Stație Meteorologică

Introducere

Proiectul constă în construirea unei stații meteorologice care colectează informații din mediul înconjurător și le afișează pe ecran. Utilizatorul poate interacționa cu stația prin intermediul unor butoane, putând accesa, astfel, mai multe ecrane care oferă diverse funcționalități. Atunci când stația este folosită, va răspunde rapid la comenzile utilizatorului, iar când nu este folosită va consuma cât mai puțină energie prin reducerea comunicațiilor cu senzorii și display-ul.

Funcționalități oferite:

- Ecran principal pe care sunt afișate temperatura, umiditatea, presiunea și data curentă
- Grafic care arată evoluția presiunii în ultimele douăzeci de ore
- Meniu în care se afișează valorile minime și maxime înregistrate pe parcursul orei curente sau pe ultimele 24 de ore
- Setări pentru dată, oră, unități de măsură pentru temperatură și presiune, corecție pentru presiune
- Două alarme, cu ora fixată, dar care pot fi făcute să pornească doar în unele zile ale săptămânii. Fiecare poate porni un dispozitiv extern controlat prin infraroșu și îl poate opri după un timp ales de utilizator

Am decis să realizez acest proiect deoarece am deja o stație meteorologică comercială și aș vrea să văd care sunt provocările construirii unui asemenea sistem. În plus, pe viitor, vreau să pot adăuga funcționalități suplimentare la acest proiect, precum un modul bluetooth pentru sincronizarea cu un telefon sau o altă placă care să colecteze informații din exteriorul locuintei.

Descriere generală

Interactiunea dintre modulele hardware

Toate componentele care pot comunica prin I2C (senzorul de presiune, ceasul și display-ul) vor fi legate prin intermediul acestei magistrale. Senzorul DHT11 folosește un protocol one-wire și va comunica umiditatea plăcuței printr-o conexiune individuală. Senzorul BMP280 va fi folosit pentru presiune și temperatură, iar DHT11 doar pentru umiditate, deoarece este mai lent și mai imprecis decât BMP280. Pentru a reduce numărul de componente, butoanele vor fi legate la arduino folosind rezistențele interne de pe pini.

Pentru a menține timpul sincronizat cu plăcuța și pentru a evita comunicația excesivă cu ceasul extern, acesta va mai avea o conexiune cu placa care va fi folosită pentru a genera o întrerupere o

dată pe secundă. Deși plăcuța poate genera întreruperi pe baza ceasurilor interne, acestea nu sunt la fel de precise ca cele generate de cel extern. Astfel, plăcuța va incrementa intern minutul și secunda și va prelua data de la ceasul extern doar o dată pe oră.

Interacțiunea dintre modulele hardware și cele software

În albastru deschis sunt reprezentate modulele hardware externe, în albastru deschis cele hardware interne, iar în verde modulele software.

Descriere sumară a modulelor software:

- `main`: Modulul principal care inițializează celelalte module și în care se găsește funcția loop
- `station`: Se ocupă de citirea și stocarea datelor de la senzorii de mediu, aici sunt salvate și o parte din setări
- `button`: Se ocupă de citirea butoanelor. Folosind întreruperi, semnalează modulului `main` dacă a fost apăsat un buton și se folosește de unul din ceasurile arduino-ului pentru a face debouncing
- `menu`: Conține implementarea pentru toate ecranele disponibile. Fiecare ecran implementează câte una din funcționalitățile descrise mai sus și interacționează cu alte module atunci când este nevoie
- `time`: Realizează sincronizarea cu ceasul extern și controlează celelalte ceasuri interne ale microcontroller-ului
- `alarm`: Se folosește de `time` pentru a implementa metodele necesare pentru a configura și a declanșa o alarmă
- `remote`: Interacționează cu receptorul și emițătorul infraroșu, salvează și retrimite la cerere un cod primit de la o telecomandă

Interacțiunea dintre modulele software

De exemplu, în schema de mai sus, modulul `main` depinde de `button`, `time` și `menu`.

Hardware Design

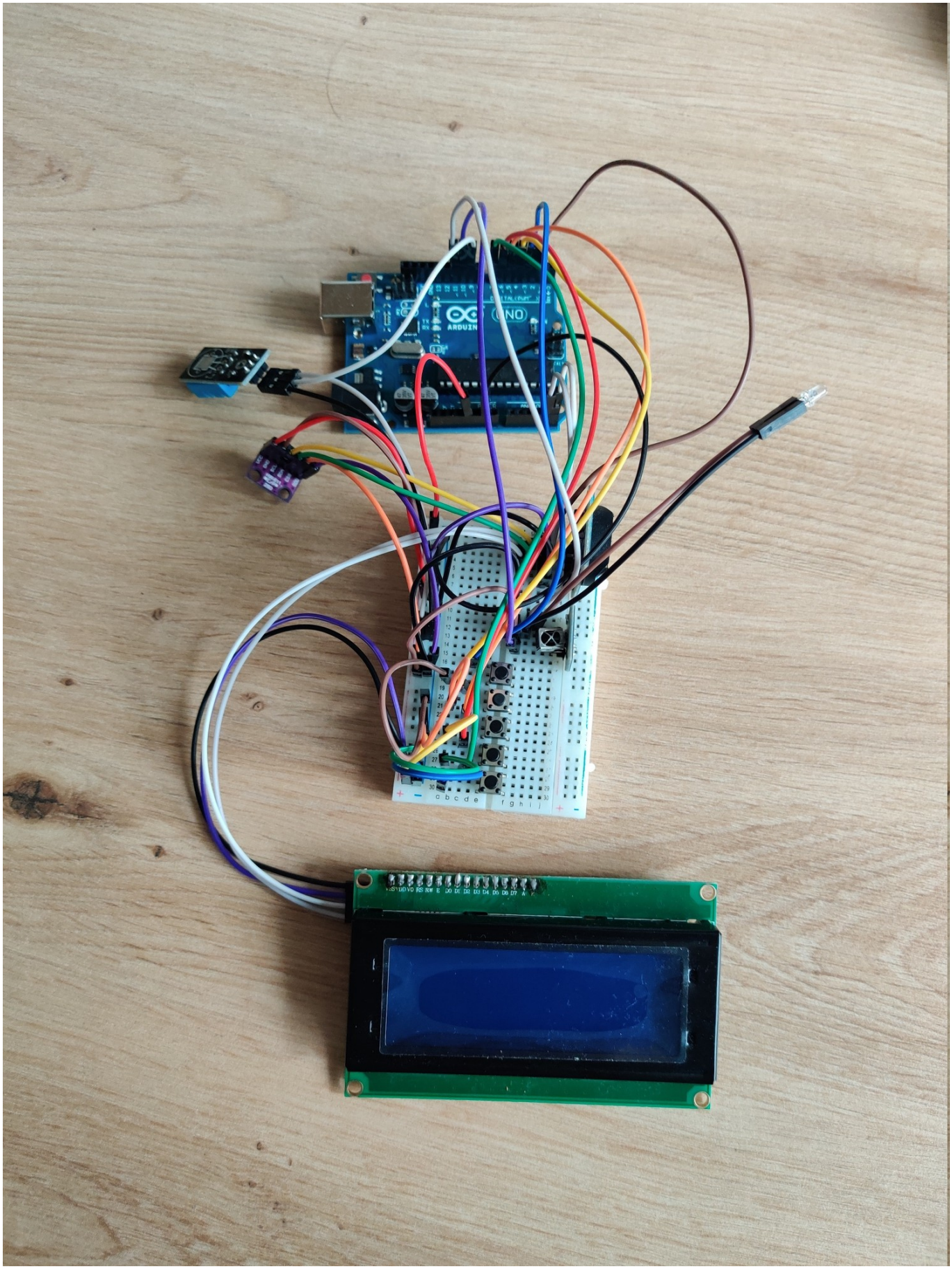
Listă de piese:

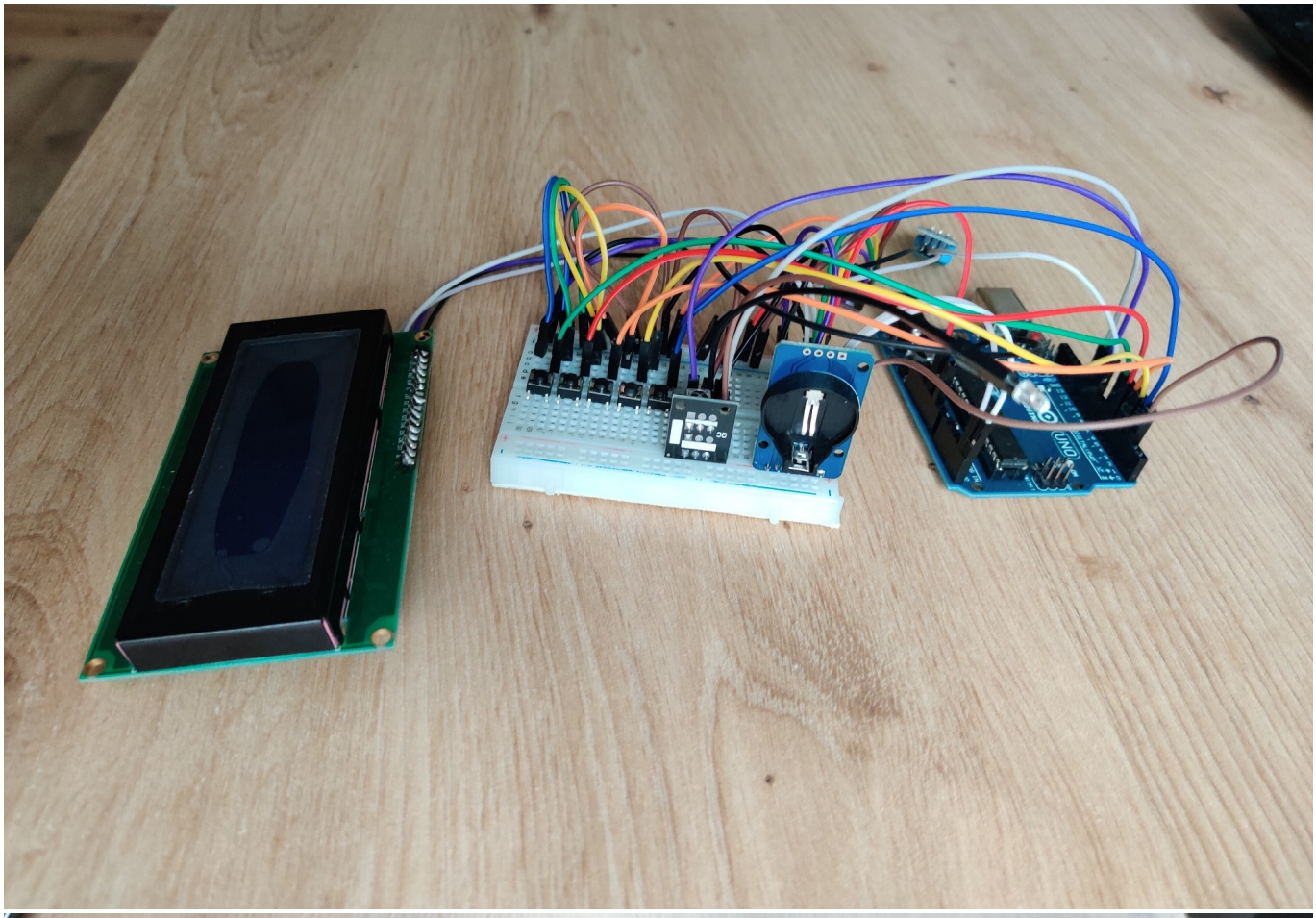
- Arduino UNO R3
- Senzor Bosch BMP280
- Senzor DHT11
- Display 20×04
- Modul I/O I2C pentru display-uri
- Ceas RTC DS3231
- Diodă infraroșu emițător
- Modul receptor infraroșu

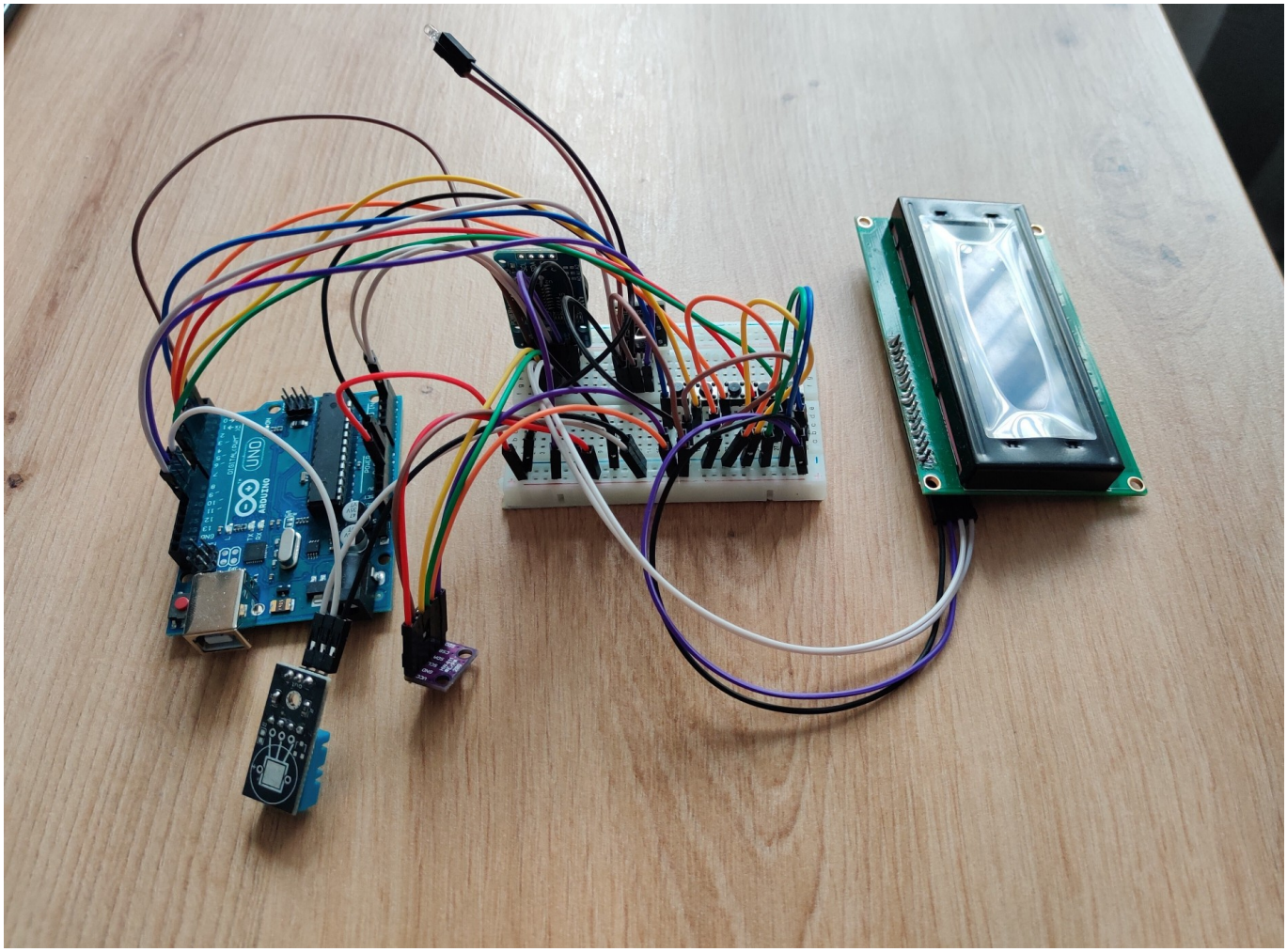
- 5 butoane

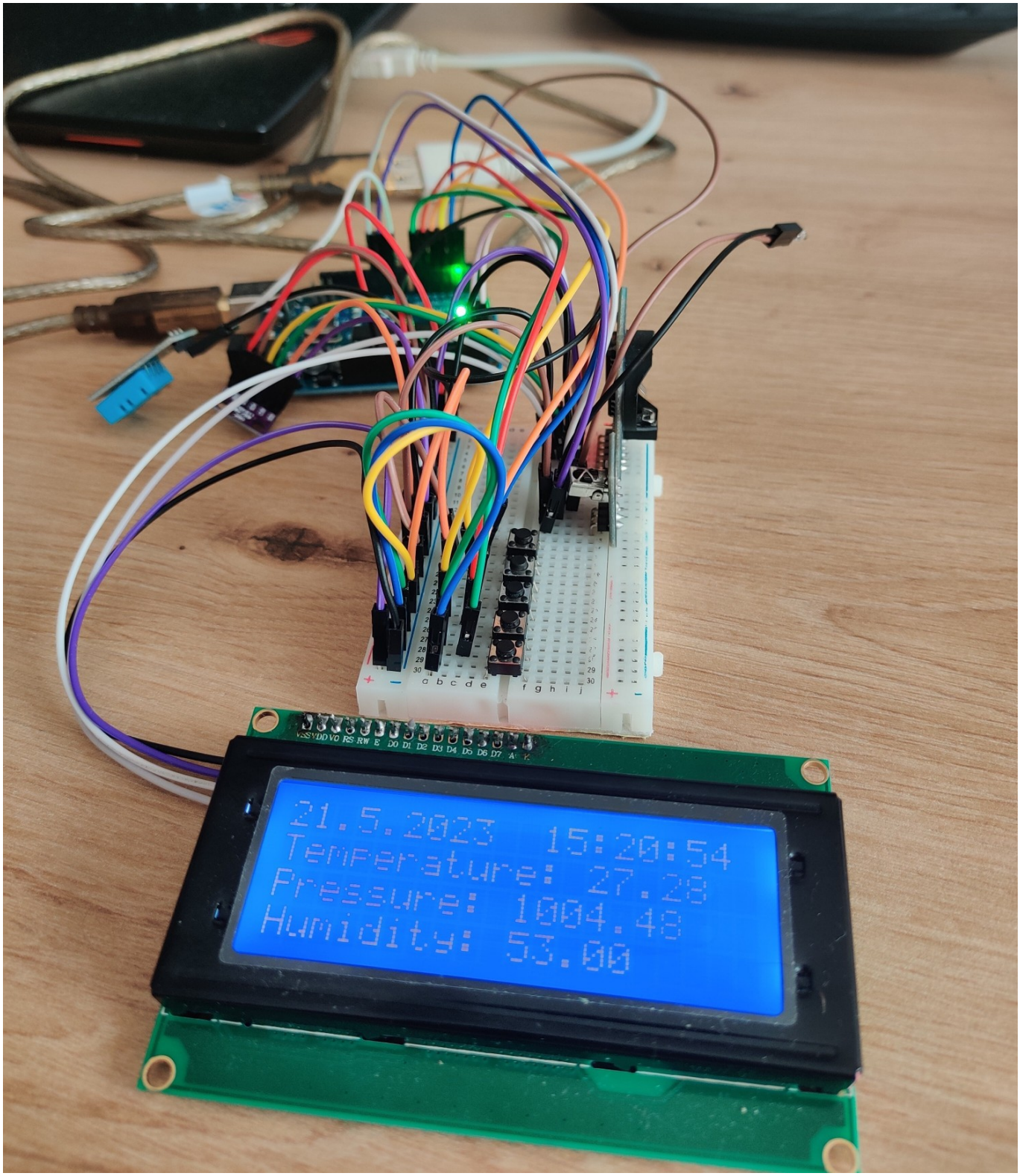
Schemă:

În urma realizării legăturilor între componente:









Codul folosit pentru imaginea de mai sus

- basic_test

Software Design

Mediu de dezvoltare

Arduino IDE v1.18

Biblioteci folosite

- RTClib
- Adafruit_BMP280
- DHT sensor library
- IRremote
- LiquidCrystal_I2C

Unele biblioteci includeau funcții pe care nu le foloseam sau foloseau la rând funcțiile millis() și delay() care au nevoie de timer-ul 0. Am modificat aceste biblioteci pentru a reduce dimensiunea ocupată dar și dependența de funcțiile care se bazează pe timer-ul 0. Mai multe modificări sunt detaliate în descrierea modulelor, unde este cazul.

Pentru biblioteca LiquidCrystal_I2C, am implementat deplasarea cursorului la stânga și la dreapta pentru că aceste funcționalități lipseau din biblioteca originală.

Descrierea modulelor software

Time

Fișiere: time.h și time.cpp
Biblioteci: RTClib

Modulul conține mai multe funcții și variabile globale care ajută comunicarea și sincronizarea cu ceasul extern DS3231. Printre altele, mai există și funcționalități pentru activarea și dezactivarea timer-elor interne, de exemplu:

```
void enableTimer2() {
  PRR &= ~(1 << PRTIM2);
}

void powerDownTimer2() {
  TCCR2B = 0;
```

```
  PRR |= (1 << PRTIM2);
}
```

Similar, există și o serie de funcții pentru timer-ul 0, doar că pentru acest timer, se păstrează și se restaurează variabila TCCR0B, care este necesară pentru functiile millis() și delay(), folosite de biblioteca IRremote. Timer-ul 2 este folosit tot de biblioteca IRremote, dar aceasta îl configurează cu parametrii corespunzători, deci nu este nevoie să se salveze registrele de configurare pentru acesta.

Pentru a reduce comunicările peste I2C cu ceasul extern, modulul păstrează minutul, secunda și data curentă în memoria locală, și comunică cu ceasul o dată pe oră pentru a resincroniza aceste variabile. Funcția syncDate() este apelată de modulul `main` de fiecare dată când trece o secundă.

```
bool syncDate() {
  bool ret = false;
  if (seconds == 60) {
    seconds = 0;
    minutes++;

    if (minutes == 60) {
      // rtc este un obiect de tipul RTC_DS3231 si reprezinta ceasul extern
      // metoda now() preia data intreaga de la ceas
      currentDate = rtc.now();

      minutes = currentDate.minute();
      seconds = currentDate.second();

      ret = true;
    }
  }

  return ret;
}
```

Secunda este incrementată odată cu apelarea întreruperii externe:

```
ISR(INT0_vect) {
  seconds++;
  readSensors++;
  alarm = true;
}
```

Variabila alarm indică faptul că ceasul a declanșat întreruperea externă, deci a trecut o secundă. Modulul `main` va reseta acest indicator odată ce începe procesările care trebuie făcute în fiecare secundă.

```
void clearTickAlarm() {
  // Se semnalează ceasului să dezactiveze alarma,
  // ceea ce va face ca semnalul de pe pin să revină
  // pe valoarea logică 1
  rtc.clearAlarm(1);
  alarm = 0;
```

```
}
```

Variabila readSensors este folosită pentru a putea mări intervalul de timp la care se citesc senzorii externi fără a fi nevoie de funcționalitate în plus în modulul `main`. Astfel, modulul `main` poate afla dacă trebuie să citească senzorii externi folosind următoarea funcție:

```
bool shouldReadSensors() {
  // Citirea se face cand variabila are valoarea 1 altfel statia ar afisa
  // mult timp date incorecte atunci cand este pornita
  if (readSensors == 1)
    return true;
  if (readSensors == SENSOR_READ_TIME - 1)
    readSensors = 0;
  return false;
}
```

Station

Fișiere: station.h și station.cpp
Biblioteci: Adafruit_BMP280, DHT sensor library

Realizează comunicarea cu senzorul de temperatură și presiune BMP280 și cu senzorul de umiditate DHT11. Aici sunt conținute o serie de array-uri globale pentru a păstra extremele pe ore, precum și setările stației: unitatea de măsură pentru temperatură, unitatea de măsură pentru presiune și o corecție pentru presiune care este adunată atunci când se face afișarea presiunii. Funcțiile implementate aici citesc senzorii pentru a afla parametrii mediului extern, actualizează array-urile globale, extrag din aceste array-uri parametrii minimi sau maximi pe ore sau pe intreaga zi și modifică setările stației.

Deoarece se memorează foarte multe date, pentru a reduce dimensiunea ocupată de acestea, valorile citite din mediu sunt păstrate în variabile de tip int16_t în loc de float. Pentru a putea păstra variabilele în acest mod a fost nevoie să modific bibliotecile DHT și Adafruit_BMP280 ca să întoarcă tipuri întregi în loc de float-uri. Astfel, temperatura păstrată într-o variabilă este temperatura reală in grade celsius cu două zecimale înmulțită apoi cu 100, umiditatea este cea reală înmulțită cu 10, iar presiunea este cea reală în pascali din care se scade 90000. Se scade 90000 din presiunea citită de la senzor, pentru că aceasta este valoarea minimă pe care senzorul o poate citi, conform datasheet-ului, și pentru că aceasta modificare permite stocarea presiunii într-o variabilă de tip int16_t în loc de una int32_t. Mai mult, modificarea unităților de măsură nu afectează semnificația acestor variabile, ci doar modul în care se afișează.

Astfel, aceste modificări permit afișarea temperaturii, de exemplu, folosind o implementare proprie fără a mai fi nevoie de anumite flag-uri de compilare pentru a include o variantă mai completă a funcției snprintf, care ar trebui să poată afișa și float-uri [1) 2)].

```
void printTemperature(char *buffer, temperature_t temperature) {
  int8_t significant;
  int8_t decimal;
```

```
  // Intai se face conversia
  if (getTemperatureUnit() == TEMPERATURE_F)
    temperature = (temperature * 9) / 5 + 3200;

  significant = temperature / 100;
  decimal = temperature % 100;

  // Se rotunjeste la prima zecimala
  if (decimal < 0)
    decimal = -decimal;
  if (decimal % 10 >= 5)
    decimal += 10;

  if (decimal >= 100) {
    if (significant < 0)
      significant--;
    else
      significant++;

    decimal = 0;
  }
  decimal = decimal / 10;

  // Unitatea de masura se afiseaza impreuna cu temperatura
  const char* temperatureString =
    getTemperatureUnit() == TEMPERATURE_C ? PSTR("%3hd.%.1hdC") : PSTR(
"%3hd.%.1hdF");
  snprintf_P(buffer, 7, temperatureString, significant, decimal);
}
```

Similar, se aplică aceeași procedură și pentru presiune. În schimb, umiditatea este afișată fără zecimale datorită lipsei de precizie a senzorului DHT11.

Button

Fisiere: button.h și button.cpp

Oferă funcțiile necesare pentru a semnala dacă un buton a fost apăsat și pentru a afla care din butoane este apăsat la un moment dat. Întreruperea folosită pentru butoane este următoarea:

```
ISR(PCINT2_vect) {
  disableButtons(); // PCMSK2 = 0
  checkButton = true;
}
```

Pentru a păstra întreruperea cât mai scurtă, aici doar se semnalează dacă un buton a fost apăsat,

stația poate detecta doar o apăsare de buton la un moment dat, prioritatea butoanelor fiind determinată în funcția care citește starea acestora:

```
button_t getButton() {
  button_t button = BUTTON_NONE;

  if (!(PIND & (1 << DDD3))) {
    button = BUTTON_PREV;
  } else if (!(PIND & (1 << DDD4))) {
    button = BUTTON_NEXT;
  } else if (!(PIND & (1 << DDD5))) {
    button = BUTTON_CANCEL;
  } else if (!(PIND & (1 << DDD6))) {
    button = BUTTON_OK;
  } else if (!(PIND & (1 << DDD7))) {
    button = BUTTON_EXTRA;
  }

  return button;
}
```

Pentru a face debouncing, modulul `button` foloseste exclusiv timer-ul 1. Timer-ul este configurat în functia de initializare a modulului pentru a genera o întrerupere odată la 200 de ms. Cum s-a văzut în întreruperea de mai sus, odată ce se apasă un buton, se dezactivează complet celelalte butoane până când sunt reactivate explicit. Modulul main va apela funcția blockButtons() pentru a dezactiva temporar butoanele, întreruperea timer-ului 1 le va reactiva după ce expiră timpul alocat.

```
void blockButtons() {
  disableButtons(); // PCMSK2 = BUTTON_MASK
  enableButtonTimer(); // reseteaza contorul timer-ului si activeaza
prescaler-ul
}

ISR(TIMER1_COMPA_vect) {
  disableButtonTimer();
  enableButtons();
  // Se semnaleaza buclei principale ca butoanele trebuie verificate din nou
  // Astfel, se realizeaza o repetare daca butonul este tinut apasat
  checkButton = true;
}
```

Remote

Fișiere: remote.h și remote.cpp
Biblioteci: IRremote

Modulul remote conține funcțiile necesare pentru a stoca și a retransmite un cod infraroșu primit de la

o telecomandă. Biblioteca folosită pentru acestea are nevoie de timer-ul 2 pentru receptie și timer-ul 0 pentru transmitere. Din moment ce e de asteptat ca recepția și trasmiterea să nu se facă foarte des, alimentarea pentru cele două timere este oprită odată ce nu mai este nevoie de acestea.

Deoarece această bibliotecă este foarte complexă, au fost folosite mai multe define-uri care configurează biblioteca înspre reducerea spațiului ocupat de aceasta:

```
#define RAW_BUFFER_LENGTH 100
#define IR_SEND_PIN 11
#define EXCLUDE_EXOTIC_PROTOCOLS
#define NO_LED_FEEDBACK_CODE
#define NO_DECODER
```

Cea mai importantă configurare de mai sus este NO_DECODER, care exclude codul care ar fi folosit pentru decodificarea semnalelor primite. Din moment ce stația doar retransmite un cod primit, nu este nevoie să-l și decodifice. Astfel, alarma setată de utilizator poate să reproducă orice cod primit, atâta timp cât există spațiu suficient ca să-l memoreze.

Biblioteca include deja două obiecte pentru transmitere si recepție și, pentru a economisi spațiu, le-am folosit pe acestea. Opțiunea IR_SEND_PIN reduce mai mult codul pentru transmițător prin excluderea unui constructor care nu ar fi fost folosit.

Alarm

Fișiere: alarm.h și alarm.cpp

Oferă sub forma clasei AlarmManager o implementare pentru o alarmă care poate să sune în anumite zile ale săptămânii și care poate să revină după un număr fix de minute. Astfel, alarma are echivalentul unei funcții de snooze, care, în realitate, va fi folosită mai departe în modulul `menu` pentru a putea opri un dispozitiv pornit.

Metoda principală a acestei clase, care trebuie apelată pentru a verifica dacă alarma sună, este următoarea:

```
bool AlarmManager::check() {
  // Pentru a evita situatii in care metoda este apelata
  // de mai multe ori in acelasi minut, posibil in aceasi
  // secunda, se impune ca metoda sa intoarca true
  // cel mult odata pe minut
  if (getMinutes() == lastMinuteChecked)
    return false;
  lastMinuteChecked = getMinutes();

  if (!active)
    return false;

  if (powerOffTimerActive) {
```

```
    powerOffTimerCounter--;
    if (powerOffTimerCounter <= 0) {
      powerOffTimerActive = false;
      return true;
    }
  }

  // getDate() si getMinutes() sunt implementate in modulul time
  const DateTime& currentDate = getDate();

  if (dayEnabled(currentDate.dayOfTheWeek()) && currentDate.hour() == hour
&& getMinutes() == minute) {
    // Daca timer-ul de oprire e activat, atunci se incepe numaratoarea
    // Daca valoarea timer-ului este 0, atunci se considera ca este
dezactivat
    if (powerOffTimerValue > 0) {
      powerOffTimerCounter = powerOffTimerValue;
      powerOffTimerActive = true;
    }
    return true;
  }

  return false;
}
```

Metoda de mai sus trebuie apelată o dată pe minut, altfel este posibil să se piardă alarma. Odată ce sună prima oară, variabila powerOffTimerActive este pusă pe true pentru a începe numărătoarea inversă a timer-ului de oprire.

Mai mult, pentru a evita un comportament impredictibil al alarmei atunci când se modifică data sau se dezactivează alarma, timer-ul de oprire pentru dispozitiv este dezactivat:

```
void AlarmManager::disable() {
  active = false;
  powerOffTimerActive = false;
}

void AlarmManager::setTime(const uint8_t newHour, const uint8_t newMinute) {
  powerOffTimerActive = false;
  hour = newHour;
  minute = newMinute;
}
```

Menu

Fișiere: menu.h și menu.cpp
Biblioteci: LiquidCrystal_I2C

Conține implementările pentru toate meniurile afișate utilizatorului, precum și instanțe ale acestor meniuri care vor fi folosite indirect de modulul principal prin intermediul unor funcții care apelează metodele meniului care este afișat pe ecran și printr-o funcție care modifică meniul care trebuie afișat. Tot aici este declarat și obiectul pentru interacțiunea cu display-ul. Meniurile interacționează cu aproape toate celelalte module, dar, la bază, acestea moștenesc următoarea interfață:

```
class Menu {
  public:
    // Functia care se apeleaza atunci cand un meniu urmeaza
    // sa fie afisat pe ecran
    virtual void load();
    // Se apeleaza dupa ce meniul a fost afisat prima oara
    // Conventia este ca meniul sa afiseze pe ecran
    // doar componentele care nu se actualizeaza frecvent
    virtual void draw() = 0;

    // Metoda care este apelata o data pe secunda, dupa celelalte
    // doua metode de mai sus, aici meniurile trebuie sa afiseze
    // doar componentele care se pot schimba
    virtual void tick() = 0;

    virtual void button(button_t button);
};
```

În general, un meniu este o mașină de stări care ia decizii în funcție de timp sau de butonul care a fost apăsat. Meniurile implementate sunt: MainMenu, MinMaxMenu, AlarmMenu, SettingsMenu, GraphMenu și ErrorMenu.

Dintre acestea, ErrorMenu este tratat diferit si va fi folosit de modulul `main` pentru a afișa o eroare în caz că este nevoie. GraphMenu folosește o serie de caractere speciale care sunt încărcate în memoria display-ului atunci când se apelează metoda de inițializare pentru display. GraphMenu afișează graficul pe baza presiunii maxime și minime înregistrate în ultimele 24 de ore, fără să existe o unitate de măsură, astfel se va putea observa modificarea presiunii în timp.

AlarmMenu incorporează un AlarmManager și folosește modulul `remote` pentru a recepționa un cod care este memorat tot în acest meniu. De asemenea, AlarmMenu mai are o metodă care este folosită pentru a retransmite codul stocat în caz că alarma setată este activată și sună la momentul în care această metodă este apelată. Modulul `main` nu are cunoștiință că ar exista mai multe alarme, așa că apelează o funcție a acestui modul care verifică alarmele ambelor meniuri.

Tot în acest modul, funcția switchCurrentMenu(), care încarcă meniul următor, folosește o serie de variabile globale pentru a reveni întotdeauna la meniul principal după un interval de timp în care nu au mai fost apăsate butoane. Astfel, stația nu va afișa mult timp meniuri care necesită mai multe calcule sau comunicații cu alte componente. În funcție de butonul apăsat, un meniu poate să semnaleze că trebuie afișat alt meniu. Odată ce plăcuța se trezește în urma apăsării butonului, modulul `main` va apela switchCurrentMenu() pentru a realiza înlocuirea. De asemenea, după ce se revine la meniul principal, backlight-ul lcd-ului va rămâne aprins doar pentru o perioadă scurtă de timp, pentru a economisi energie.

Main

Fișiere: weather_station.ino

Modulul principal care reunește celelalte module. Pe lângă funcțiile arduino setup() și loop(), aici mai sunt implementate alte funcții care nu puteau să aparțină altui modul, de exemplu disableOther(), care dezactivează anumite componente ale microcontroller-ului care nu sunt folosite sau powerDownUnused() care taie alimentarea din părți ale microcontroller-ului care nu sunt folosite.

Dintre aceste funcții, funcția sleep() pune microcontroller-ul într-o stare de sleep, alegând starea de sleep în funcție de activitatea microcontroller-ului, mai exact va pune microcontroller-ul în idle dacă urmează să se reactiveze butoanele sau dacă timer-ul 2 este activat, adică meniul AlarmMenu așteaptă recepționarea unui cod infraroșu. Dacă nu există activitate, sleep() va pune plăcuța în modul power down.

Sumar, pentru a inițializa complet toate modulele și a pune stația în funcțiune, metoda setup() va face următoarele:

- inițializează display-ul ca să se poată afișa erori în caz că alte module nu se pot inițializa
- inițializează senzorii apelând initStation() din modulul `station` și apoi apelează metoda initEnvironment() din același modul pentru a inițializa valorile minime și maxime înregistrate în ultimele 24 de ore
- inițializează ceasul extern printr-un apel la metoda initTime() din modulul `time`
- activează întreruperea pentru ceas și comunică ceasului să activeze local alarma care va declanșa această întrerupere o dată pe secundă
- dezactivează și taie alimetarea componentelor nefolosite ale microcontroller-ului
- activează watchdog-ul
- inițializează modulul `button` cu metoda setupButtons() și modulul `menu` cu metoda initMenus()
- afișează pe ecran meniul principal

Odată ce stația este inițializată, în buclă se vor face următoarele:

1. dacă întreruperea externă a fost declanșată, deci a trecut o secundă, atunci:
   1. se reseteaza watchdog-ul
   2. se notifică ceasul extern să reseteze întreruperea
   3. în caz că a trecut o oră atunci se desenează din nou meniul curent
   4. dacă a trecut suficient timp se de la ultima citire a senzorilor atunci se recitesc și se actualizează și extremele înregistrate
   5. se verifică alarmele
   6. se apelează funcția tick() a meniului curent, actualizându-se astfel meniul fără a se reface o desenare completă
2. dacă a fost apăsat un buton
   1. se citește starea butoanelor, aflându-se ce buton este apăsat
   2. se trimite butonul mai departe meniului care este pe ecran, ca acesta să-și poată modifica starea
   3. se blochează temporar butoanele
3. dacă meniul curent a semnalat că trebuie înlocuit cu alt meniu, atunci se încarcă meniul ales
4. se pune plăcuța în sleep doar dacă nu a venit întreruperea de la ceasul extern sau dacă nu a fost apăsat între timp un buton

Pentru funcția loop() am ales să realizez toate operațiile în ordinea de mai sus pentru a prioritiza citirea senzorilor, alarmele și, în special, păstrarea sincronizării cu ceasul extern. Deși există posibilitatea ca anumite apăsări de butoane să fie pierdute dacă durează prea mult ca stația să ajungă la citirea stării butoanelor, păstrând această ordine a operațiilor se asigură că stația nu se va desincroniza, nu va rata citiri ale senzorilor și nici nu va rămâne blocată. Tot pentru a păstra sincronizarea, nu se intră în sleep dacă a ceasul extern a declanșat întreruperea altfel se poate ajunge la situații în care stația se blochează dacă utilizatorul apasă un buton în același timp cu rularea întreruperii ceasului extern.

Rezultate Obţinute

Pentru a desena graficul, valorile pentru presiune au fost inițializate în prealabil folosind un define configurabil în cod.

Concluzii

În urma acestui proiect am învățat multe despre ecosistemul arduino si despre programarea microcontroller-elor.

Astfel, pot spune că am tras următoarele concluzii:

- organizarea e importantă: inițial am aveam tot codul pus într-un număr mic de fișiere fără ca acesta să fie modularizat prea bine, ceea ce a creat foarte multe probleme atunci când am vrut să adaug mai multe funcționalități la stație. După ce am reorganizat codul în structura actuală, implementarea meniurilor s-a făcut destul de ușor
- float-urile sunt de evitat: original, foloseam float-uri pentru a stoca un număr mare de valori citite din mediu. Nu numai că ocupa foarte multă memorie ram, dar și multă memorie program pentru că era nevoie de o implementare software care să facă operațiile cu acestea. Sunt convins că eliminarea float-urilor a redus nu numai spațiul necesar dar și consumul de energie al stației pentru că s-a redus complexitatea calculelor care se fac pentru afișări
- citește manualul (din nou): deoarece umblu la timere folosind registre și folosesc întreruperi definite manual, asta a făcut foarte greu să găsesc o bibliotecă pentru recepții și transmisii pe infraroșu care să accepte codul pe care îl aveam deja scris. Biblioteca IRremote este scrisă foarte bine și tratează aproape toate situațiile imaginabile, dar tot din cauza asta ocupă foarte mult spațiu (în jur de 8-11KB, și 400-500B de ram). Crezând că nu pot s-o folosesc, am trecut printr-o serie de alte biblioteci mai prost scrise care funcționau doar parțial (doar recepție sau transmisie, dar nu amândouă) sau de cele mai multe ori deloc. În final am revenit la biblioteca IRremote pentru că am găsit în codul sursă acele define-uri care pot exclude anumite componente, reducând dimensiunea ocupată la aproximativ 3KB si ram-ul utilizat la o valoare care să nu facă stiva să dea peste heap
- dacă vrei ceva făcut bine...: majoritatea bibliotecilor scrise pentru arduino sunt de cele mai multe ori prea simple, uneori nu interacționează prea grozav unele cu altele și aproape sigur fac presupuneri “rezonabile”, cum că funcția millis() ar fi disponibilă tot timpul, doar că în cazul meu nu este.
- fiecare octet contează: nu am dat deloc atenție utilizării ram până când am inclus biblioteca IRremote și am aflat că mi-ar trebui 2200 de octeti de memorie ram ca să încapă tot (asta înainte să

descopăr define-urile pentru optimizări). După ce am început să fac alte optimizări și să mai reduc din variabile, am ajuns la 1600-1700 de octeți folosiți, dar nu funcționa nimic pentru că stiva există în continuare și, din nefericire, costul modularizării este că nu toate funcțiile pot fi făcute inline. În final, am descoperit că orice sub 1500 de octeți funcționează fără probleme în cazul meu

Download

Bibliotecile modificate: statie_meteo_biblioteci.zip
Codul sursă: statie_meteo_cod.zip

Jurnal

28.05 - Adăugat demo funcționare, corectat greșeli din text, actualizat codul
27.05 - Adăugat milestone-ul software, actualizat poza pentru schema electrică și poza pentru hardware
21.05 - Adăugat un modul software nou, actualizat diagramele vechi și adăugat o poză nouă
20.05 - Actualizat pagina de proiect, adăugat milestone-ul hardware
02.05 - Creat pagina de proiect și adăugat documentația inițială

Bibliografie/Resurse

Biblioteci 3rd-party:

- RTClib: https://github.com/adafruit/RTClib
- Adafruit_BMP280: https://github.com/adafruit/Adafruit_BMP280_Library
- DHT sensor library: https://github.com/adafruit/DHT-sensor-library
- IRremote: https://github.com/Arduino-IRremote/Arduino-IRremote
- LiquidCrystal_I2C: https://github.com/johnrickman/LiquidCrystal_I2C

Resurse folosite:

- codul sursă și exemplele incluse în bibliotecile folosite, în special cele din IRremote
- datasheet-ul ATmega328p pentru configurările cu registre: ATmega328P
- datasheet-ul DS3231M: DS3231M
- datasheet-ul BMP280: BMP280
- datasheet-ul LCD2004A: LCD2004A

Export to PDF

[1] https://forum.arduino.cc/t/solved-why-cant-i-print-a-float-value-with-sprintf/367971
[2] https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html#gaa3b98c0d17b35642c0f3e464

9092b9f1