Politechnika Rzeszowska
Wydział Elektrotechniki i Informatyki
Katedra Podstaw Elektroniki

Podstawy Elektroniki

Stacja Pogodowa na bazie Raspberry Pi

Wykonał: Pazowski Marcin
158841
II EF-DI

Rzeszów 2018

Spis treści

Część I
Opis

Pomiary pogody dla niektórych osób są nieodzownym elementem dnia. Jeszcze przed wyjściem z łóżka ludzie oceniają jak powinni przygotować swój ubiór na resztę dnia. Pomiary warunków atmosferycznych nie ograniczają się jedynie do pomiarów zewnętrznych. Bardzo ważnym jest utrzymywanie odpowiednich parametrów wilgotności oraz temperatury na przykład w serwerowniach, chłodniach, a także w biurach.

1 Założenia projektowe

Projekt stacji pogodowej na bazie Raspberry Pi ma na celu przedstawienie możliwości implementacji jej podstawowych funkcjonalności. Stacja pogody jest wyposażona w podstawowe czujniki podłączone bezpośrednio do platformy komputerowej. Ważnym elementem funkcjonalności takowej stacji jest sposób dostarczania danych do użytkownika końcowego jak i do innych systemów niezależnych wykorzystujących przeprowadzane pomiary.

2 Projekt na tle dostępnych rozwiązań

Na rynku dostępnych jest wiele pojedynczych czujników, a także pełnych urządzeń, a także zintegrowanych systemów. W dużej mierze systemy te mają ograniczony dostęp do odczytów. Możliwe jest odczytanie ich z ekranu LCD, który nie sprawdza się gdy potrzebne są odczyty z miejsc oddalonych od siebie.
Zaletami przedstawionego projektu jest możliwość rozbudowany o dodatkowe funkcje jak np. podłączenie ogrzewania gdy temperatura spadnie poniżej określonej wartości.
Dodatkowo dane przechowywane są w bazie danych, która w łatwy sposób może być dostępna w globalnej sieci Internet. Dzięki temu dane mogę spływać z różnych stacji pogodowych. Oprócz tego pomiary można przedstawić w przyjaźniejszej formie dla użytkownika. Pomiary mogę być wyświetlane w formie wykresu, tabeli na stronie interpretować ale też w specjalnie napisanej aplikacji na smartphony.

Część II

Implementacja rozwiązania

Stacja pogody ma dwa najważniejsze punkty.

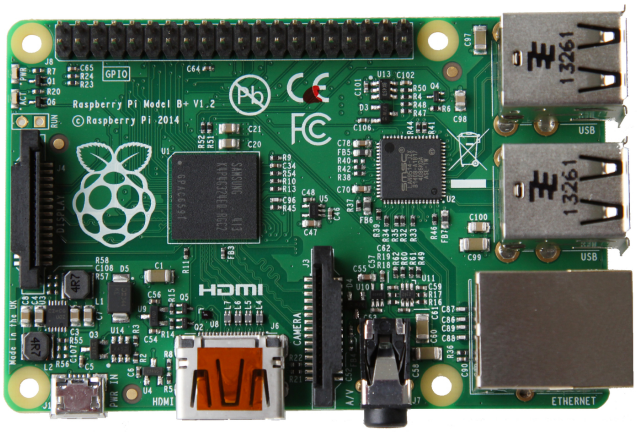
- Raspberry Pi: zbiera dane z podłączonych do niego czujników i wysyła je do serwera opisanego poniżej.
- Serwer bazodanowy oraz WWW: kolekcjonuje zebrane dane, a także udostępnia je końcowym użytkownikom przez protokół https.

3 Wykorzystane elementy

3.1 Raspberry Pi 2 Model B+ V1.2

Raspberry Pi to platforma komputerowa, która posiada podstawowe interfejsy komunikacyjne jak port Ethernet, USB, HDMI, a co najważniejsze GPIO oraz I2C, SPI, które wykorzystuje ten projekt.
System operacyjny instaluje się na zewnętrznym dysku. W tym wypadku na zewnętrznej karcie pamięci zainstalowany jest Raspbian bez nakładki graficznej, który jest oficjalnie wspieranym systemem dla tej platformy.

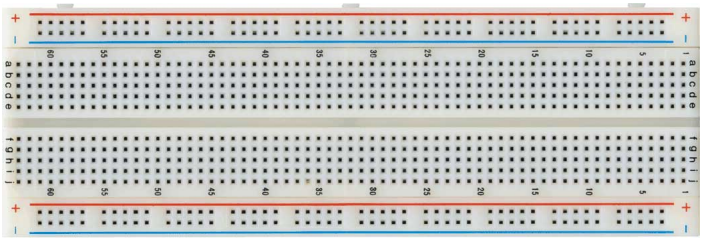
Pełna specyfikacja urządzenia dostępna jest na stronie producenta:
https://www.raspberrypi.org/products/raspberry-pi-1-model-b-plus/



Rysunek 1: Raspberry Pi 2 Model B+ V1.2

3.2 Płytka stykowa

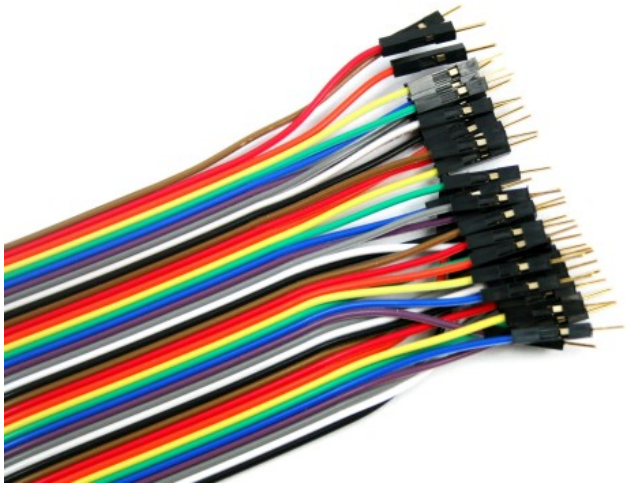
Płytka prototypowa, stykowa pozwala na bezproblemowe łączenie elementów elektronicznych w celu zbudowania prototypu.



Rysunek 2: Płytka prototypowa

3.3 Przewody połączeniowe - męsko-męski, żeńsko-męskie oraz żeńsko-żeńskie

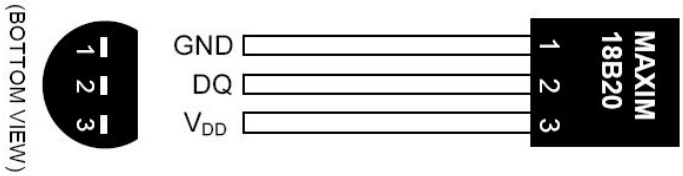
Przewody połączeniowe idealnie nadają się do projektowania prototypów układów elektronicznych. W projekcie zostaną wykorzystane trzy rodzaje połączeń, męsko-męski, żeńsko-męski oraz żeńsko-żeński.



Rysunek 3: Przewody połączeniowe

3.4 Czujnik temperatury DS18B20

Cyfrowy termometr DS18B20 o rozdzielczośći od 9 do 12 bitów mierzy temperaturę w zakresie od $-55°C$ do $+125°C$ z dokładnością $\pm 0.5°C$. Zaletami czujnika jest możliwość czerpania energii z linii danych, komunikacji po jednej linii oraz unikatowy 64-bitowy serial. Dzięki temu na jednej linii można podłączyć wiele czujników.



Rysunek 4: Schemat sensora DS18B20

3.5 Czujnik temperatury i wilgotności DHT11

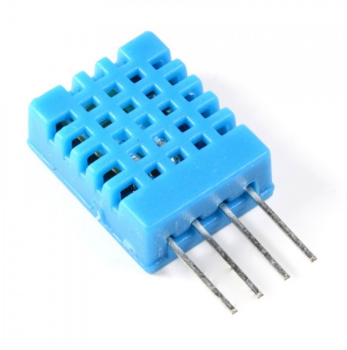
DHT11 jest to popularny czujnik firmy DFRobot, który mierzy temperaturę oraz wilgotność powietrza.

Specifikacja:

- Napięcie zasilania: $3.3V$ do $5.5V$
- Średni pobór prądu: $0,2mA$
- **Temperatura**
  - Zakres pomiarowy: do $-20°C$ do $+50°C$
  - Rozdzielczość: 8-bitów ($1°C$)
  - Dokładność: $1°C$
  - Czas odpowiedzi: $6 - 15s$ (typowo $10s$)
- **Wilgotność:**
  - Zakres pomiarowy: od 5% do 95% RH

– Rozdzielczość: 8-bitów ($\pm 1\%$ RH)
– Dokładność ± 4 RH (przy $25°C$)
– Zakres pomiarowy: $6 - 30s$

gdzie RH to wilgotność względna wyrażana w procentach. Jest to stosunek rzeczywistej wilgoci w powietrzu do maksymalnej jej ilości, którą może utrzymać powietrze w danej temperaturze.



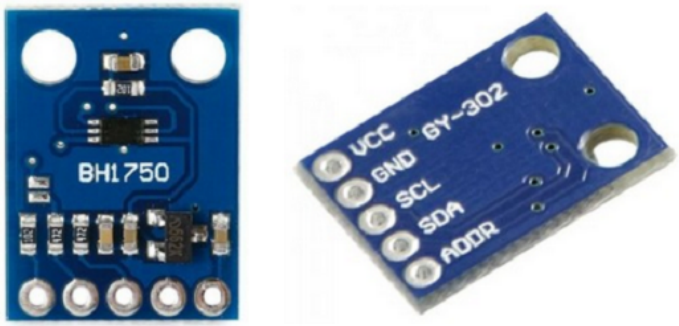
Rysunek 5: Sensor DHT11

3.6 Czujnik natężenia światła BH1750

BH1750 to czujnik przetwarzający natężenie światła o długości z zakresu od $320nm$ do $1050nm$ na mierzalną proporcjonalną częstotliwość. Sensor działa w zakresie $1 - 65535lx$ (luks) z rozdzielczością $1 - 4lx$. Komunikacja przebiega przez magistralę I^2C czyli przez linię danych (SDA) oraz zegarową (SCL).

Specyfikacja:

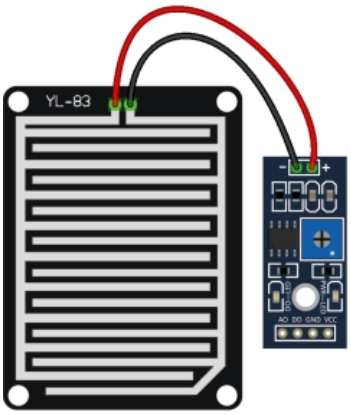
- Napięcie zasilania: $3V$ do $5V$
- Średni pobór prądu: $120uA$
- Interfejs komunikacyjny: $I2C$
- Wbudowane rezystory $4.7\,\mathrm{k}\Omega$ podciągające linie SCL i SDA magistrali I^2C
- Wbudowany regulator napięcia
- Zakres: $1 - 65535lx$
- Wymiary modułu: 21 x 16 x $3,3mm$



Rysunek 6: Sensor BH1750

3.7 Czujnik opadów deszczu YL-83

YL-83 służy do wykrywania opadów. Urządzenie składa się z dwóch części, detektora połączonego z sondą pomiarową. W tym projekcie zostanie użyte wyjście cyfrowego modułu, które po wykryciu opadów przechodzi ze stanu wysokiego w stan niski. Próg napięciowy (czułość) można regulować za pomocą potencjometru umieszczonego na płytce detektora.



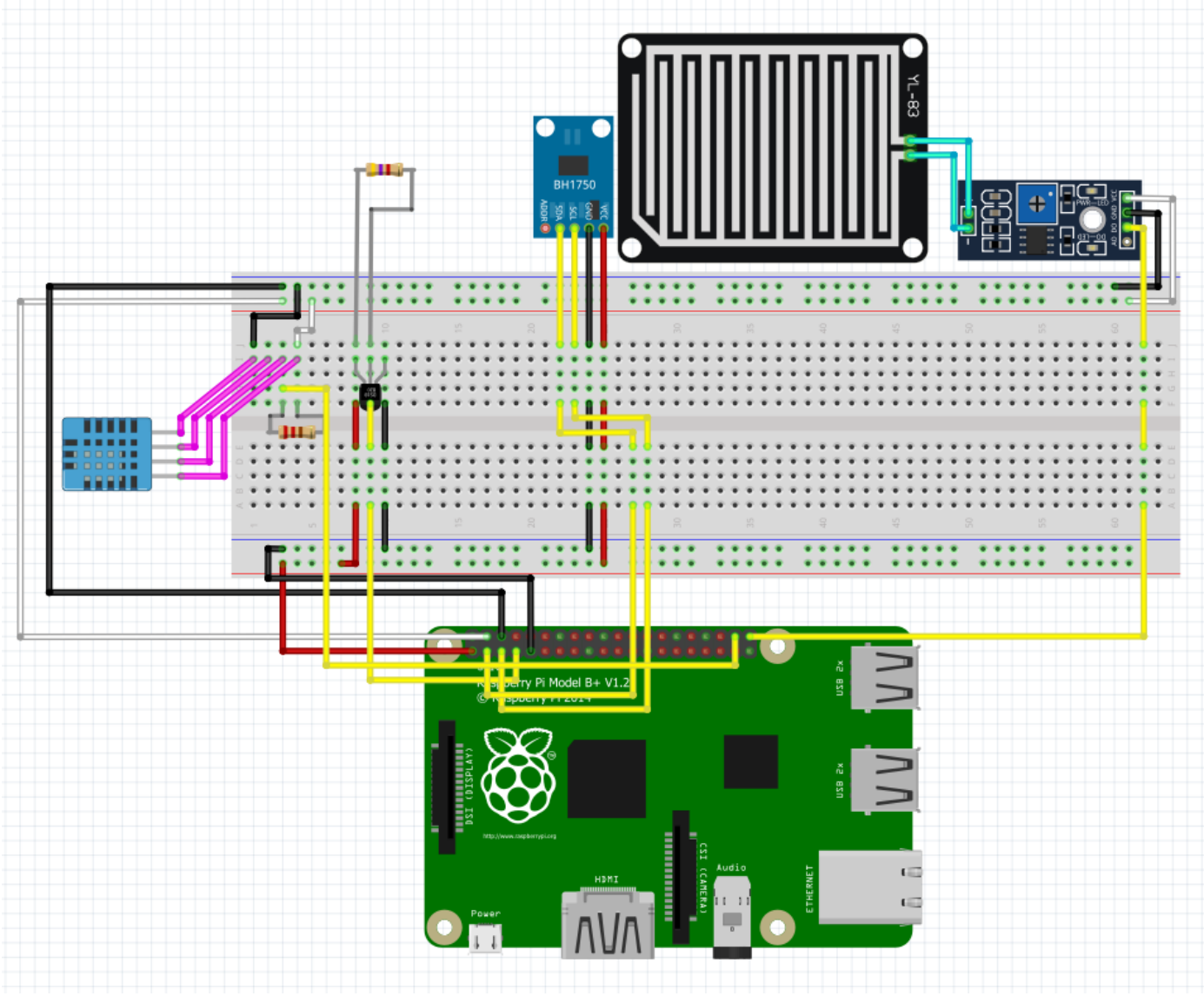
Rysunek 7: Moduł YL-83

4 Konfiguracja Raspberry Pi

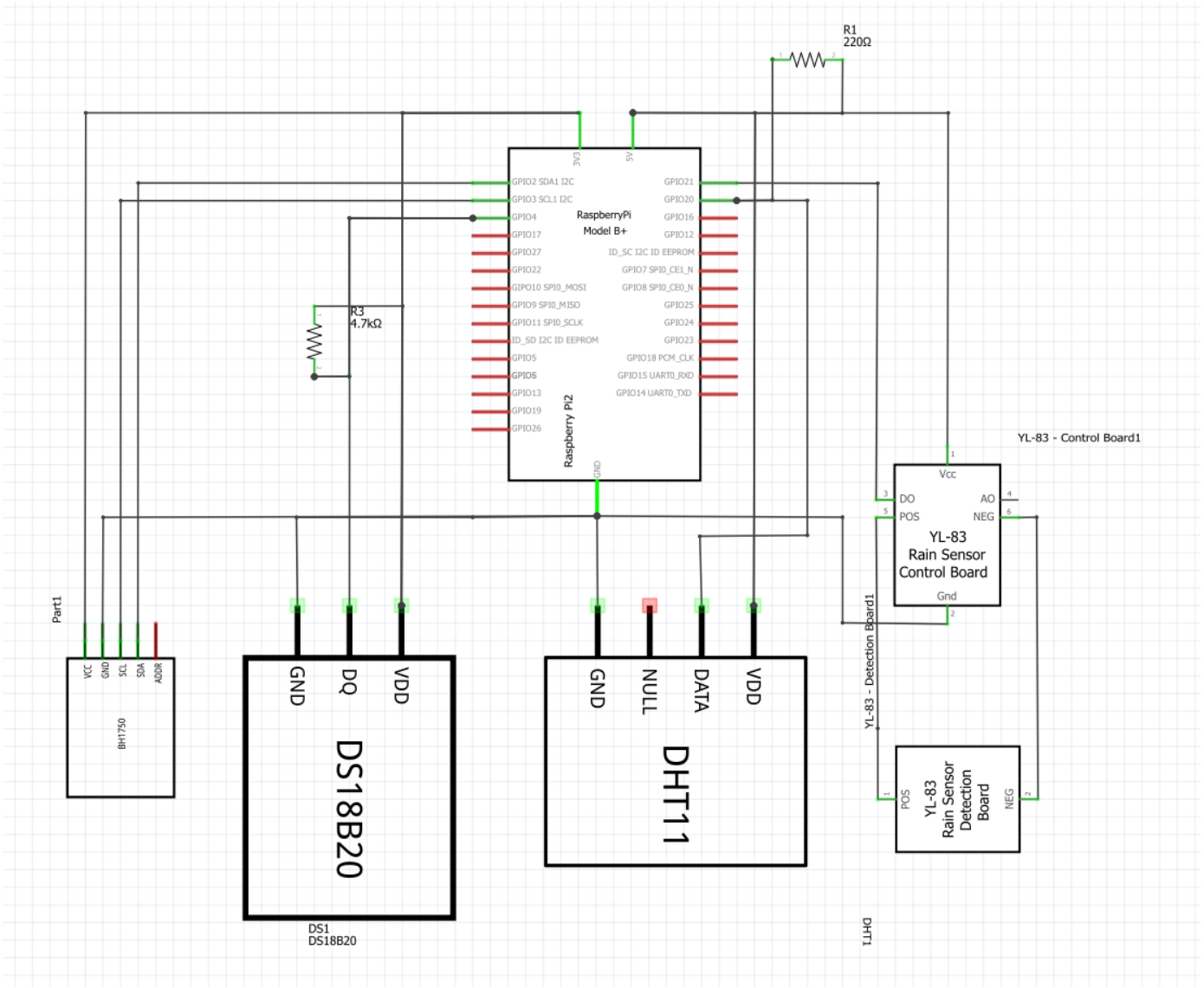
Karta sieciowa skonfigurowana jest statycznie 192.168.0.5 255.255.255.0 i można łączyć się do niej poprzez protokół SSH.
Skrypt pobierające dane i wysyłające je do serwera bazodanowego napisany jest w pełni w języku programowanie Python3. Skrypt uruchamiany jest systematycznie w tle co jedną minutę dzięki demonomi cron. Aktualne ustawienie czasowe jest na potrzeby projekty. Z oczywistych względów nie jest konieczna taka częstotliwość aktualizacji pomiarów.

4.1 Połączenie fizyczne czujników

Sensory fizycznie podłączone są dokładnie tak samo jak na schematach pokazanych poniżej.



Rysunek 8: Układ połączeń w programie Fritzing



Rysunek 9: Schemat połączeń w programie Fritzing

4.2 Pobieranie danych z czujników

4.2.1 DS18B20

Do wygodnego pobierania danych z czujnika korzystamy z biblioteki w1thermsensor: `https://github.com/timofurrer/w1thermsensor`

Listing 1: Funkcja zwracajaca temperaturę

```
1     def getTemp():
2         sensor = w1thermsensor.W1ThermSensor()
3         return sensor.get_temperature()
```

4.2.2 DHT11

Tak jak w przypadku powyżej dla tego czujnika też jest dostępna biblioteka Adafruit DHT11: `https://github.com/adafruit/Adafruit_Python_DHT`

Listing 2: Funkcja zwracająca wilgotność

```
1     def getHumidity():
2         humidity, temperature = Adafruit_DHT.read_retry(11, 20)
3         return humidity
```

W drugiej linii liczba 11 oznacza model sensora. Natomiat 20 oznacza numer portu GPIO, do którego podłączony jest czujnik.

4.2.3 BH1750

Czujnik ten przesyła dane przez magistralę I^2C. Do łatwiejszego używania tej magistrali została przygotowana biblioteka *smbus*.
Na początku funkcji zostają zdefiniowane parametry takie jak rozdzielczość, adres urządzenia oraz inne wartości dostępne w dokumentacji czujnika. Funkcja zwraca wartość liczbową natężenia światłą w luxach.

Listing 3: Funkcja zwracajaca natężenie światła

```
1     def getLux():
2         # Define some constants from the datasheet
3
4         DEVICE      = 0x23 # Default device I2C address
5
6         POWER_DOWN = 0x00 # No active state
7         POWER_ON    = 0x01 # Power on
8         RESET       = 0x07 # Reset data register value
9
10        # Start measurement at 4lx resolution. Time typically 16ms.
11        CONTINUOUS_LOW_RES_MODE = 0x13
12        # Start measurement at 1lx resolution. Time typically 120ms
13        CONTINUOUS_HIGH_RES_MODE_1 = 0x10
14        # Start measurement at 0.5lx resolution. Time typically 120ms
15        CONTINUOUS_HIGH_RES_MODE_2 = 0x11
16        # Start measurement at 1lx resolution. Time typically 120ms
17        # Device is automatically set to Power Down after measurement.
18        ONE_TIME_HIGH_RES_MODE_1 = 0x20
19        # Start measurement at 0.5lx resolution. Time typically 120ms
20        # Device is automatically set to Power Down after measurement.
21        ONE_TIME_HIGH_RES_MODE_2 = 0x21
22        # Start measurement at 1lx resolution. Time typically 120ms
23        # Device is automatically set to Power Down after measurement.
24        ONE_TIME_LOW_RES_MODE = 0x23
25
26        #bus = smbus.SMBus(0) # Rev 1 Pi uses 0
27        bus = smbus.SMBus(1)  # Rev 2 Pi uses 1
28
29        def convertToNumber(data):
30            # Simple function to convert 2 bytes of data
31            # into a decimal number. Optional parameter 'decimals'
32            # will round to specified number of decimal places.
33            result=(data[1] + (256 * data[0])) / 1.2
34            return (result)
35
36        def readLight(addr=DEVICE):
37            # Read data from I2C interface
38            data = bus.read_i2c_block_data(addr,ONE_TIME_HIGH_RES_MODE_1)
39            return convertToNumber(data)
40
41        return format(readLight(),'.2f')
```

4.2.4 YL-83

Funkcja jest tak zaprojektowana, że zwraca 0 (fałsz) gdy nie pada deszcz, a 1 (prawdę) gdy pada. Do poprawnego działa potrzebujemy zaimportować bibliotekę RPi.GPIO, która pomaga w kominukacji ze złączami

GPIO.

Listing 4: Funkcja sprawdzająca czy pada deszcz

```
1       def isRain():
2           GPIO.setmode(GPIO.BCM)
3           GPIO.setup(21, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
4
5           rain = GPIO.input(21)
6           if rain: return 0
7           else:    return 1
```

4.3 Komunikacja z bazą danych

Funkcja przedstawiona poniżej ma za zadanie łączyć się z bazą danych i dodawać nowy rekord z aktualnymni danymi pobranymi z czujników. Aby połączyć się MySQL należy zaimportować mysql.connector.
Konfigurację, którą widać poniżej można skonfigurować tak aby w kodzie źródłowym nie było w tak łatwy sposób dostępu do danych dostępowych. Jednak w celu uproszczenia zostały one dodane w jak najprostrzy sposób.

Listing 5: Funkcja komunikująca się z bazą danych

```
1       def addToDataBase(time, temp, lux, humidity, rain):
2           #RELEASE: transfer to file
3           config = {
4               'user': 'remote',
5               'password': 'asd',
6               'host': '192.168.0.6',
7               'database': 'weather_station',
8               'raise_on_warnings': True
9           }
10          ########################################
11
12          #RELEASE: catch exceptions
13          cnx = mysql.connector.connect(**config)
14          cursor = cnx.cursor()
15
16          add_row = ("INSERT INTO Surveys "
17                  "(datePlace, tempValue, luxValue, humidityValue, isRain)"
18                  "VALUES (%s, %s, %s, %s, %s)")
19          newData = (time, temp, lux, humidity, rain)
20
21          cursor.execute(add_row, newData)
22          emp_no = cursor.lastrowid
23
24          cnx.commit()
25
26          cursor.close()
27          cnx.close()
```

4.4 Pełny skrypt wykonujące określone zadanie

Listing 6: Pełen skrypt wywołujący się co minutę

```
1       #!/usr/bin/python
2
3       from __future__ import print_function
4       from datetime import date, datetime
5       #import sys
```

```
6       import mysql.connector #import lib for mySQL
7       import w1thermsensor #import lib for DS18B20
8       import smbus #import lib for I2C
9       import Adafruit_DHT #import lib for DHT
10      import RPi.GPIO as GPIO
11
12
13      def addToDataBase(time, temp, lux, humidity, rain):
14          #RELEASE: transfer to file
15          config = {
16              'user': 'remote',
17              'password': 'asd',
18              'host': '192.168.0.6',
19              'database': 'weather_station',
20              'raise_on_warnings': True
21          }
22          ########################################
23
24          #RELEASE: catch exceptions
25          cnx = mysql.connector.connect(**config)
26          cursor = cnx.cursor()
27
28          add_row = ("INSERT INTO Surveys "
29                  "(datePlace, tempValue, luxValue, humidityValue, isRain)"
30                  "VALUES (%s, %s, %s, %s, %s)")
31          newData = (time, temp, lux, humidity, rain)
32
33          cursor.execute(add_row, newData)
34          emp_no = cursor.lastrowid
35
36          cnx.commit()
37
38          cursor.close()
39          cnx.close()
40
41      def getTemp():
42          sensor = w1thermsensor.W1ThermSensor()
43          return sensor.get_temperature()
44
45      def getLux():
46          # Define some constants from the datasheet
47
48          DEVICE      = 0x23 # Default device I2C address
49
50          POWER_DOWN = 0x00 # No active state
51          POWER_ON   = 0x01 # Power on
52          RESET      = 0x07 # Reset data register value
53
54          # Start measurement at 4lx resolution. Time typically 16ms.
55          CONTINUOUS_LOW_RES_MODE = 0x13
56          # Start measurement at 1lx resolution. Time typically 120ms
57          CONTINUOUS_HIGH_RES_MODE_1 = 0x10
58          # Start measurement at 0.5lx resolution. Time typically 120ms
59          CONTINUOUS_HIGH_RES_MODE_2 = 0x11
60          # Start measurement at 1lx resolution. Time typically 120ms
61          # Device is automatically set to Power Down after measurement.
62          ONE_TIME_HIGH_RES_MODE_1 = 0x20
```

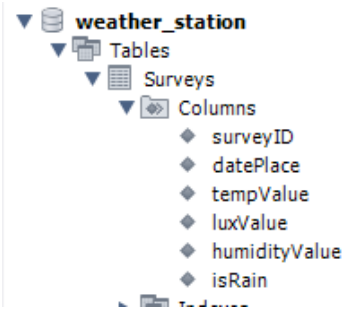
```
63          # Start measurement at 0.5lx resolution. Time typically 120ms
64          # Device is automatically set to Power Down after measurement.
65          ONE_TIME_HIGH_RES_MODE_2 = 0x21
66          # Start measurement at 1lx resolution. Time typically 120ms
67          # Device is automatically set to Power Down after measurement.
68          ONE_TIME_LOW_RES_MODE = 0x23
69
70          #bus = smbus.SMBus(0) # Rev 1 Pi uses 0
71          bus = smbus.SMBus(1)  # Rev 2 Pi uses 1
72
73          def convertToNumber(data):
74              # Simple function to convert 2 bytes of data
75              # into a decimal number. Optional parameter 'decimals'
76              # will round to specified number of decimal places.
77              result=(data[1] + (256 * data[0])) / 1.2
78              return (result)
79
80          def readLight(addr=DEVICE):
81              # Read data from I2C interface
82              data = bus.read_i2c_block_data(addr,ONE_TIME_HIGH_RES_MODE_1)
83              return convertToNumber(data)
84
85
86          return format(readLight(),'.2f')
87
88      def getHumidity():
89          humidity, temperature = Adafruit_DHT.read_retry(11, 20)
90          return humidity
91
92      def isRain():
93          GPIO.setmode(GPIO.BCM)
94          GPIO.setup(21, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
95
96          rain = GPIO.input(21)
97          if rain: return 0
98          else:    return 1
99
100     now = datetime.now()
101     temp = getTemp()
102     lux = getLux()
103     humidity = getHumidity()
104     rain = isRain()
105
106     addToDataBase(now, temp, lux, humidity, rain)
```

5 Konfiguracja serwera bazodanowego i WWW

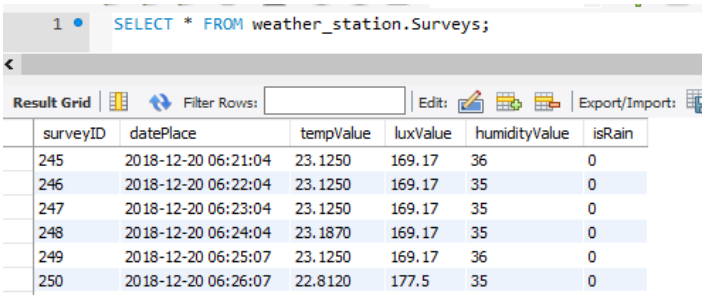
Serwer uruchomiony jest na wirtualnej maszynie dzięki narzędziu do wirtualizacji Oracle VirtualBox. System zainstalowany na maszynie to: Ubuntu Server 18.04.1 LTS.
Dodatkowo zainstalowane zostało Nginx Web Server, MySQL, a także PHP.

5.1 Baza danych

Baza danych składa się z jednej tabeli, w której umieszczamy czas pobrania pomiarów, a także pomiary.



Rysunek 10: Struktura bazy danych

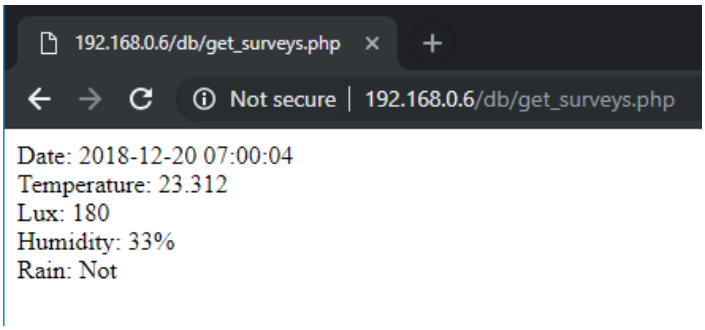


surveyID	datePlace	tempValue	luxValue	humidityValue	isRain
245	2018-12-20 06:21:04	23.1250	169.17	36	0
246	2018-12-20 06:22:04	23.1250	169.17	35	0
247	2018-12-20 06:23:04	23.1250	169.17	35	0
248	2018-12-20 06:24:04	23.1870	169.17	35	0
249	2018-12-20 06:25:07	23.1250	169.17	36	0
250	2018-12-20 06:26:07	22.8120	177.5	35	0

Rysunek 11: Zawartość tabeli

5.2 Serwer WWW

Na serwerze WWW wykonuje się skrypt, który łączy się do bazy danych i pobiera ostatni rekord z tablicy pomiarów. Następnie dane wyświetlane się na odpowiedniej stronie. W tym wypadku:
`http://192.168.0.6/db/get_surveys.php`



Rysunek 12: Wyświetlone pomiary

Listing 7: Nawiązanie komunikacji z bazą danych

```
1   <?php
2       define('DB_USER', "remote"); // db user
3       define('DB_PASSWORD', "asd"); // db password
4       define('DB_DATABASE', "weather_station"); // db name
5       define('DB_SERVER', "192.168.0.6"); // db server
6
7       $con = mysqli_connect(DB_SERVER, DB_USER, DB_PASSWORD, DB_DATABASE);
8
9       // check connection
10      if(mysqli_connect_errno()){
11          echo "Failed to connect to MySQL: " . mysqli_connect_errno();
12      }
13  ?>
```

Listing 8: Wywołanie kwerendy i wyświetlenie wyników na stronie

```php
<?php
    include 'db_connect.php';

    // get last record
    $query = "SELECT datePlace, tempValue, luxValue, humidityValue,
        isRain FROM Surveys ORDER BY surveyID DESC LIMIT 1";
    $response = array();

    if($stmt = $con->prepare($query)){
        $stmt->execute();
        //Bind fethed result to variable
        $stmt->bind_result($datePlace, $tempValue,
            $luxValue, $humidityValue, $isRain);
        if($stmt->fetch()){
            echo "Date:  " . $datePlace .
                "<br>Temperature: " . $tempValue .
                "<br>Lux: " . $luxValue .
                "<br>Humidity: " . $humidityValue . "%" .
                "<br>Rain: ";

            if($isRain) echo "Yep";
            else echo "Not";
            $surveyData["surveyTemp"] = $tempValue;
            $surveyData["surveyDate"] = $datePlace;
            $result[] = $surveyData;
        }
        $stmt->close();
        $response["success"] = 1;
        $response["data"] = $result;

    } else {
        // Some error while fetchind data
        $response["success"] = 0;
        $response["message"] = mysqli_error($con);
    }

    //echo json_encode($response);
?>
```

Część III
Podsumowanie

Konfiguracja całego projektu nie odbyłaby się bez szeroko dostępnych bibliotek, które w znacznym stopniu upraszczają komunikację pomiędzy czujnikami oraz portami komunikacyjnymi. Dodatkowo większość urządzeń wykorzystanych w projekcie ma rozbudowaną dokumentację, która jest nieodzownym pomocnikiem.
Cudownym faktem jest to iż dużo rozwiązań, przykładów zastosowań tych urządzeń jest szerokodostępnych w Internecie. Poznając tajniki komunikacji poprzez magistrale, podłączania urządzeń możemy natknąć się na sporo materiałów edukacyjnych.
Podczas pracy z bazą danych oraz stroną internetową korzysta się również z szeroko dostępnych narzędzi jak PHP, NGINX, a także MySQL. W celu zastosowania tych elementów konieczna jest odpowiednia konfiguracja środowiska, a także serwera. Niezbędnym elementem jest również ugruntowana wiedza z zakresu działania sieci komputerowych, baz danych oraz stron internetowych.

Spis rysunków

Listings