled.height() - 9, BLACK);
69    oled.setCursor(0, 24 + 10 * (x - 1));
70    oled.print(F(">"));
71    oled.display();
72  }
73
74  // Displaying submenus
75  void displaySubMenu(int8_t p, int8_t r) {
76    int8_t toDisplay = r + 4 * p - 5;
77    oled.clearDisplay();
78
79    switch (toDisplay + 1) {
80      case 0: {
81          break;
82        }
83
84      // Set display mode to show single node or to cycle nodes
85      case 1: {
86          displayHeader("SET DISP");
87
```

```
88          oled.setCursor(0, 24);
89          oled.print(F(">"));
90          oled.setCursor(10, 24);
91          oled.print(F("| Single Node"));
92          oled.setCursor(10, 34);
93          oled.print(F("| Cycle Nodes"));
94          oled.display();
95
96          int8_t n = 1;
97
98          while (true) {
99            if (!digitalRead(UP)) {
100             delay(DEBOUNCE);
101             if (++n > 2) {
102               n = 1;
103             }
104             displayArrow(n);
105           }
106
107           if (!digitalRead(DOWN)) {
108             delay(DEBOUNCE);
109             if (--n < 1) {
110               n = 2;
111             }
112             displayArrow(n);
113           }
114
115           if (!digitalRead(ENTR)) {
116             delay(DEBOUNCE);
117             if (n == 1) {
118               displayHeader("SEL NODE");
119
120               oled.setCursor(0, 24);
121               oled.print(F(">"));
122               for (int8_t i = 0; i < MAX_NODES; ++i) {
123                 oled.setCursor(10, 24 + 10 * i);
124                 oled.print(F("| "));
125                 oled.print(nodes[i].nodeName);
126               }
127               oled.display();
128
129               int8_t node = 1;
130
131               while (true) {
132                 if (!digitalRead(UP)) {
133                   delay(DEBOUNCE);
134                   if (++node > MAX_NODES) {
135                     node = 1;
136                   }
137                   displayArrow(node);
138                 }
139
140                 if (!digitalRead(DOWN)) {
141                   delay(DEBOUNCE);
142                   if (--node < 1) {
143                     node = MAX_NODES;
144                   }
145                   displayArrow(node);
146                 }
```

```
147
148                 if (!digitalRead(ENTR)) {
149                   delay(DEBOUNCE);
150                   displayMode = n;
151                   nodeToDisplay = node - 1;
152                   break;
153                 }
154               }
155             break;
156           }
157           else {
158             displayMode = n;
159             break;
160           }
161         }
162       }
163       break;
164     }
165
166     // Display battery for all nodes
167     case 2: {
168       displayHeader("DISP BATT");
169
170       for (size_t i = 0; i < MAX_NODES; ++i) {
171         oled.setCursor(3, 24 + 10 * i);
172         oled.print(nodes[i].nodeName);
173         oled.print(F(": "));
174         oled.print(nodes[i].data.battery);
175       }
176       oled.display();
177
178       while (true) {
179         if (!digitalRead(ENTR)) {
180           delay(DEBOUNCE);
181           break;
182         }
183       }
184       break;
185     }
186
187     // Display gas level of all nodes
188     case 3: {
189       displayHeader("DISP GAS");
190
191       for (size_t i = 0; i < MAX_NODES; ++i) {
192         oled.setCursor(3, 24 + 10 * i);
193         oled.print(nodes[i].nodeName);
194         oled.print(F(": "));
195         oled.print(nodes[i].data.gas);
196       }
197       oled.display();
198
199       while (true) {
200         if (!digitalRead(ENTR)) {
201           delay(DEBOUNCE);
202           break;
203         }
204       }
205       break;
```

```
206       }
207
208     // Display SNR for all nodes
209     case 4: {
210         displayHeader("DISP SNR");
211
212         for (size_t i = 0; i < MAX_NODES; ++i) {
213           oled.setCursor(3, 24 + 10 * i);
214           oled.print(nodes[i].nodeName);
215           oled.print(F(": "));
216           oled.print(nodes[i].snr);
217           oled.print(F(" dB"));
218         }
219         oled.display();
220
221         while (true) {
222           if (!digitalRead(ENTR)) {
223             delay(DEBOUNCE);
224             break;
225           }
226         }
227         break;
228       }
229
230     // Display RSSI for all nodes
231     case 5: {
232         displayHeader("DISP RSSI");
233
234         for (size_t i = 0; i < MAX_NODES; ++i) {
235           oled.setCursor(3, 24 + 10 * i);
236           oled.print(nodes[i].nodeName);
237           oled.print(F(": "));
238           oled.print(nodes[i].rssi);
239           oled.print(F(" dBm"));
240         }
241         oled.display();
242
243         while (true) {
244           if (!digitalRead(ENTR)) {
245             delay(DEBOUNCE);
246             break;
247           }
248         }
249         break;
250       }
251
252     // Change the node names
253     // 0-9, A-Z, a-z supported only
254 //    case 6: {
255 //        displayHeader("NOD NAME");
256 //
257 //        oled.setCursor(0, 24);
258 //        oled.print(">");
259 //        for (size_t n = 0; n < MAX_NODES; ++n) {
260 //          oled.setCursor(10, 24 + 10 * n);
261 //          oled.print("| " + String(nodes[n].nodeName));
262 //        }
263 //        oled.display();
264 //
```

```
265 //        int8_t node = 1;
266 //
267 //        while (true) {
268 //          if (!digitalRead(UP)) {
269 //            delay(DEBOUNCE);
270 //            if (++node > MAX_NODES) {
271 //              node = 1;
272 //            }
273 //            displayArrow(node);
274 //          }
275 //
276 //          if (!digitalRead(DOWN)) {
277 //            delay(DEBOUNCE);
278 //            if (--node < 1) {
279 //              node = MAX_NODES;
280 //            }
281 //            displayArrow(node);
282 //          }
283 //
284 //          if (!digitalRead(ENTR)) {
285 //            delay(DEBOUNCE);
286 //            oled.clearDisplay();
287 //
288 //            oled.setCursor(3, 10);
289 //            oled.print("Old name: ");
290 //            oled.setCursor(60, 10);
291 //            oled.print(nodes[node - 1].nodeName);
292 //            oled.setCursor(3, 30);
293 //            oled.print("New name: ");
294 //            oled.display();
295 //
296 //            int8_t i = 0;
297 //            int8_t X = 60;
298 //            int8_t counter = 47;
299 //            char tempName[6];
300 //
301 //            while (true) {
302 //              if (!digitalRead(UP)) {
303 //                delay(DEBOUNCE);
304 //                ++counter;
305 //                if (counter < 48) {
306 //                  counter = 48;
307 //                }
308 //                if (counter > 57 && counter < 65) {
309 //                  counter = 65;
310 //                }
311 //                if (counter > 90 && counter < 97) {
312 //                  counter = 97;
313 //                }
314 //                if (counter > 122) {
315 //                  counter = 48;
316 //                }
317 //                oled.fillRect(X, 30, X + 6, 38, BLACK);
318 //                oled.setCursor(X, 30);
319 //                oled.print((char)counter);
320 //                oled.display();
321 //              }
322 //
323 //              if (!digitalRead(DOWN)) {
```

```
324 //                delay(DEBOUNCE);
325 //                --counter;
326 //                if (counter < 48) {
327 //                  counter = 122;
328 //                }
329 //                if (counter > 57 && counter < 65) {
330 //                  counter = 57;
331 //                }
332 //                if (counter > 90 && counter < 97) {
333 //                  counter = 90;
334 //                }
335 //                if (counter > 122) {
336 //                  counter = 122;
337 //                }
338 //                oled.fillRect(X, 30, X + 6, 38, BLACK);
339 //                oled.setCursor(X, 30);
340 //                oled.print((char)counter);
341 //                oled.display();
342 //              }
343 //
344 //              if (!digitalRead(ENTR)) {
345 //                delay(DEBOUNCE);
346 //                X += 6;
347 //                tempName[i] = (char)counter;
348 //                i++;
349 //
350 //                if (i == 5) {
351 //                  tempName[i + 1] = '\0';
352 //                  nodes[node - 1].nodeName = tempName;
353 //                  break;
354 //                }
355 //              }
356 //            }
357 //            break;
358 //          }
359 //        }
360 //        break;
361 //      }
362
363     // Change temp units between C and F
364     case 6: {
365         displayHeader("TEMP U");
366
367         oled.fillRect(0, 22, oled.width(), oled.height(), BLACK);
368         oled.setCursor(oled.width() / 2, oled.height() / 2);
369         if (celsius) {
370           oled.print(F("C"));
371         }
372         else {
373           oled.print(F("F"));
374         }
375         oled.display();
376
377         while (true) {
378           if (!digitalRead(UP) || !digitalRead(DOWN)) {
379             delay(DEBOUNCE);
380             if (celsius) {
381               celsius = 0;
382               oled.fillRect(0, 22, oled.width(), oled.height(), BLACK);
```

```
383               oled.setCursor(oled.width() / 2, oled.height() / 2);
384               oled.print(F("F"));
385             }
386             else {
387               celsius = 1;
388               oled.fillRect(0, 22, oled.width(), oled.height(), BLACK);
389               oled.setCursor(oled.width() / 2, oled.height() / 2);
390               oled.print(F("C"));
391             }
392             oled.display();
393           }
394
395           if (!digitalRead(ENTR)) {
396             delay(DEBOUNCE);
397             break;
398           }
399         }
400         break;
401       }
402
403     // Set the time on the master RTC not the nodes
404     case 7: {
405         displayHeader("SET TIME");
406
407         byte i = 0;
408         int h = 0;
409         int m = 0;
410         int s = 0;
411         int D = 1;
412         int M = 1;
413         int Y = 2020;
414
415         while (true) {
416           if (!digitalRead(DOWN)) {
417
418             delay(DEBOUNCE);
419             oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
420             if (i == 0) {
421               if (++Y > 2025) {
422                 Y = 2015;
423               }
424               oled.setCursor(3, 24);
425               oled.print(F("Year: "));
426               oled.print(Y);
427               oled.display();
428
429             }
430
431             if (i == 1) {
432               oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
433
434               if (++M > 12) {
435                 M = 1;
436               }
437               oled.setCursor(3, 24);
438               oled.print(F("Month: "));
439               oled.print(M);
440               oled.display();
441             }
```

```
442
443             if (i == 2) {
444               oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
445
446               if (++D > 31) {
447                 D = 1;
448               }
449               oled.setCursor(3, 24);
450               oled.print(F("Day: "));
451               oled.print(D);
452               oled.display();
453             }
454
455             if (i == 3) {
456               oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
457
458               if (++h > 23) {
459                 h = 0;
460               }
461               oled.setCursor(3, 24);
462               oled.print(F("Hour: "));
463               oled.print(h);
464               oled.display();
465             }
466
467             if (i == 4) {
468               oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
469
470               if (++m > 59) {
471                 m = 0;
472               }
473               oled.setCursor(3, 24);
474               oled.print(F("Minute: "));
475               oled.print(m);
476               oled.display();
477             }
478
479             if (i == 5) {
480               oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
481
482               if (++s > 59) {
483                 s = 0;
484               }
485               oled.setCursor(3, 24);
486               oled.print(F("Second: "));
487               oled.print(s);
488               oled.display();
489             }
490           }
491
492           if (!digitalRead(UP)) {
493             delay(DEBOUNCE);
494             if (i == 0) {
495               oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
496
497               if (--Y < 2015) {
498                 Y = 2025;
499               }
500               oled.setCursor(3, 24);
```

```
501                 oled.print(F("Year: "));
502                 oled.print(Y);
503                 oled.display();
504               }
505
506               if (i == 1) {
507                 oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
508
509                 if (--M < 1) {
510                   M = 12;
511                 }
512                 oled.setCursor(3, 24);
513                 oled.print(F("Month: "));
514                 oled.print(M);
515                 oled.display();
516               }
517
518               if (i == 2) {
519                 oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
520
521                 if (--D < 1) {
522                   D = 31;
523                 }
524                 oled.setCursor(3, 24);
525                 oled.print(F("Day: "));
526                 oled.print(D);
527                 oled.display();
528               }
529
530               if (i == 3) {
531                 oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
532
533                 if (--h < 0) {
534                   h = 23;
535                 }
536                 oled.setCursor(3, 24);
537                 oled.print(F("Hour: "));
538                 oled.print(h);
539                 oled.display();
540               }
541
542               if (i == 4) {
543                 oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
544
545                 if (--m < 0) {
546                   m = 59;
547                 }
548                 oled.setCursor(3, 24);
549                 oled.print(F("Minute: "));
550                 oled.print(m);
551                 oled.display();
552               }
553
554               if (i == 5) {
555                 oled.fillRect(3, 20, oled.width(), oled.height(), BLACK);
556
557                 if (--s < 0) {
558                   s = 59;
559                 }
```

```
560                 oled.setCursor(3, 24);
561                 oled.print(F("Second: "));
562                 oled.print(s);
563                 oled.display();
564               }
565             }
566
567             if (!digitalRead(ENTR)) {
568               delay(DEBOUNCE);
569               i += 1;
570               if (i > 5) {
571                 break;
572               }
573             }
574           }
575
576           tmElements_t tm;
577           tm.Hour = h;
578           tm.Minute = m;
579           tm.Second = s;
580           tm.Day = D;
581           tm.Month = M;
582           tm.Year = Y - 1970;
583           RTC.set(makeTime(tm));
584
585           break;
586         }
587       }
588     }
589
590     // The menu routine
591     void menu() {
592       updated = false;
593
594       delay(DEBOUNCE);
595       displayHeader("MENU");
596
597       oled.setCursor(0, 24 + 10 * (row - 1));
598       oled.print(F(">"));
599       displayMenuItems(page);
600
601       while (true) {
602         if (!digitalRead(UP)) {
603           delay(DEBOUNCE);
604           row += 1;
605           if (page == 1) {
606             if (row > maxRows[0]) {
607               page = 2;
608               row = 1;
609             }
610           }
611           else if (page == 2) {
612             if (row > maxRows[1]) {
613               page = 1;
614               row = 1;
615             }
616           }
617           displayMenuItems(page);
618           displayArrow(row);
```

```
619       }
620
621       if (!digitalRead(DOWN)) {
622         delay(DEBOUNCE);
623         row -= 1;
624         if (page == 1) {
625           if (row < 1) {
626             row = maxRows[1];
627             page = 2;
628           }
629         }
630         else if (page == 2) {
631           if (row < 1) {
632             row = maxRows[0];
633             page = 1;
634           }
635         }
636         displayMenuItems(page);
637         displayArrow(row);
638       }
639
640       if (!digitalRead(ENTR)) {
641         delay(DEBOUNCE);
642         displaySubMenu(page, row);
643         break;
644       }
645     }
646   }
```