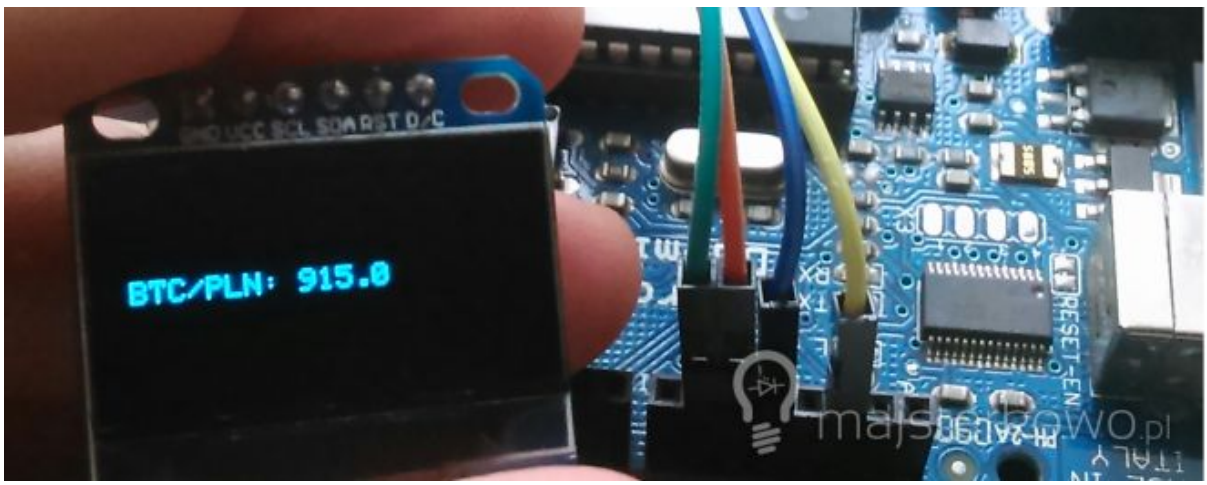
Artykuł ten przedstawia projekt łączący w sobie kilka różnych technik, które wzajemnie ze sobą współpracują. Projekt, który jest opiszę jest to zewnętrzny wyświetlacz przedstawiający aktualny kurs kryptowaluty Bitcoin. Używając przedstawionych technik można z powodzeniem zrealizować wiele innych gadżetów, jak na przykład stację pogodową, wyświetlającą aktualne dane o pogodzie pobierane z internetu.

Do zrealizowania projektu użyłem:

- płytki “Arduino Duemilanove”, która dostarcza nam programowalny mikrokontroler
- wyświetlacza Oled o rozdzielczości 128x64 punkty
- aplikacji dostarczanej wraz z Arduino, służącej do programowania mikrokontrolera
- języka Ruby, do pobierania danych z internetu i wysyłania ich do Arduino
- publicznego API dostarczanego przez serwis bitmarket.pl
- komputera PC z systemem operacyjnym GNU/Linux (Ubuntu 14.04)

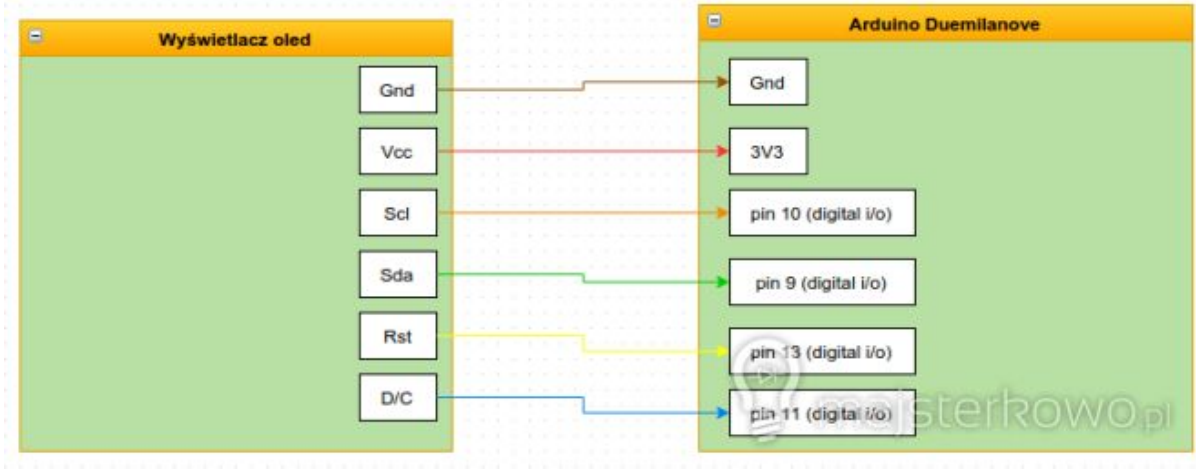
Arduino nie posiada żadnych modułów, poza wyświetlaczem OLED, zatem do pobierania danych z sieci, będziemy używać aplikacji uruchomionej na komputerze PC. Komunikacja z Arduino będzie się odbywać poprzez port USB.

Końcowy rezultat będzie wyglądał tak:




**Wyświetlanie tekstu**

Na początek podłączymy wyświetlacz do Arduino. Nazwy punktów są wypisane na obu płytkach (Arduino i wyświetlacza). Połączeń dokonujemy według schematu:




Następnie podłączamy Arduino do komputera, przy pomocy kabla USB i napiszemy prosty program, dzięki któremu sprawdzimy czy wyświetlacz jest podłączony poprawnie i działa.

Tworzymy nowy szkic w aplikacji do programowania Arduino i importujemy biblioteki:

- Adafruit GFX Library
- Adafruit SSD1306

Biblioteki dodajemy poprzez menu “Szkic > Include Library > Manage Libraries”. Jeśli nie znajdziemy tam potrzebnej biblioteki, należy sciągnąć ją z internetu i dodać przy pomocy opcji “Szkic > Include Library > Add .ZIP Library”.

Do szkicu wklejamy poniższy kod:

```
#include <SPI.h>
    #include <Wire.h>

    #include <Adafruit_SSD1306.h>
    #include <Adafruit_GFX.h>

    #define OLED_DC 11  //OLED -- D/C
    #define OLED_CS 12  //Not connected
    #define OLED_CLK 10 //OLED -- SCL
    #define OLED_MOSI 9 //OLED -- SDA
    #define OLED_RESET 13//OLED -- RST
    Adafruit_SSD1306 display(OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);

    void setup()
    {
     display.begin(SSD1306_SWITCHCAPVCC);
     display.clearDisplay();

     display.setTextSize(1.2);
     display.setTextColor(WHITE);
    }

    void loop ()
    {
     display.setCursor(0,30);
```

```
 display.clearDisplay();
 display.println(random(0,100));
 display.display();
 delay(1000);
}
```

Program należy skompilować i wgrać do Arduino. Jeśli wszystko przebiegło pomyślnie, po chwili na ekranie OLED podłączonym do Ardunio zacznie się wyświetlać losowa liczba z zakresu <0,100>. Liczba będzie się zmieniać co sekundę.


**Wyjaśnienie kodu:**

Na początku dołączamy potrzebne biblioteki:

```
#include <SPI.h>
#include <Wire.h>

#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
```

Następnie ustawiamy piny zgodnie z połączeniami jakie zrobiliśmy między Arduino i wyświetlaczem:

```
#define OLED_DC 11  //OLED -- D/C
#define OLED_CS 12  //Not connected
#define OLED_CLK 10 //OLED -- SCL
#define OLED_MOSI 9 //OLED -- SDA
#define OLED_RESET 13//OLED -- RST
Adafruit_SSD1306 display(OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);
```

Tworzymy metodę setup, w której inicjalizujemy obsługę wyświetlacza oraz ustawiamy rozmiar i kolor czcionki.

```
void setup()
    {
     display.begin(SSD1306_SWITCHCAPVCC);
     display.clearDisplay();

     display.setTextSize(1.2);
     display.setTextColor(WHITE);
    }
```

Teraz tworzymy metodę loop, która wykonywana jest cyklicznie przez Arduino:

```
void loop ()
    {
     display.setCursor(0,30);
     display.clearDisplay();
     display.println(random(0,100));
     display.display();
     delay(1000);
    }
```

W metodzie tej mamy następujące polecenia:

setCursor - ustawia kursor w pozycji x,y

clearDisplay - czyści ekran

println - wypisuję linię tekstu

random(0,100) - losuje liczbę całkowitą od 0 do 100

display - wyświetla to co zostało “narysowane” od ostatniego wyczyszczenia ekranu

delay(1000) - “usypia” na 1000ms, czyli następne wykonanie metody loop nastąpi po upływie jednej sekundy.

**Komunikacja Arduino z komuterem**

Dane do wyświetlenia na ekranie będą dostarczane poprzez port szeregowy, zatem do programu musimy dodać jego obsługę.

W metodzie setup dodajemy:

```
Serial.begin(9600);

    while (!Serial) {
     ; // wait for serial port to connect.
    }
```

Prędkość transmisji ustawiliśmy na 9600. Dla naszej aplikacji nie potrzebujemy wyższej prędkości.

Natomiast metodę loop przerobimy tak, żeby wyświetlała na ekranie odebrany przez port szeregowy tekst:

```
void loop ()
    {
      display.setCursor(0,30);
      if (Serial.available()) {
       display.clearDisplay();
       display.println(Serial.readString());
```

```
    display.display();
  }

  delay(1000);
}
```

Całość kodu na ten moment wygląda następująco:

```
#include <SPI.h>
#include <Wire.h>

#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>

#define OLED_DC 11  //OLED -- D/C
#define OLED_CS 12  //Not connected
#define OLED_CLK 10 //OLED -- SCL
#define OLED_MOSI 9 //OLED -- SDA
#define OLED_RESET 13//OLED -- RST
Adafruit_SSD1306 display(OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);

void setup()
{
  display.begin(SSD1306_SWITCHCAPVCC);
  display.clearDisplay();
  display.setTextSize(1.2);
  display.setTextColor(WHITE);
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect.
  }
}

void loop ()
{
  display.setCursor(0,30);
  if (Serial.available()) {
    display.clearDisplay();
    display.println(Serial.readString());
    display.display();
  }

  delay(1000);
}
```

Po skompilowaniu programu i wgraniu go do Arduino, możemy przetestować komunikację z urządzeniem. Do tego celu bardzo przydatny jest “Szeregowy monitor”, który znajdziemy w aplikacji do programowania Arduino, w menu narzędzia.

W “szeregowym monitorze” ustawiamy prędkość transmisji na taką jaka była zapisana w kodzie programu, czyli 9600. Następnie w pierwszym polu wpisujemy jakiś tekst, np. “abc” i klikamy “Wyślij”.


W tym momencie tekst “abc” powinien wyświetlić się na ekranie oled.


**Przesyłanie danych do Arduino, program w języku Ruby**

Napiszemy teraz prosty program w języku Ruby, który na początek będzie pełnił funkcję podobną do tej, jaką pełnił “Szeregowy Monitor”. Zatem zadaniem tego programu będzie: wysłać do arduino krótki tekst.

Oto kod programu:

```
require 'rubygems'
    require 'serialport' # use Kernel::require on windows, works better.

    #params for serial port
    port_str = "/dev/ttyUSB0"  #may be different for you
    baud_rate = 9600
    data_bits = 8
    stop_bits = 1
    parity = SerialPort::NONE

    sp = SerialPort.new(port_str, baud_rate, data_bits, stop_bits, parity)
    sp.write(“Text from Ruby”)
```

Istotne jest ustawienie prawidłowego portu. W moim przypadku jest to /dev/ttyUSBo

Skrypt zapisujemy w pliku controler.rb i uruchamiamy poprzez polecenie ruby controller.rb.

Jeśli wystąpi błąd jakiejś biblioteki, należy ją doinstalować i uruchomić polecenie ponownie.

Na pewno wymagana jest biblioteka ruby-serialport (dostępna w repozytorium ubuntu).

Jeśli wszystko poszło sprawdznie, na ekranie oled zobaczymy tekst “Text from Ruby”.


**Pobieranie danych z internetu i przesyłanie ich do Arduino**

Została nam ostatnia część aplikacji: pobranie kursu Bitcoina w skrypcie Ruby i przesłanie go do Arduino. Celowo pominę takie kwestie jak sprawdzanie poprawności odpowiedzi API, czy walidacja otrzymanych danych, aby skupić się na tym, co jest istotne dla wykonania naszego zadania.

Na początek zmienię nieco wywołanie metody println w naszej aplikacji Arduino, aby wyświetlał się tekst “BTC/PLN: ” przed wartością otrzymaną ze skryptu Ruby.

```
display.println("BTC/PLN: " + Serial.readString());
```

Ostateczny kod programu dla Arduino wygląda następująco:

```
#include <SPI.h>
    #include <Wire.h>

    #include <Adafruit_SSD1306.h>
    #include <Adafruit_GFX.h>

    #define OLED_DC 11  //OLED -- D/C
    #define OLED_CS 12  //Not connect
    #define OLED_CLK 10 //OLED -- SCL
    #define OLED_MOSI 9 //OLED -- SDA
```

```
#define OLED_RESET 13//OLED -- RST
Adafruit_SSD1306 display(OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);

void setup()
{
  Serial.begin(9600);

  while (!Serial) {
   ; // wait for serial port to connect.
  }

  display.begin(SSD1306_SWITCHCAPVCC);  // Switch OLED
  display.clearDisplay();  // Clear OLED

  display.setTextSize(1.2);
  display.setTextColor(WHITE);
}

void loop ()
{
  display.setCursor(0,30);
  if (Serial.available()) {
            display.clearDisplay();
    display.println("BTC/PLN: " + Serial.readString());
    display.display();
  }

  delay(1000);
}
```

Teraz zmodyfikujemy skrypt Ruby.

Dodajemy potrzebne nam biblioteki, do pobierania zawartości z url'a oraz do parsowania jsona (odpowiedź api jest w formacie json).

```
require 'open-uri'
require 'json'
```

Dodajemy również parametr 'url', którego wartością jest adres api, z którego będziemy pobierać dane:

```
url = 'https://www.bitmarket.pl/json/BTCPLN/ticker.json'
```

Następnie zamiast metody serial.write dodajemy pętlę:

```
while true do
  open(url) {|f|
   content = f.read()
   data = JSON.parse(content)
   sp.write(data["last"])
  }

  sleep 2
end
```

Pętla wykonywać się będzie w nieskończoność (do momentu przerwania programu). Wewnątrz pętli:

- Otwieramy url, open(url). Połączenie zapisujemy w zmiennej f
- Pobieramy content z urla używając metody f.read(), content zapisujemy w zmiennej content
- Parsujemy json'a (JSON.parse(content)) i wynik parsowania zapisujemy w zmiennej data.

- Wyciągamy z data klucz last i przesyłamy go portem szeregowym do Arduino sp.write(data["last"])
- Wykonujemy sleep 2, czyli następny przebieg pętli wykona się po upływie dwóch sekund.

Ostateczny skrypt ruby wygląda następująco:

```
require 'rubygems'
    require 'serialport' # use Kernel::require on windows, works better.
    require 'open-uri'
    require 'json'

    url = 'https://www.bitmarket.pl/json/BTCPLN/ticker.json'

    #params for serial port
    port_str = "/dev/ttyUSB0"  #may be different for you
    baud_rate = 9600
    data_bits = 8
    stop_bits = 1
    parity = SerialPort::NONE

    sp = SerialPort.new(port_str, baud_rate, data_bits, stop_bits, parity)

    #just write forever
    while true do
      open(url) {|f|  #url must specify the protocol
        content = f.read()
        data = JSON.parse(content)
        sp.write(data["last"])
      }
      sleep 2
    end
```

Jeśli wszystko poszło sprawnie, na ekranie OLED zobaczymy aktualny kurs kryptowaluty Bitcoin do PLN w formacie: “BTC/PLN: 915.0”. Czas uśpienia obu aplikacji możemy wydłużyć, ponieważ API Bitmarket.pl nie odświeża danych tak często, żeby był możliwy odczyt nowej wartości co dwie sekundy. Z drugiej strony, krótki czas uśpienia będzie przydatny w przypadku innych API, które oferują odświeżanie danych w czasie rzeczywistym.


**Konkluzja**

Napisaliśmy kod aplikacji Arduino oraz skrypt Ruby, które wykonują zadanie założone w projekcie. Po opanowaniu wiedzy przedstawionej w artykule, napisanie samodzielnie obu części nie powinno zająć więcej niż pół godziny. Dodatkowo nauczyliśmy się, jak podłączyć wyświetlacz OLED do Arduino, oraz jak go używać.

W celu dalszego kształcenia się polecam dorobić następujące funkcjonalności:

- Wyświetlanie symboli graficznych na wyświetlaczu oled (metoda display.drawBitmap() oraz narzędzie Img2Code, służące do konwersji bitmap na kod (dostarczane wraz z biblioteką)
- Ostatnich 10 wartości i rysowanie ich wykresu (zapis ostatnich wartości można rozwiązać po stronie Ruby, aby ograniczyć użycie pamięci Arduino).
- Pobieranie danych z innych api, np. danych pogodowych
- Niezależna aplikacja Arduino (nie wymagająca komunikacji z komputerem). Np. Licznik rowerowy - będziemy do niego potrzebować czujnika magnetycznego do zliczania obrotów koła oraz baterii/akumulatora do zasilania urządzenia.