Karta projektu zaliczeniowego

Systemy mikroprocesorowe - 2018

Temat projektu: **Zegarek z funkcją pomiaru wilgotności i temperatury**

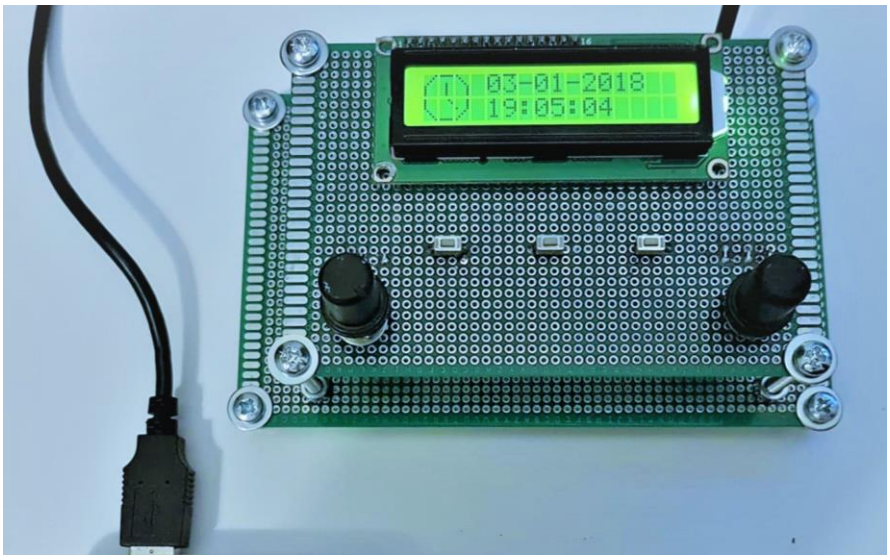
Imię i nazwisko: **Kamil Andrzejewski**

Politechnika Poznańska

kierunek: **AiR**, grupa: **A3**, nr albumu: **127072**

1. Opis projektu

Realizacja projektu dotyczy zbudowania zegarka z dodatkowymi funkcjami pomiaru temperatury i wilgotności względnej w pomieszczeniu. Układ oparty jest na mikrokontrolerze Atmega328P z wgranym bootloaderem Arduino. Dane wyświetlane są na wyświetlaczu LCD 2x16 opartym na kontrolerze HD44780 w kolorze zielonym. Za pomocą 2 potencjometrów można regulować kontrast wyświetlacza oraz jasność podświetlenia. Odmierzanie czasu realizowane jest za pomocą modułu RTC DS1307 z podtrzymaniem bateryjnym. Do pomiaru temperatury i wilgotności służy czujnik DHT11. Użytkownik ma do dyspozycji 3 przyciski za pomocą, których może przełączać informacje wyświetlane na ekranie LCD oraz do zmiany daty i godziny. Układ wymaga zasilania napięciem 5 [V] poprzez kabel USB. Projekt zawiera elementy z odzysku takie jak przyciski, potencjometry, rezystory, kabel USB.



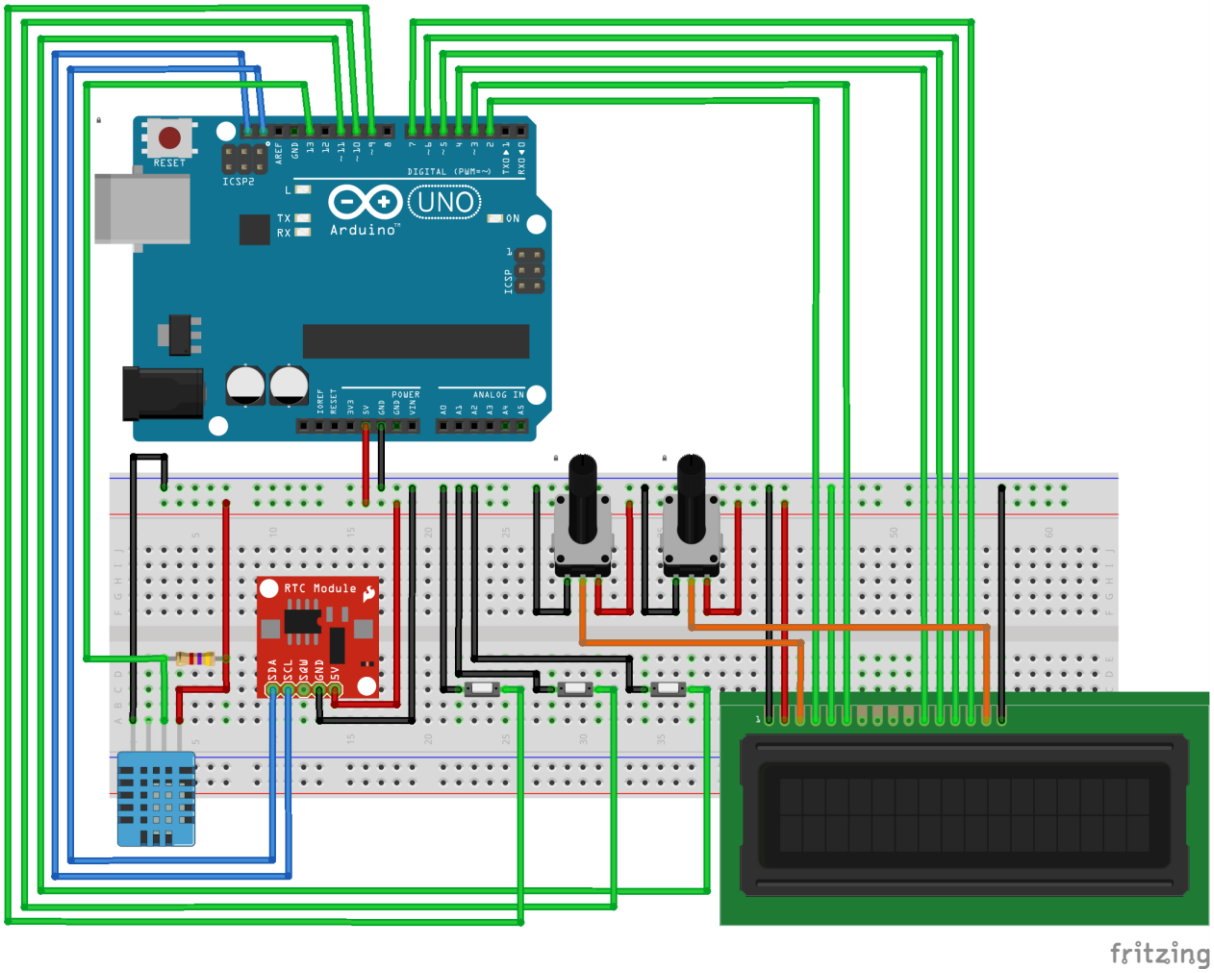
2. Budowa układu

Zastosowane elementy:

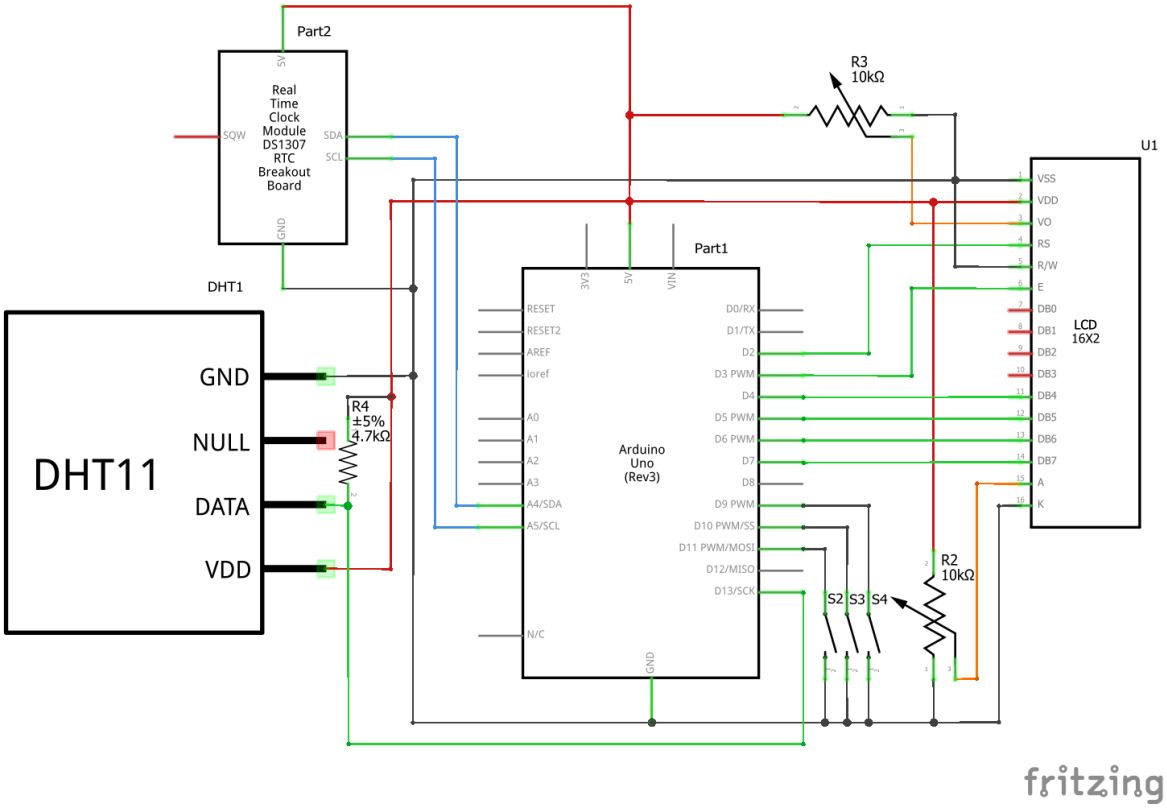
- 1 x mikrokontroler Atmega328P
- 1 x wyświetlacz LCD 16x2 oparty na kontrolerze HD44780
- 1 x moduł zegara RTC DS1307 z podtrzymaniem bateryjnym

- 1 x czujnik temperatury i wilgotności względnej DHT11
- 1 x rezonator kwarcowy 16 [MHz]
- 2 x kondensator ceramiczny 22 [pF]
- 2 x potencjometry 10 [kΩ] pozyskane ze starego radzieckiego telewizora.
- 3 x przyciski PTA-111 pozyskane z radia samochodowego.
- 3 x rezystory 4,7 [kΩ] pozyskane z latarki z paralizatorem.
- 1 x kabel z wtyczką USB.

Początkowo prototyp zbudowano wykorzystując sterowanie za pomocą Arduino Uno. Schematy układu wyglądały następująco:

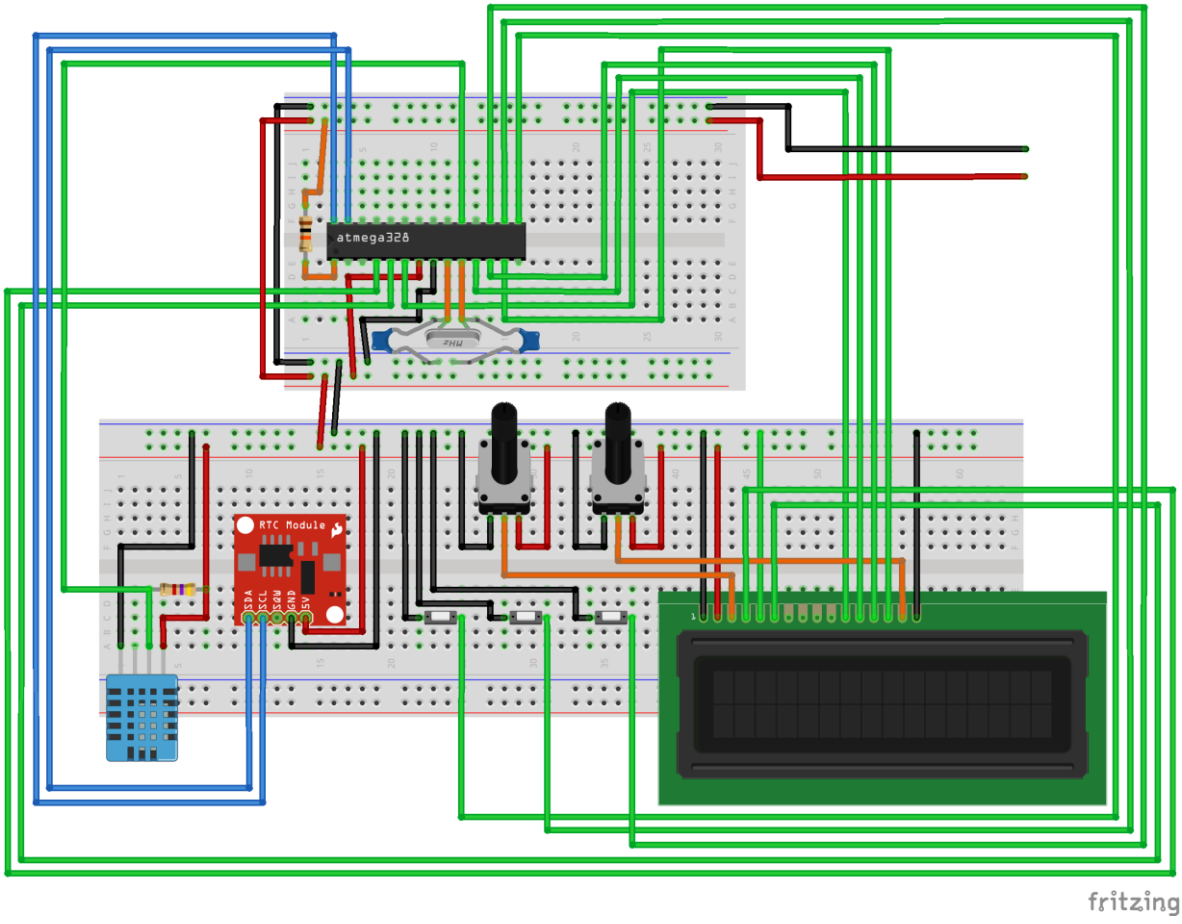


Rysunek 1 Schemat graficzny prototypu oparty o Arduino Uno

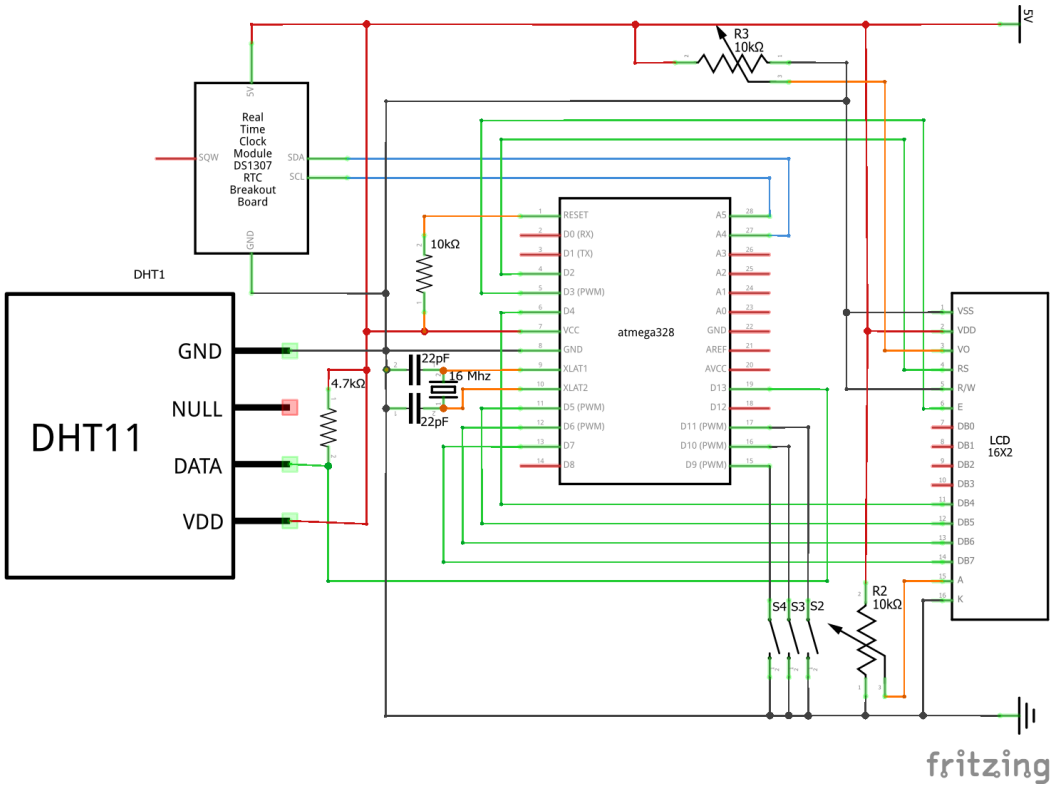


Rysunek 2 Schemat elektryczny prototypu oparty o Arduino Uno

Po poprawnym uruchomieniu programu, zbudowano kolejny prototyp. Tym razem oparty na mikrokontrolerze Atmega328P. Schematy prototypu wyglądają następująco:



Rysunek 3 Schemat graficzny prototypu opartego o mikrokontroler Atmega328P



Rysunek 4 Schemat elektryczny prototypu opartego o mikrokontroler Atmega328P

Ostatecznie prototyp został zbudowany na 2 płytkach uniwersalnych skręconych śrubami, tworzących sztywną i samodzielną konstrukcje. Do połączeń wykorzystano pojedyncze żyły przewodu UTP CAT5.

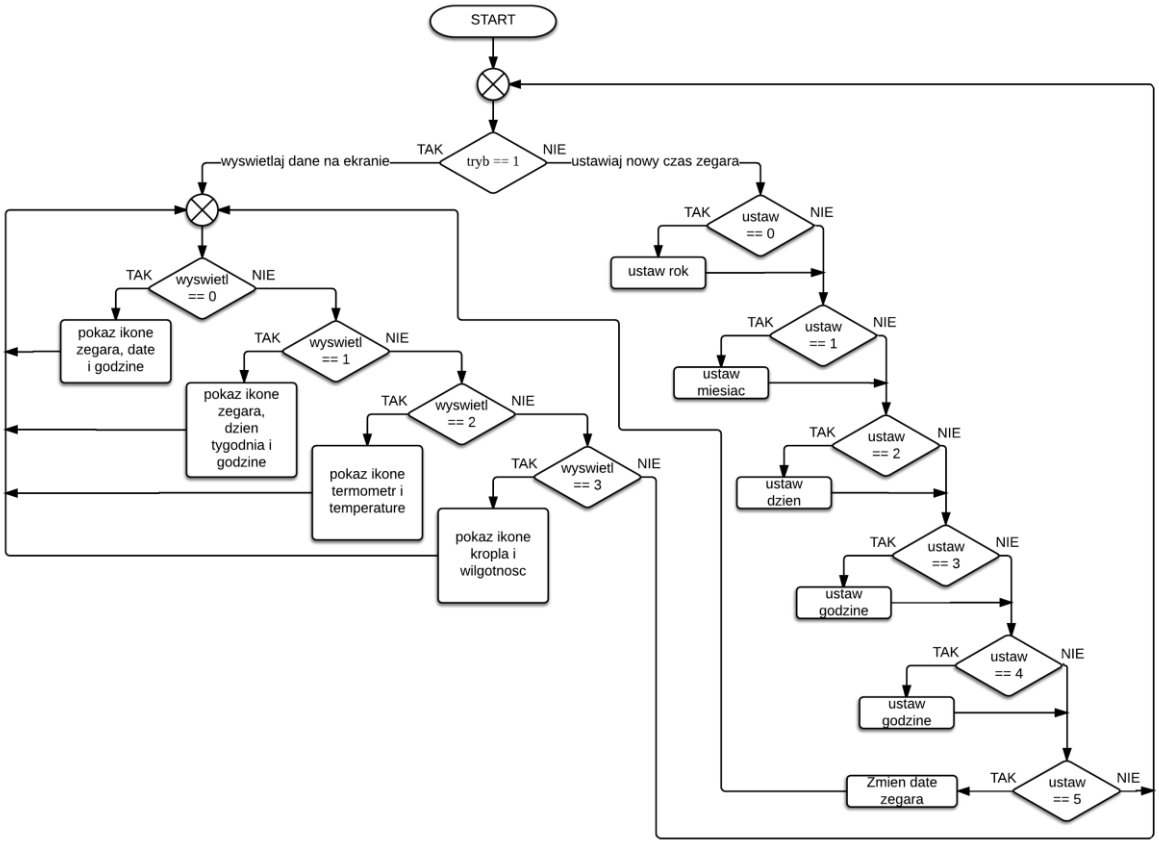
3. Elementy oprogramowania

Biblioteki wykorzystane w programie:

- Wire.h – umożliwia komunikację za pomocą magistrali I^2C.
- LiquidCrystal.h – obsługa wyświetlacza LCD.
- Timers.h – umożliwia wykorzystywanie w programie timerów.
- DS1307.h – biblioteka obsługująca moduł RTC.
- DHT.h – obsługa czujnika temperatury i ciśnienia.

Program sterujący układem posiada 2 zasadnicze tryby. Tryb wyświetlania danych na ekranie, podczas którego użytkownik ma możliwość za pomocą przycisków wyświetlać różne dane na ekranie, w tym aktualną datę i godzinę, dzień tygodnia oraz temperaturę i wilgotność względną w pomieszczeniu. Tryb programowania, który pozwala ustawić nam ustawić nową datę i godzinę urządzenia. Po ustawieniu daty i godziny oprogramowanie automatycznie przełącza urządzenie w tryb wyświetlania.

Schemat blokowy oprogramowania:



Rysunek 5 Schemat blokowy działania oprogramowania układu

Funkcje dodatkowe:

W lewej części wyświetlacza wyświetlane są autorskie ikony utworzone z własnoręcznie utworzonych znaków ASCII. Z powodu ograniczeń kontrolera wyświetlacza, który potrafi zapisać w pamięci tylko 8 niestandardowych znaków ASCII, przy zmianie informacji wyświetlanej na ekranie należało także zapisać w pamięci inne znaki niestandardowe. Rozwiązano to za pomocą funkcji:

```
//--------------------------------------------------------------------------
//                TWORZENIE ZNAKÓW WŁASNYCH
//--------------------------------------------------------------------------
void tworzenie_znakow()
{
  if ((przelacz_lcd == 0)
      || (przelacz_lcd == 1))
  {
    lcd.createChar(1, gora_lewy);
    lcd.createChar(2, gora_srodek_czas);
    lcd.createChar(3, gora_prawy);
    lcd.createChar(4, dol_lewy);
    lcd.createChar(5, dol_srodek_czas);
    lcd.createChar(6, dol_prawy_czas);
  }
  if (przelacz_lcd == 2)
  {
    lcd.createChar(1, gora_lewy);
    lcd.createChar(2, gora_srodek_temperatura);
    lcd.createChar(3, gora_prawy);
    lcd.createChar(4, dol_lewy);
    lcd.createChar(5, dol_srodek_temperatura);
    lcd.createChar(6, dol_prawy_temperatura);
  }
  if (przelacz_lcd == 3)
  {
    lcd.createChar(1, gora_lewy_wilgotnosc);
    lcd.createChar(2, gora_srodek_wilgotnosc);
    lcd.createChar(3, gora_prawy_wilgotnosc);
    lcd.createChar(4, dol_lewy_wilgotnosc);
    lcd.createChar(5, dol_srodek_wilgotnosc);
    lcd.createChar(6, dol_prawy_wilgotnosc);
  }
}
```

W trybie programowania wyświetla nam się data i godzina, którą chcemy ustawić. Aby ułatwić interakcję z użytkownikiem, element daty, który jest aktualnie ustawiany, pulsuje naprzemiennie pokazując się i znikając. Pulsacja zostaje przerywana, gdy wciśniemy któryś z przycisków służących do ustawiania daty. Realizuje to funkcja:

```
//--------------------------------------------------------------------------
//                MRUGANIE USTAWIANEGO SKŁADNIKA DATY
//--------------------------------------------------------------------------
void mruganie_skladnika(byte ustawiany_skladnik, int skladnik)
{
  if ((ustawiany_skladnik  == ustaw)
      && (digitalRead(przycisk_plus) == HIGH)
      && (digitalRead(przycisk_minus) == HIGH))
  {
      if (mrugniecie)
      {
        if (skladnik < 100)
        {
          uzupelnianie_zer(skladnik);
        }
        lcd.print(skladnik);
        mrugniecie = !mrugniecie;
      }
      else
      {
        if (skladnik > 99)
        {
          lcd.print("    ");
        }
        else
        {
          lcd.print("  ");
        }
        mrugniecie = !mrugniecie;
      }
    }
    else
    {
      if (skladnik < 100)
      {
        uzupelnianie_zer(skladnik);
      }
      lcd.print(skladnik);
    }
}
```

Gdy użytkownik przejdzie do trybu programowania, jednak ostatecznie nie chce zmieniać daty wystarczy, że nie będzie używał żadnego z przycisków przez ok. 6 sekund, wtedy program automatycznie przejdzie z powrotem do trybu wyświetlania. Rozwiązanie to umożliwia funkcja:

```
//--------------------------------------------------------------------------
//                WYŁACZANIE TRYBU PROGRAMOWANIA PRZY BRAKU AKTYWNOŚCI
//--------------------------------------------------------------------------
void wylaczanie_programowania()
{
 //JEŻELI NIE WYKRYTO AKTYWNOŚCI NA ŻADNYM PRZYCISKU
  if ((digitalRead(przycisk_tryb) == HIGH)
      && (digitalRead(przycisk_plus) == HIGH)
      && (digitalRead(przycisk_minus) == HIGH))
  {
    licznik = licznik + 1; // LICZNIK = 20 ===> 6 sekund (0.3*20)
    if (licznik == 20)
    {
        przelacz_tryb = false;
        ustaw = 0;
        licznik = 0;
    }
  }
 //JEŻELI  WYKRYTO AKTYWNOŚCI NA JEDNYM Z PRZYCISKÓW
  if ((digitalRead(przycisk_tryb) == LOW)
      || (digitalRead(przycisk_plus) == LOW)
      || (digitalRead(przycisk_minus) == LOW))
  {
    licznik = 0;
  }
}
```

Jak wiadomo miesięcy jest 12, dni miesiąca maksymalnie 31, minut 60 itd. Należało więc zablokować możliwość przektoczenia przez użytkownika tych wartości maksymalnych, aby data została ustawona poprawnie. Jest to osiągnięte za pomocą funkcji:

```
//--------------------------------------------------------------------------
//                ZABEZPIECZENIE PRZED WYKROCZENIEM POZA ZAKRES
//--------------------------------------------------------------------------
int zakres_zmiennej(int argument, int zakres_dol, int zakres_gora)
{
  if (argument == zakres_gora + 1)
  {
    argument = zakres_dol;
  }
  if (argument == zakres_dol - 1)
  {
    argument = zakres_gora;
  }
  return argument;
}
```

Po przekroczeniu którejś z wartości granicznej, program przechodzi na początek lub koniec zakresu (w zależności od tego czy przekroczono granicę dolną czy górną).

Aby przyciski działały prawidłowo, wyeliminowano zjawisko bounceingu. Zrealizowano to za pomocą timerów. Po zmianie stanu na pinie odpowiedzialnym za komunikacje z danym przyciskiem, instrukcje, za które wykonanie odpowiada akcja przycisku, wykonywane są po upłynięciu czasu odliczanego przez timer.

Biblioteka DS1307.h służąca do obsługi zegara czasu rzeczywistego została edytowana, aby dni tygodnia były wyświetlane w języku polskim.

Pełen kod oprogramowania znajduje się pod adresem:

http://cpp.sh/4hc5

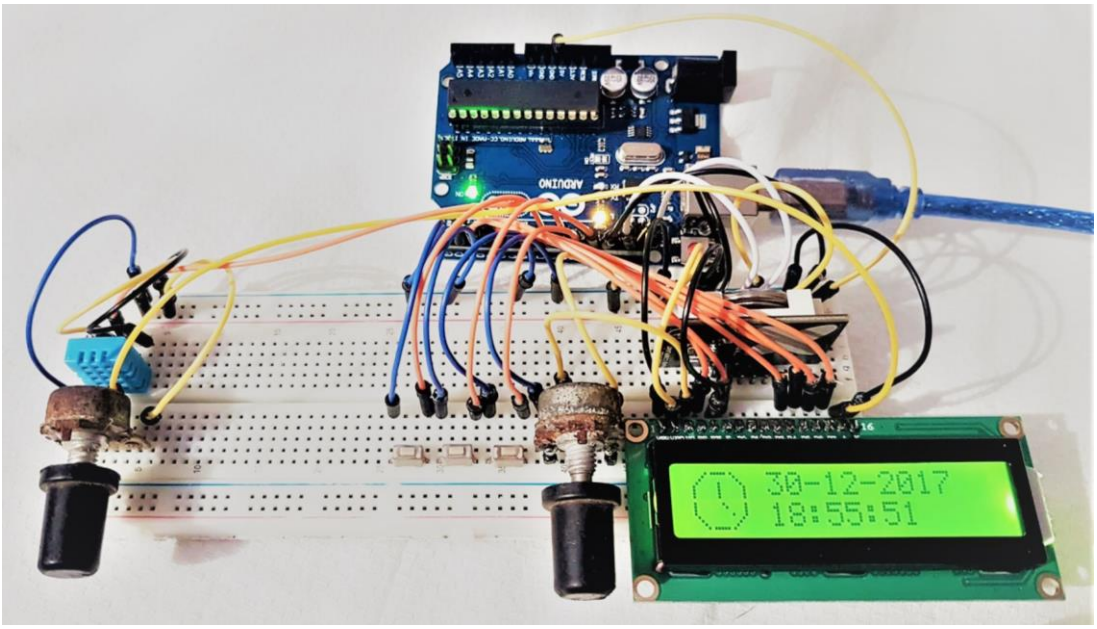
4. Wykorzystane narzędzia projektowe

Oprogramowanie zostało napisane w programie Arduino IDE 1.8.5. Służyło również do komunikowania się z Arduino UNO i wgrywania programu do mikrokontrolera.

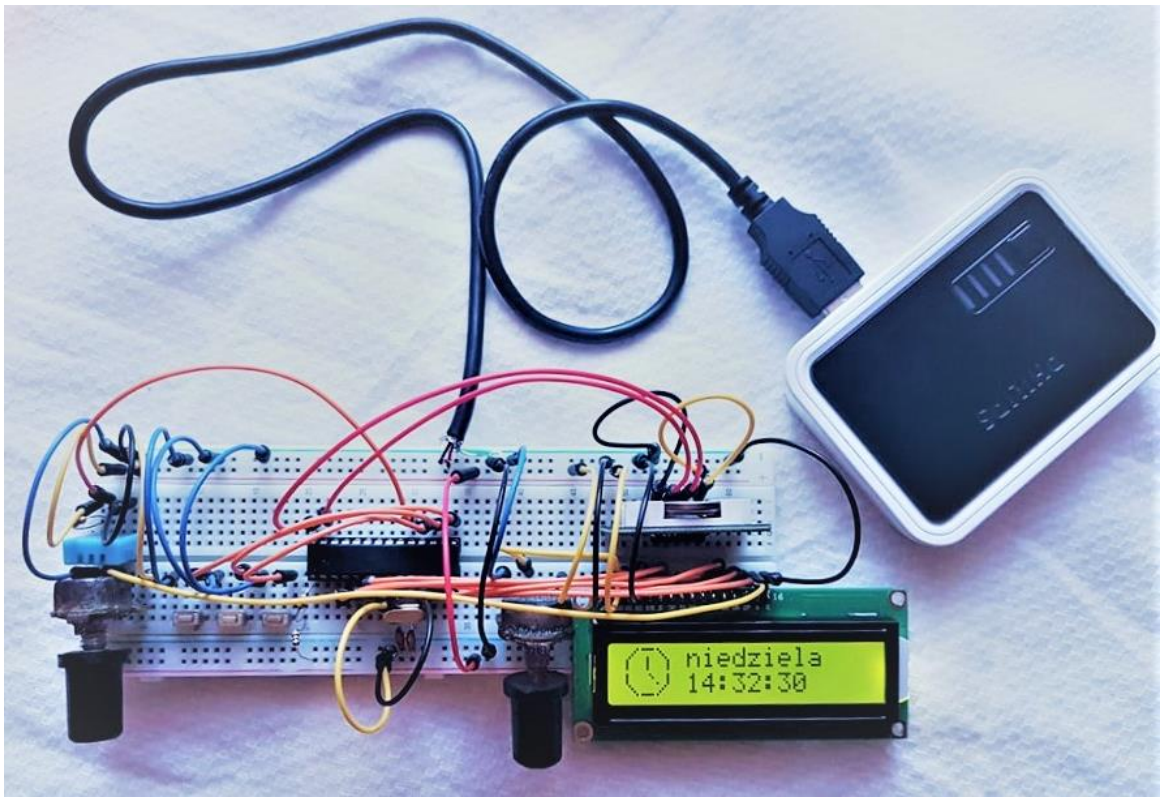
Do wypalania bootloadera Arduino na mikrokontrolerze Atmega328P również posłużył program Arduino IDE oraz programator Arduino ISP.

5. Weryfikacja poprawności działania układu

Pierwsze testy po napisaniu oprogramowania odbywały się z pomocą Arduino Uno oraz układu zmontowanego na płytce prototypowej. Układ działał bez żadnych zastrzeżeń. Zdjęcie prototypu:



Kolejnym etapem było pozbycie się Arduino UNO i usamodzielnienie się układu. W tym celu został wypalony bootloader Arduino na mikrokontrolerze Atmega328P, po czym stał się on gotowy do zaprogramowania. Po umieszczeniu mikrokontrolera z płytce prototypowej oraz wprowadzeniu kilku zmian w układzie, jego praca również pozostała bez żadnych zastrzeżeń. Od tej chwili układ zasilany był baterią 5V i nie wymagał Arduino UNO do działania. Zdjęcie prototypu:

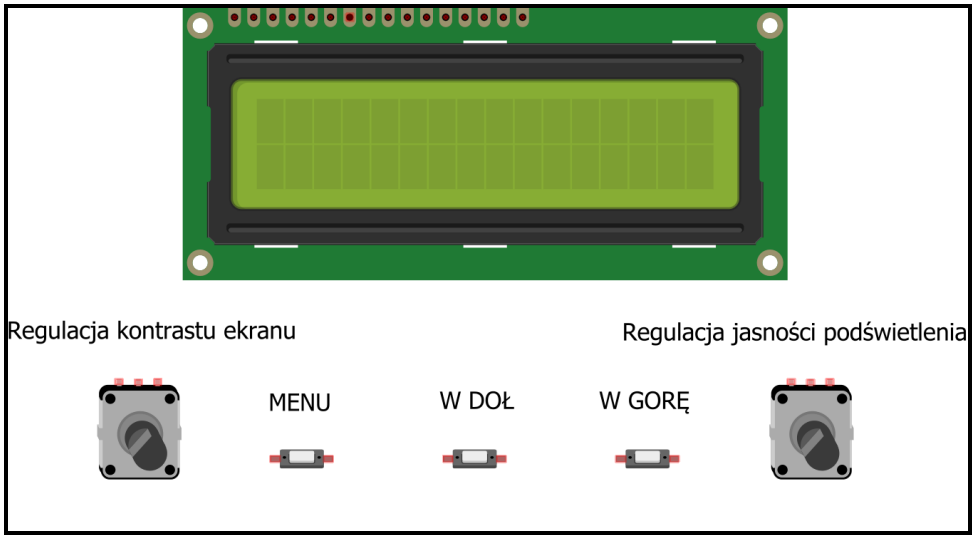


Następnie wszystkie elementy przeniesiono na 2 płytki uniwersalne i połączono przewodami z kabla UTP od dołu płytek, dzięki czemu żadne połączenia nie są widoczne. Gotowy projekt prezentuje się następująco:

Układ poddano testom sprawności. Po zaprogramowaniu modułu RTC pozostawiono urządzenie na tydzień bez zasilania, po tym czasie uruchomiono układ i sprawdzono czy godzina i data jest poprawna. Oczywiście nie mogło być inaczej i była poprawna. Kolejny test polegał na sprawdzeniu pracy urządzenia. Tym razem podłączono zegar do zasilania i pozostawiono na 72 godziny. Urządzenie działało prawidłowo w każdym momencie testu. Na koniec sprawdzano poprawność działania czujnika temperatury porównując jego wyniki z termometrem rtęciowym, w 3 różnych temperaturach. Margines błędu wahał się w granicach ±2°C. Z powodu braku czujnika wilgotności, nie zostało wykonane porównanie z czujnikiem wilgotności względnej zamontowanym w układzie.

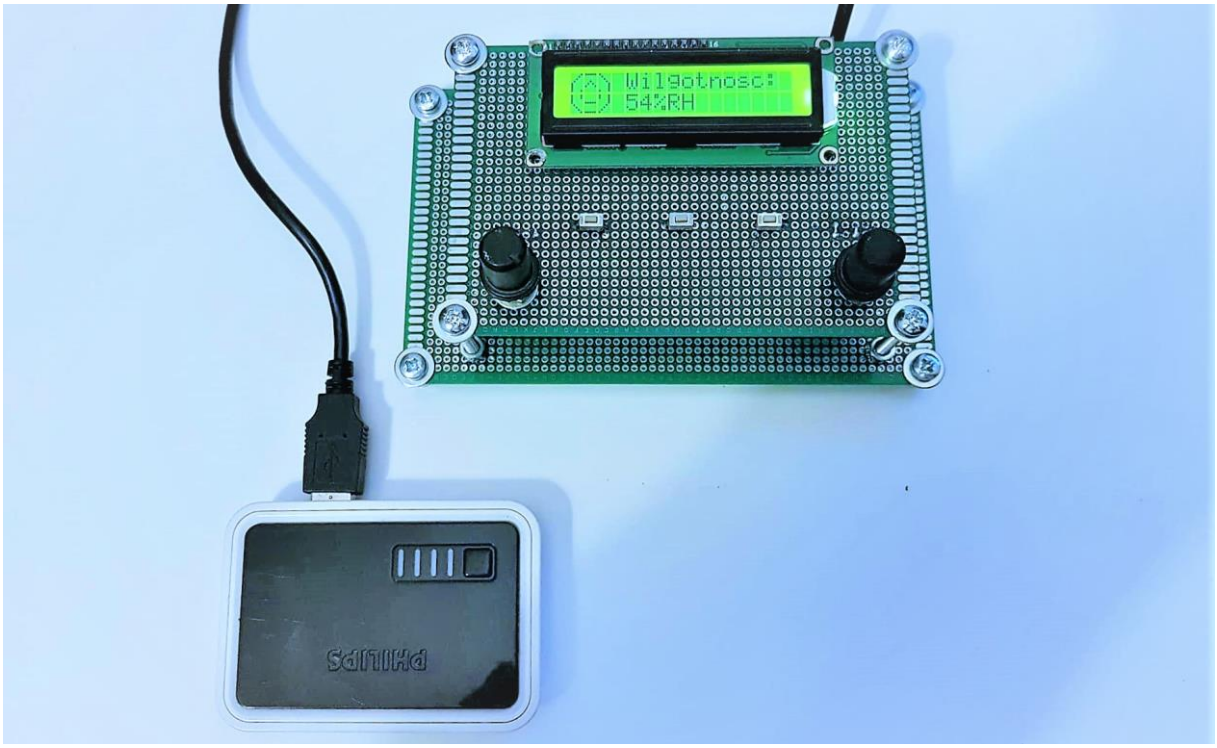
6. Obsługa układu

Górna płytka zawiera elementy do interakcji z użytkownikiem, czyli wyświetlacz, na którym wyświetlane są informacje oraz zestawem przycisków które zostały oznaczone na poniższym schemacie.

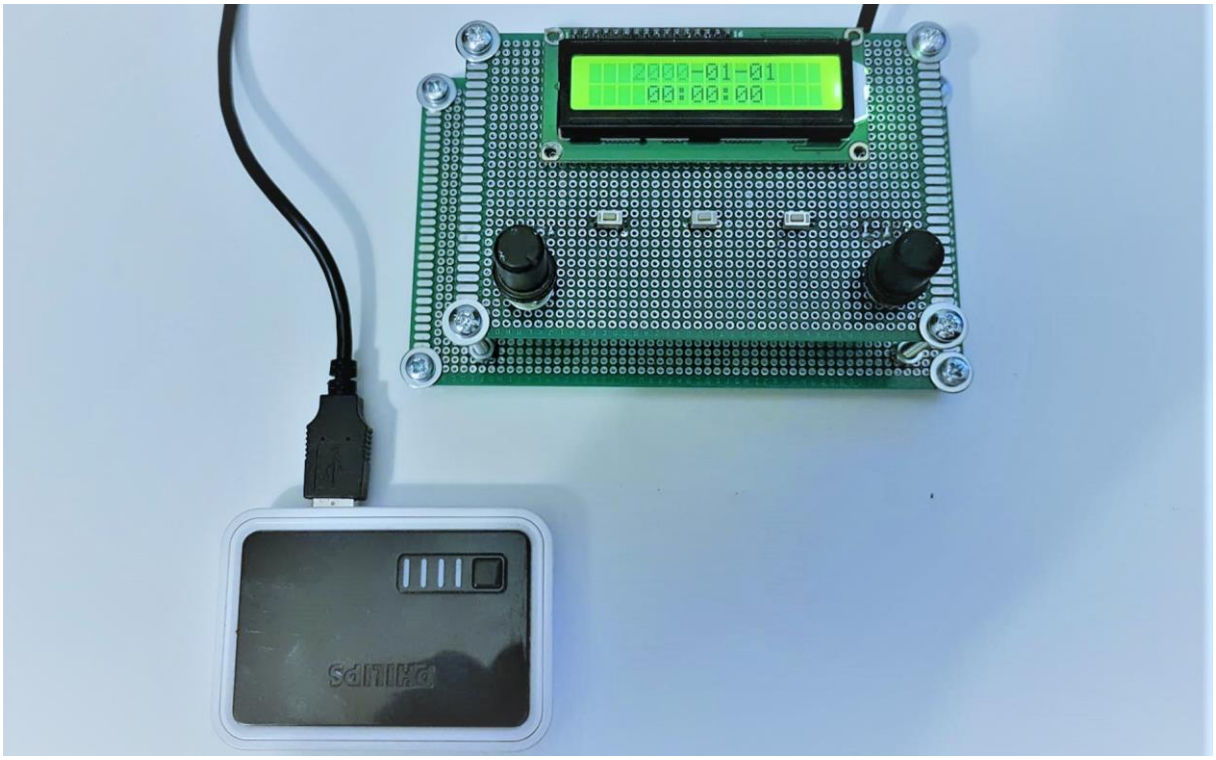


Rysunek 6 Widok górnej płytki projektu z oznaczeniami funkcji elementów.

Za pomocą przycisków W DOŁ, W GÓRĘ możemy wybierać informacje wyświetlane na ekranie. Zdjęcia prezentujące wyświetlane informacje:



Po wciśnięciu przycisku MENU przechodzimy w tryb programowania, czyli mamy możliwość ustawienia nowej daty i godziny. Przyciskami W GÓRĘ, W DÓŁ ustawiamy interesującą nas wartość, po czym potwierdzamy wybór klawiszem MENU. Po ustawienia ostatniego składnika daty i zatwierdzeniu wyboru nowa data zostanie wyświetlona na ekranie.



Dla wygody oceniającego, na górną płytkę układu nałożono, tabliczkę z oznaczeniami funkcjonalności klawiszy oraz potencjometrów.

7. Literatura

1. „Arduino dla początkujących. Podstawy i szkice”, Simon Monk, Wydawnictwo Helion
2. https://github.com/jarzebski/Arduino-DS1307 - Biblioteka modułu RTC DS1307 oraz opis jej funkcjonalności.
3. https://github.com/nettigo/Timers - Biblioteka Timers.h oraz opis jej funkcjonalności.
4. https://github.com/adafruit/DHT-sensor-library - Biblioteka czujnika DHT11.
5. http://www.jarzebski.pl...czujnik dht11 - wykorzystanie czujnika DHT11.
6. https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard - wypalanie bootloadera na mikrokontrolerze Atmega328P.