Pomiary z udostępnieniem wyników w sieci lokalnej

Użytkowanie Odroid-C1+ (4)

Podczas pracy przy projektach programistycznych, bardzo rzadko używa się wyłącznie jednego języka programowania. Najczęściej projekty są podzielone na moduły, które mogą być tworzone za pomocą różnych narzędzi i bibliotek. Dotyczy to także systemów wbudowanych, zwłaszcza tych działających pod kontrolą systemu operacyjnego, takiego jak Linux. W artykule, będącym kolejną częścią cyklu poświęconego ODROIDowi, zostanie pokazany sposób na pomiar wilgotności oraz temperatury z udostępnieniem wyników w sieci lokalnej, z wykorzystaniem języków C, Python i JavaScript.

Fotografia 1. Moduł z czujnikiem HTS221

Jako pierwszy zostanie opisany użyty w projekcie, pokazany na **fotografii 1**, sensor HTS221 (*https://goo.gl/AFFkfS*). Jest to czujnik wilgotności i temperatury firmy ST, komunikujący się za pomocą interfejsu SPI lub I²C. Przed rozpoczęciem pomiarów nie wymaga kalibracji, ale za to jest konieczna interpolacja zmierzonych wyników za pomocą zapisanych w pamięci urządzenia współczynników. W przykładzie sensor HTS221 dołączono do magistrali I²C Odroida w sposób opisany w **tabeli 1**.

Sterownik HTS221

Po wykonaniu połączeń czujnika i płytki Odroida, można rozpocząć pisanie sterownika. Podobnie jak w poprzedniej części, zostanie użyty podsystem *iio* (*Industrial Input Output*) jądra, jednak tym razem sterownik będzie obsługiwał dwa kanały pomiarowe i jeden kanał do testowania komunikacji z urządzeniem. Do pracy będą oczywiście potrzebne źródła jądra oraz kompilator, co opisano wcześniej.

Tabela 1. Połączenie Odroida z modułem KAmodHTS221

Odroid	HTS221 (KAmodHTS221)
1 (3V3)	+VIN
6 (GND)	GND
3 (SDA1)	SDA
5 (SCL1)	SCL

W pierwszym kroku, w katalogu ze źródłami Linuxa, zostaną dodane pliki niezbędne do kompilacji nowego sterownika:

- **drivers/iio/humidity/Kconfig** (dodaje sekcję czujników wilgotności do konfiguracji jądra),

```
# Humidity sensors
menu „Humidity sensors”
config HTS221
tristate „STMicroelectronics humidity and temperature sensor”
depends on I2C
help
  Say yes here to build support for HTS221 humidity and temperature
  sensor.
  To compile this driver as a module, choose M here: the module
  will be called hts221.
endmenu
```

- **drivers/iio/humidity/Makefile** (dodaje kod sterownika do kompilacji jądra)

```
# Makefile for industrial I/O Humidity sensor drivers
obj-$(CONFIG_HTS221) += hts221.o
```

Ponadto, należy zmodyfikować plik *drivers/iio/Kconfig*, dodając ścieżkę do utworzonego pliku *drivers/iio/humidity/Kconfig* – source „drivers/iio/humidity/Kconfig”, a w pliku *drivers/iio/Makefile* podać ścieżkę do nowego katalogu *humidity* – obj-y += humidity/.

Nowy sterownik będzie też wymagał drobnej zmiany w samym podsystemie *iio*. W pliku nagłówkowym *include/linux/iio/types.h*, do enumeracji *iio_chan_type*, zawierającej typy mierzonych wartości, należy dodać dwa pola: *IIO_ID* oraz *IIO_HUMIDITY*. Pierwsze z nich będzie używane do sprawdzenia połączenia z sensorem, natomiast drugie do pomiaru wilgotności. Do pomiaru temperatury posłuży istniejące już pole *IIO_TEMP*. Dodane dwa nowe typy kanałów wymagają także nazw plików tworzonych automatycznie po rejestracji urządzenia wykorzystującego sterownik. Można je ustawić w tablicy *iio_chan_type_name_spec*, znajdującej się w pliku *drivers/iio/industrialio-core.c*, dodając na końcu:

```
[IIO_ID] = „id”,
[IIO_HUMIDITY] = „humidity”,
```

Po wykonaniu powyższych kroków można zabrać się za przygotowanie sterownika. Sama struktura kodu jest bardzo podobna do omawianych poprzednio sterowników *MCP4716 i MCP3021*, dlatego nie będzie tutaj dokładnie opisywana. Komentarza wymagają jedynie tablica struktur *hts221_channels*, struktura *hts221_data* oraz funkcje *hts221_read_raw* i *hts221_setup*.

Tablica *hts221_channels* zawiera niezbędne informacje na temat wszystkich kanałów udostępnianych przez sterownik. W przykładzie wyodrębniono trzy kanały:

1. **IIO_ID** służący do testu komunikacji z sensorem poprzez odczyt rejestru WHO_AM_I o adresie 0x0F. Po zarejestrowaniu urządzenia w odpowiednim katalogu zostanie utworzony plik *in_id_raw*, którego odczyt zawsze powinien zwrócić wartość 0xBC.

```
.type = IIO _ ID,
.indexed = 0,
.info _ mask _ separate =
BIT(IIO _ CHAN _ INFO _ RAW),
.address = HTS221 _ WHO _ AM _ I,
```

2. **IIO_TEMP** to kanał reprezentujący pomiar temperatury. W przeciwieństwie do poprzedniego, poza bitem *IIO_CHAN_INFO_RAW*, ma ustawiony także bit *IIO_CHAN_INFO_PROCESSED*. W rezultacie, po rejestracji, zostaną utworzone dwa pliki: *in_temp_raw* i *in_temp_input*. Odczyt pierwszego z nich zwróci 16-bitową wartość ze znakiem, odczytaną z rejestrów 0x2A oraz 0x2B, natomiast po odczytaniu drugiego pliku zostanie zwrócona wartość temperatury po interpolacji. Ponadto, w adresie rejestru został ustawiony najstarszy bit, który oznacza inkrementację adresu podczas transakcji na magistrali I²C. Może być to wykorzystane do odczytu dwóch lub więcej kolejnych rejestrów urządzenia.

```
.type = IIO _ TEMP,
.indexed = 0,
.info _ mask _ separate = BIT(IIO _ CHAN _ INFO _
RAW) | BIT(IIO _ CHAN _ INFO _ PROCESSED),
.address = HTS221 _ TEMP _ OUT |
HTS221 _ INC _ ADDR,
```

3. **IIO_HUMIDITY** jest kanałem służącym do odczytu wilgotności, skonfigurowanym tak samo, jak kanał temperatury. Po rejestracji generuje pliki *in_humidity_raw* i *in_humidity_input*.

```
.type = IIO _ HUMIDITY,
.indexed = 0,
.info _ mask _ separate = BIT(IIO _ CHAN _ INFO _
RAW) | BIT(IIO _ CHAN _ INFO _ PROCESSED),
.address = HTS221 _ HUMIDITY _ OUT |
HTS221 _ INC _ ADDR,
```

Dane sterownika przechowywane są w strukturze *hts221 _ data*:

```
struct hts221 _ data {
    struct i2c _ client *client;
    int aHx1000;
    int bHx1000;
    int aTx1000;
    int bTx1000;
};
```

Są to: wskaźnik na klienta i2c (dostarczany podczas rejestracji) oraz współczynniki *a* i *b* prostych do interpolacji wyników pomiaru temperatury i wilgotności. Do nazw współczynników dodano *x1000* w celu podkreślenia, że ich wartości zostały pomnożone przez 1000. Mnożenie to jest wykonywane dla uniknięcia operacji zmiennoprzecinkowych w kodzie modułu jądra, którym jest sterownik. Należy mieć na uwadze, że takie podejście zmniejsza dokładność obliczeń podczas interpolacji.

Listing 1. Funkcja hts221_read_raw

```
static int hts221_read_raw(struct iio_dev *indio_dev, struct iio_chan_spec const
*chan, int *val, int *val2, long mask)
{
    struct hts221_data *data = iio_priv(indio_dev);
    char buf[2];
    int raw;
    switch (mask) {
        case (IIO_CHAN_INFO_RAW):
            switch (chan->type) {
                case (IIO_ID):
                    hts221_read_i2c_value(data->client, chan->address, buf, 1);
                    *val = (int)buf[0];
                    return IIO_VAL_INT;
                case (IIO_TEMP):
                case (IIO_HUMIDITY):
                    hts221_read_i2c_value(data->client, chan->address, buf, 2);
                    *val = (short)((short)buf[0] + ((short)buf[1] << 8));
                    return IIO_VAL_INT;
                default:
                    return -EINVAL;
            }
        case (IIO_CHAN_INFO_PROCESSED):
            switch (chan->type) {
                case (IIO_TEMP):
                    hts221_read_i2c_value(data->client, chan->address, buf, 2);
                    raw = (short)((short)buf[0] + ((short)buf[1] << 8));
                    raw = (raw * data->aTx1000 + data->bTx1000);
                    *val = raw / 1000;
                    *val2 = (raw % 1000) * 1000;
                    return IIO_VAL_INT_PLUS_MICRO;
                case (IIO_HUMIDITY):
                    hts221_read_i2c_value(data->client, chan->address, buf, 2);
                    raw = (short)((short)buf[0] + ((short)buf[1] << 8));
                    raw = (raw * data->aHx1000 + data->bHx1000) / 1000;
                    if (raw < 0) *val = 0;
                        else if (raw > 100) *val = 100;
                            else *val = raw;
                    return IIO_VAL_INT;
                default:
                    return -EINVAL;
            }
        default:
            return -EINVAL;
    }
}
```

Listing 2. Informacja o sensorze współpracującym z płytką

```
i2c@c1108500{ /*I2C-A*/
    compatible = „amlogic,aml_i2c”;
    dev_name = „i2c-A”;
    status = „ok”;
    reg = <0xc1108500 0x20>;
    device_id = <1>;
    pinctrl-names=”default”;
    pinctrl-0=<&a_i2c_master>;
    #address-cells = <1>;
    #size-cells = <0>;
    use_pio = <0>;
    master_i2c_speed = <100000>;
    hts221@5f{
        compatible = „st,hts221”;
        reg = <0x5f>;
    };
};
```

Spośród funkcji znajdujących się w sterowniku HTS221 warto zwrócić szczególną uwagę na *hts221_setup*. Jest ona wywoływana podczas rejestracji urządzenia i wykonuje jego konfigurację oraz oblicza współczynniki prostych interpolacyjnych na podstawie odczytanych rejestrów kalibracyjnych. Przy ich odczycie należy zwrócić uwagę na to, które są wartościami ze znakiem, a które bez.

Funkcja *hts221_read_raw* jest z kolei wywoływana przy każdym odczycie plików sterownika. Na podstawie przekazywanych argumentów decyduje ona, który kanał jest odczytywany i czy należy zwrócić wartości bezpośrednio odczytane z rejestrów, czy też interpolowane. Decyzja o wartości zwracanej podejmowana jest na podstawie argumentów *mask* i *chan* (pole *type*). Wartości wilgotności są liczbami całkowitymi od 0 do 100. Natomiast temperatura jest zwracana jako liczba z przecinkiem, której część całkowita jest wpisywana do zmiennej *val*, a ułamkowa do *val2*. Funkcja musi zwrócić typ odczytywanej wartości, lub informację o błędzie. Kod funkcji *hts221_read_raw* przedstawiono na **listingu 1**. Informację o współpracującym sensorze, podobnie jak w przypadku poprzednich urządzeń, należy umieścić w pliku *arch/arm/boot/dts/*

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **77322**, pass: **8qxonzsb**

meson8b_odroidc.dts w sekcji magistrali I2C-A (**listing 2**).

Po przygotowaniu kodu sterownika można przygotować jądro do kompilacji wywołując kolejno:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- odroidc_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

Następnie należy dodać do kompilacji jądra sterownik I2C, podsystem IIO i napisany sterownik HTS221:

```
Device Drivers > Amlogic
Device Drivers > I2C Hardware Bus support > Amlogic I2C Driver
Device Drivers > Industrial I/O support
Device Drivers > Industrial I/O support > Humidity sensors > STMicroelectronics humidity and temperature sensor
```

Ostatni z nich warto dodać jako moduł, zwłaszcza podczas testowania. Dzięki temu wystarczy wymienić moduł na karcie SD (*/lib/modules/3.10.104/kernel/drivers/iio/humidity/hts221.ko*, ścieżka może się różnić w zależności od wersji źródeł), bez potrzeby ponownej kompilacji całego jądra. Kompilację należy przeprowadzić za pomocą poleceń:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=../modules modules_install
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- uImage
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs
```

Po zakończeniu skopiować obraz jądra (*./arch/arm/boot/uImage*) wraz z drzewem urządzeń (*./arch/arm/boot/dts/meson8b_odroidc.dtb*) na partycję BOOT karty SD, a moduły na partycję systemową do katalogu */lib/*. Powyższe kroki zostały dokładniej opisane w poprzednich częściach cyklu.

Po uruchomieniu Odroida powinien zostać utworzony katalog */sys/bus/iio/devices/iio:device0* zawierający pliki sensora HTS221 (**rysunek 2**). Jego działanie można zweryfikować odczytując plik *in_id_raw*, który powinien zawierać wartość 188.

Listing 3. Aplikacja odczytująca wyniki pomiarów i umieszczająca je w bazie danych /home/odroid/measurements/measurements_database

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <fcntl.h>
#include <sqlite3.h>

static bool executeStatement(sqlite3* database, const char* statement)
{
    char* errMsg;
    if(sqlite3_exec(database, statement, NULL, NULL, &errMsg) != SQLITE_OK) {
        fprintf(stderr, „Can't create table: %s\n", errMsg);
        return false;
    }
    return true;
}

int main(int argc, char* argv[])
{
    sqlite3* database;
    //Open the measuements database.
    const char* databasePath = „/home/odroid/measurements/measurements_database";
    if(sqlite3_open(databasePath, &database) != SQLITE_OK) {
        fprintf(stderr, „Can't open database: %s\n", sqlite3_errmsg(database));
        sqlite3_close(database);
        return -1;
    }
    //Create measurements table if not exists.
    const char* createTableStatement =
        „CREATE TABLE IF NOT EXISTS measurements „
        „(Timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,"
        „ Humidity INT NOT NULL,"
        „ Temperature REAL NOT NULL)";
    if(!executeStatement(database, createTableStatement)) {
        sqlite3_close(database);
        return -1;
    }
     //Read humidity and temperature from HTS221 driver.
    const char* humidityFilePath = „/sys/bus/iio/devices/iio:device0/in_humidity_input";
    const char* temperatureFilePath = „/sys/bus/iio/devices/iio:device0/in_temp_input";
    char valueStr[128];
    int fd;
    int humidity;
    float temperature;
    if((fd = open(humidityFilePath, O_RDONLY)) < 0) {
        fprintf(stderr, „Error opening file for read: %s\n", humidityFilePath);
        return -1;
    }
    if(read(fd, valueStr, 128) <= 0) {
        fprintf(stderr, „Error reading file: %s\n", humidityFilePath);
        return -1;
    }
    if(close(fd)<0) {
        printf(„Error closing file: %s\n", humidityFilePath);
        return 1;
    }
    humidity = atoi(valueStr);
    if((fd = open(temperatureFilePath, O_RDONLY)) < 0) {
        fprintf(stderr, „Error opening file for read: %s\n", temperatureFilePath);
        return -1;
    }
    if(read(fd, valueStr, 128) <= 0) {
        fprintf(stderr, „Error reading file: %s\n", temperatureFilePath);
        return -1;
    }
    if(close(fd)<0) {
        printf(„Error closing file: %s\n", temperatureFilePath);
        return 1;
    }
    temperature = atof(valueStr);
    //Insert measured values into database.
    char insertDataStatement[128];
    sprintf(insertDataStatement, „INSERT INTO measurements (Humidity, Temperature) VALUES (%d, %f)",
    humidity, temperature);
    if(!executeStatement(database, insertDataStatement)) {
        sqlite3_close(database);
        return -1;
    }
   sqlite3_close(database);
   return 0;
}
```

Pomiary i baza danych

Kolejnym krokiem jest przygotowanie programu, który będzie w stanie odczytać i zarchiwizować wyniki pomiarów w bazie danych. Do przykładu została wybrana baza SQLite ze względu na jej kilka cech mających znaczenie w zastosowaniu w systemach wbudowanych. Nie wymaga ona serwera ani konfiguracji i jest widoczna jako plik w lokalnym systemie plików. Komunikacja z nią odbywa się za pomocą zapytań języka SQL.

Aby w kodzie programu można było korzystać z bazy danych należy dołączyć nagłówek #include <sqlite3.h>. Następnie można wywołać m. in. funkcje:

- sqlite3_open,
- sqlite3_exec,
- sqlite3_close.

Pierwsza z nich otwiera połączenie z bazą danych, druga umożliwia wykonywanie zapytań SQL przekazywanych jako łańcuchy znaków, natomiast ostatnia zamyka połączenie. Przykładowe zapytania to:

- Utworzenie tabeli o nazwie *measurements* w bazie (o ile tabela o podanej nazwie nie istnieje) – tabela będzie zawierała trzy

kolumny: *Timestamp* (automatycznie generowany znacznik czasowy podczas wstawiania danych do bazy), *Humidity* (typu całkowitego) i *Temperature* (typu zmiennoprzecinkowego):

```
CREATE TABLE IF NOT EXISTS measurements (Time-
stamp DATETIME DEFAULT CURRENT_TIMESTAMP, Humid-
ity INT NOT NULL, Temperature REAL NOT NULL);
```

- Wstawienie do tabeli przykładowych danych (data nie jest podawana, ponieważ jest generowana automatycznie):

```
INSERT INTO measurements (Humidity, Temperature) VA-
LUES (1, 1.5)
```

Odczyt danych z sensora nie wymaga komentarza, ponieważ jest to po prostu odczyt plików wymienionych w poprzednim rozdziale.

Na **listingu 3** przedstawiono kod aplikacji odczytującej wyniki pomiarów i umieszczający je w bazie danych umieszczonej w pliku */home/odroid/measurements/measurements_database*. Aplikację najłatwiej skompilować na ODROIDzie po wcześniejszym zainstalowaniu pakietów *sqlite3* i *libsqlite3-dev:*

```
sudo apt-get install sqlite3
libsqlite3-dev
gcc measurements.c -lsqlite3
-o measurements
```

Pozostaje jeszcze tylko zapewnić jej cykliczne wywoływanie w celu dokonania pomiaru. Można użyć do tego opisywanego już wcześniej programu *cron*, który jest w stanie wykonywać polecenia według podanego harmonogramu. Aby dodać nowy wpis wystarczy wywołać polecenie *crontab -e* i wpisać nową linię:

```
*/5 * * * * /home/odroid/measurements/
measurements
```

Oznacza ona, że program */home/odroid/measurements/measurements* będzie wywoływany co 5 minut.

Aplikacja webowa

Trzecią i jednocześnie ostatnią warstwą projektu jest aplikacja webowa prezentująca wyniki pomiarów odczytane z bazy danych. Do tego zadania wybrany został Python wraz z frameworkiem Flask i skryptami JavaScript Chart.js. Dzięki takiemu połączeniu można w bardzo łatwy sposób wygenerować wykresy danych pomiarowych i udostępnić je w sieci lokalnej do wyświetlenia za pośrednictwem przeglądarki internetowej.

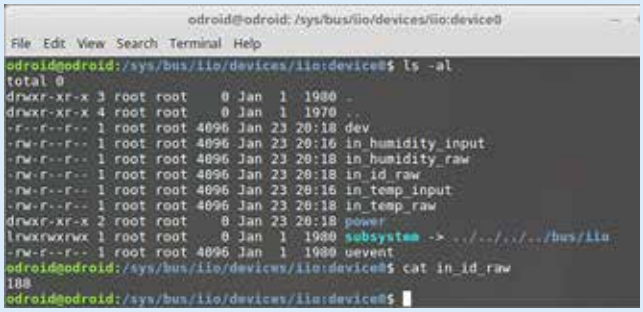
Na początek trzeba zainstalować na ODROIDzie menadżera bibliotek Pythona i pakiet Flask:

```
sudo apt-get install python-pip
sudo pip install flask
```

Sama aplikacja Pythona jest bardzo prosta i sprowadza się do kilku operacji. Pierwszą z nich jest otworzenie połączenia z bazą danych za pomocą wywołania *sqlite3.connect*, które przyjmuje pełną ścieżkę do pliku utworzonego przez program zapisujący dane, z poprzedniego rozdziału. Otrzymane połączenie można wykorzystać do pobrania danych z bazy SQLite:

```
SELECT * FROM (SELECT time(Timestamp,
,localtime') AS LocalTimestamp, Humidity,
Temperature FROM measurements ORDER BY
Timestamp DESC LIMIT 80) ORDER BY Local-
Timestamp ASC
```

Oznacza ono, że ze wszystkich danych tabeli *measurements*, posortowanych według malejących znaczników czasowych, zostanie pobranych 80 pierwszych wyników i uszeregowanych rosnąco. Równie dobrze



Rysunek 2. Pliki sterownika HTS221

Listing 4. Aplikacja testowa w Pythonie

```
from flask import Flask
from flask import Markup
from flask import Flask
from flask import render_template
import sqlite3

app = Flask(__name__)
conn = sqlite3.connect(„/home/odroid/measurements/measurements_data-
base")

selectStatement = („SELECT * FROM („
                   „SELECT time(Timestamp, \'localtime\') AS Local-
Timestamp, Humidity, Temperature „
                   „FROM measurements ORDER BY Timestamp DESC LIMIT
80'
                   „) ORDER BY LocalTimestamp ASC')

@app.route(„/")
def chart():
    dbCursor = conn.cursor()
    measurementsList = zip(*dbCursor.execute(selectStatement).fetch-
all())
    labels = measurementsList[0]
    humidityValues = measurementsList[1]
    temperatureValues = measurementsList[2]
    return render_template(„chart.html', humidityValues=humidityVal-
ues, temperatureValues=temperatureValues, labels=labels)

if __name__ == „__main__": app.run(host='0.0.0.0', port=5001)
```

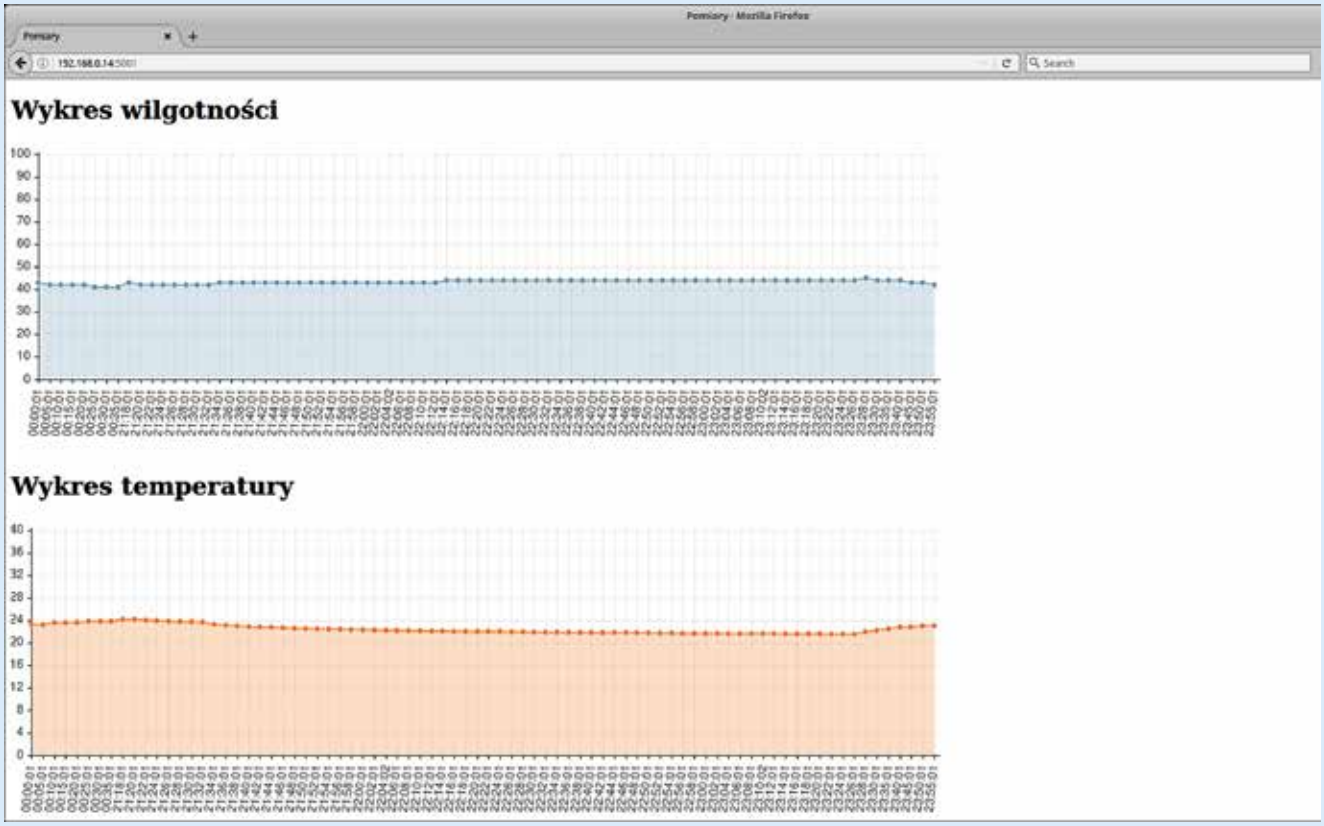
Listing 5. Fragment szablonu chart.html

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="utf-8" />
  <title>Pomiary</title>
  <script src='//www.chartjs.org/assets/Chart.min.js'></script>
</head>
<body>
  <h1>Wykres wilgotności</h1>
  <canvas id="humidityChart" width="1200" height="400"></canvas>
  <script>
    var lineData = {
    labels : [{% for item in labels %}
                „{{item}}",
              {% endfor %}],
    datasets : [
       {
            fillColor: „rgba(151,187,205,0.2)",
            strokeColor: „rgba(151,187,205,1)",
            pointColor: „rgba(151,187,205,1)",
            pointStrokeColor: „#fff",
            pointHighlightFill: „#fff",
            pointHighlightStroke: „rgba(151,187,205,1)",
            data : [{% for item in humidityValues %}
                      {{item}},
                    {% endfor %}]
      }]
    }
    Chart.defaults.global.showTooltips = false;
    Chart.defaults.global.animationSteps = 50;
    Chart.defaults.global.animationEasing = „easeOutBounce";
    Chart.defaults.global.responsive = false;
    Chart.defaults.global.scaleLineColor = „black";
    Chart.defaults.global.scaleFontSize = 16;

    var mychart = document.getElementById(„humidityChart").getContex-
t(„2d");

    steps = 10
    max = 100
    min = 0

    var humidityChart = new Chart(mychart).Line(lineData, {
        scaleOverride: true,
        scaleSteps: steps,
        scaleStepWidth: Math.ceil(max / steps),
        scaleStartValue: 0,
        scaleShowVerticalLines: true,
        scaleShowGridLines : true,
        barShowStroke : true,
        scaleShowLabels: true,
    });
  </script>
```



Rysunek 3. Strona serwisu

można by pobrać wszystkie dane a ich selekcji dokonać w kodzie Pythona. Zapytanie to jest wykonywane za każdym razem, gdy nastąpi połączenie z serwerem, czyli kiedy w przeglądarce zostanie otwarta strona aplikacji. Jest to także jedyny moment, kiedy prezentowane dane są aktualizowane.

Z danych pobranych z bazy są następnie tworzone trzy listy: znaczników czasu (sformatowanych jako łańcuchy znakowe wewnątrz zapytania SQL), wilgotności i temperatury. Listy te są przekazywane do funkcji *render_template*, która z dostępnego szablonu *chart.html* (opisanego w skrócie poniżej) i otrzymanych argumentów generuje stronę html.

Na **listingu 4** znajduje się pełny kod aplikacji Pythona. Warto zwrócić uwagę na dyrektywę *@app.route(„/”)* znajdującą się przed funkcja chart(). Oznacza ona, że funkcja ta zostanie wywołana w momencie wejścia na stronę główną serwisu znajdującą się pod adresem <IP ODROIDa>:5001. Numer portu jest podawany jako argument funkcji *app.run*.

Szablon *chart.html* należy umieścić w podkatalogu *templates/*. Jego fragment przedstawiony na **listingu 5**. Na początku, w sekcji *<head>* umieszczony został adres, z którego zostanie pobrany skrypt generujący wykresy. Skrypt ten można także zapisać lokalnie i podać do niego ścieżkę – wówczas nie będzie wymagane połączenie z Internetem. Następnie, w sekcji *<body>* umieszczono dwa wykresy (na listingu jest widoczny tylko pierwszy z nich). Wykresy mają określony rozmiar na stronie za pomocą atrybutów *width* i *height*. Większość kodu odpowiedzialna jest za wygląd i zachowanie wykresów, można więc go swobodnie modyfikować aby dostosować je do własnych potrzeb. Istotne są natomiast fragmenty:

```
labels : [{% for item in labels %}
                „{{item}}”,
          {% endfor %}]
data : [{% for item in humidityValues %}
          {{item}},
        {% endfor %}]
```

Służą one bibliotece Flask do dynamicznego generowania kodu html na podstawie przekazanych argumentów do funkcji *render_template*. Są to pętle *for* wpisujące w miejsce *item* poszczególne elementy z list *labels* i *humidityValues*, o czym można się przekonać wchodząc na stronę serwisu i oglądając źródła strony. Aby uruchomić serwis wystarczy przejść do katalogu z aplikacją Pythona i wywołać *python app.py*. Na **rysunku 3** pokazano rezultat wejścia na stronę serwisu w przeglądarce na dowolnym komputerze w sieci lokalnej.

Zakończenie

Przedstawiony przykład pokazuje jak w prosty sposób połączyć kilka narzędzi, aby otrzymać działający system pomiarowy. Dzięki podziałowi na moduły, który naturalnie wynika z zastosowanych bibliotek i języków, można niezależnie wprowadzać zmiany do rożnych warstw aplikacji. Ponadto zastosowanie bazy SQLite znacząco ułatwia zapis danych przez aplikację pomiarową, a także dostęp do nich z poziomu aplikacji odpowiedzialnej za prezentację, dzięki standardowym zapytaniom SQL.

Kompletny kod źródłowy przykładu dostępny jest w materiałach dodatkowych do artykułu. Zmodyfikowane źródła jądra zostały udostępnione jako patch, który można nanieść na lokalne repozytorium za pomocą polecenia *git apply hts221.patch*. Źródła aplikacji pomiarowej i serwisu webowego, ze względu na ustawione ścieżki do bazy danych, powinny znaleźć się w katalogu */home/odroid/measurements* na karcie SD Odroida. Ścieżki te należy zmodyfikować, aby móc zmienić układ katalogów.

Krzysztof Chojnowski

Literatura:

1. Podsystem iio *https://goo.gl/goY5v2*
2. Baza danych SQLite *https://goo.gl/5RNfP0*
3. Kurs SQL *https://goo.gl/ECjBIu*
4. Python Flask *https://goo.gl/HPsmXs*
5. Flask getting started *https://goo.gl/UwRdYN*