

# Workshop – budowa “IoT”

Tomasz Kopacz



Czas trwania NIEZNANY

# Składniki

12 stanowisk – Adafruit Windows IoT Starter Kit  
+ 1 Dexter (jak ktoś chętny ☺)

Raczej do obejrzenia:

+ 1 Dragonboard  
+ 1 Toradex  
+ 1 MinnowBoard Max (też dla osoby chętnej da się zrobić coś)

# Dexter GroovePi

## GrovePi Board

Grove – Sound Sensor

Grove – Temperature and Humidity

Grove – Light Sensors

Grove – Relay

Grove – Button

Grove – Ultrasonic Ranger

Grove – Rotary Angle Sensor

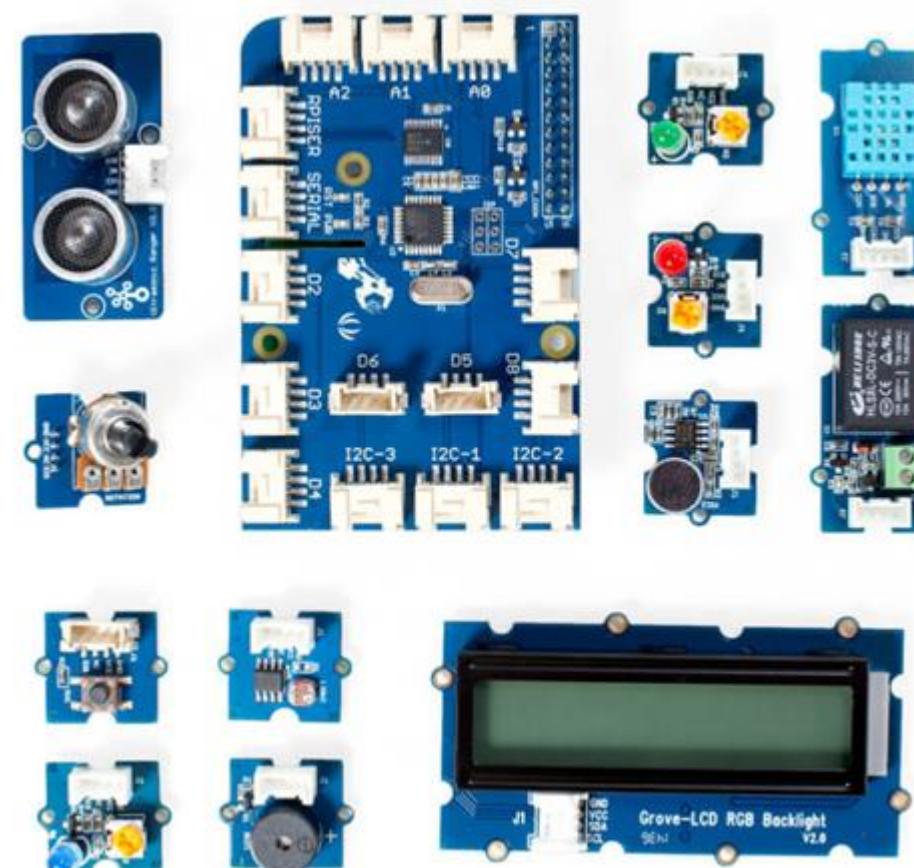
Grove – LCD RGB Backlight

Grove – Buzzer

Grove – Red LED

Grove – Blue LED

Grove – Green LED



# A dalej – dygresja

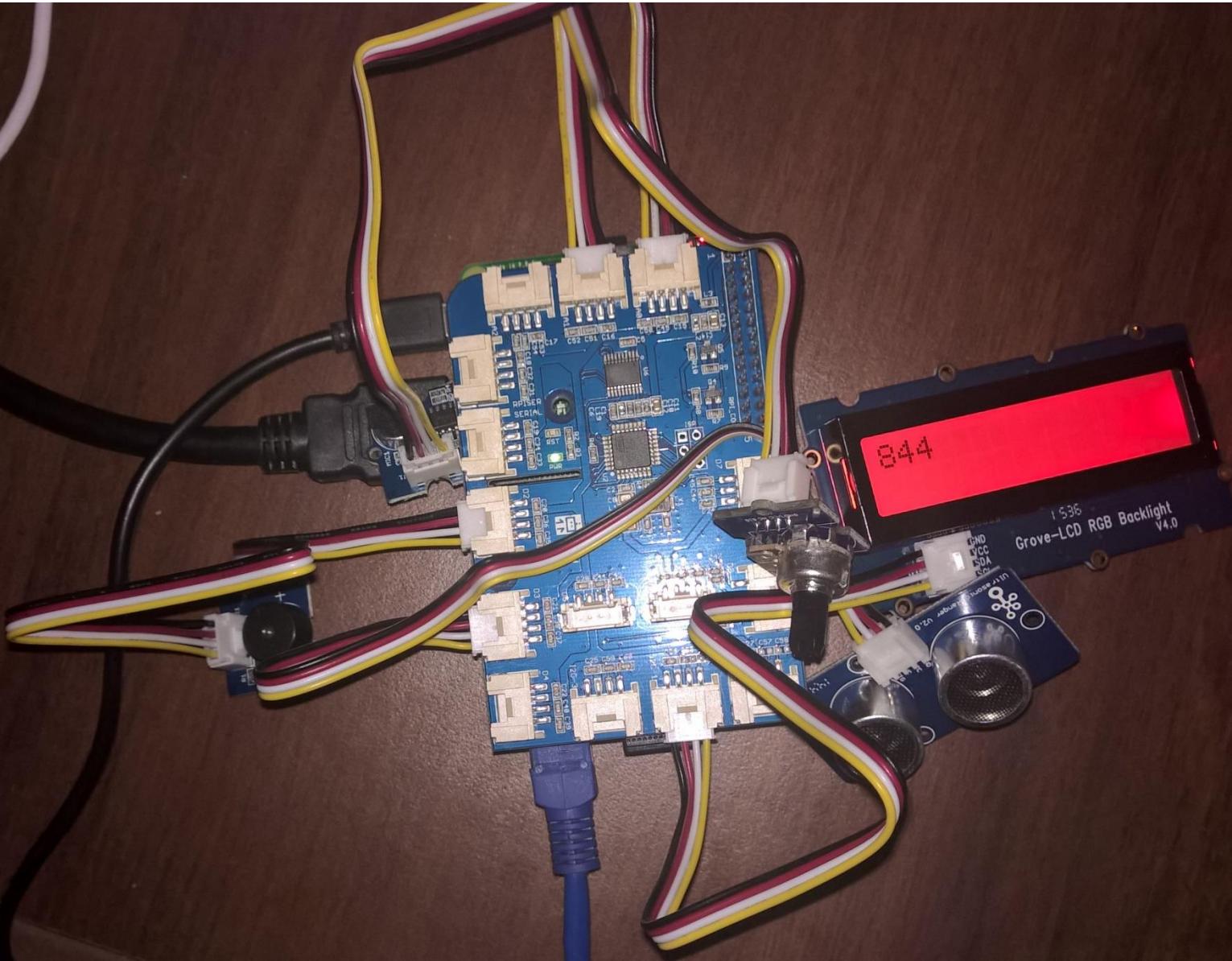
<https://github.com/DexterInd/GrovePi>

Folder: Software/CSharp

```
IRotaryAngleSensor m_rotary = DeviceFactory.Build.RotaryAngleSensor(Pin.AnalogPin2);
ILed led = DeviceFactory.Build.Led(Pin.DigitalPin5);
m_sound = DeviceFactory.Build.SoundSensor(Pin.AnalogPin0);
m_ut = DeviceFactory.Build.UltraSonicSensor(Pin.DigitalPin3);

angle = m_rotary.SensorValue();
led.AnalogWrite(Convert.ToByte(brightness));
m_lcd = DeviceFactory.Build.RgbLcdDisplay();
m_lcd.SetBacklightRgb(128, 0, 0);
var ut = m_ut.MeasureInCentimeters();
m_lcd.SetText(ut.ToString());
```

# A dalej – dygresja



# Adafruit - Microsoft IoT Pack for Raspberry Pi 2

## Components

(\*) MCP3008 - 8-Channel 10-Bit ADC

(\*) Assorted LEDs

(\*) Potentiometers

Tactile Switches

(\*) Resistors

Capacitor

## Sensors

(\*) Photocell

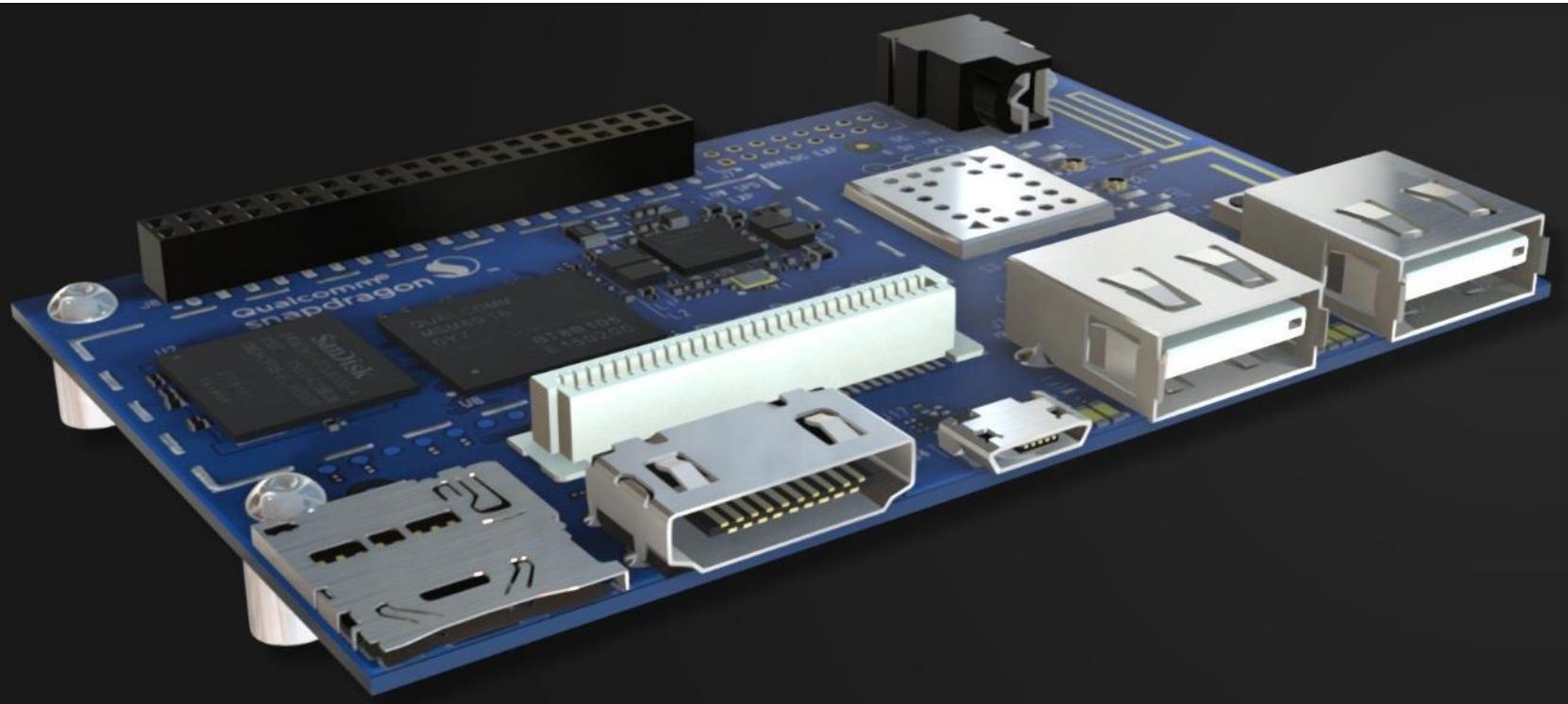
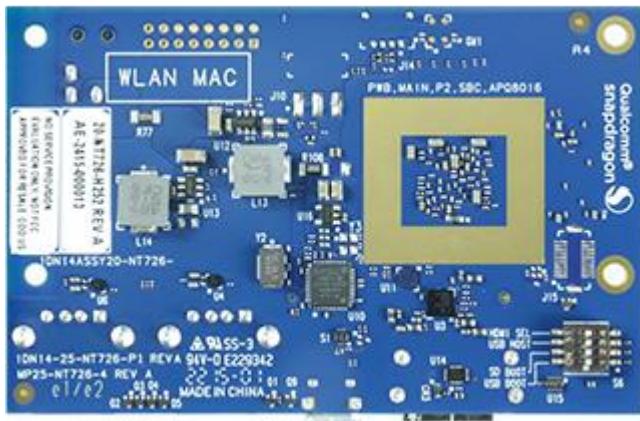
(\*) Adafruit BMP280 Temperature + Barometric Sensor

(\*) Adafruit TCS34725 Color Sensor

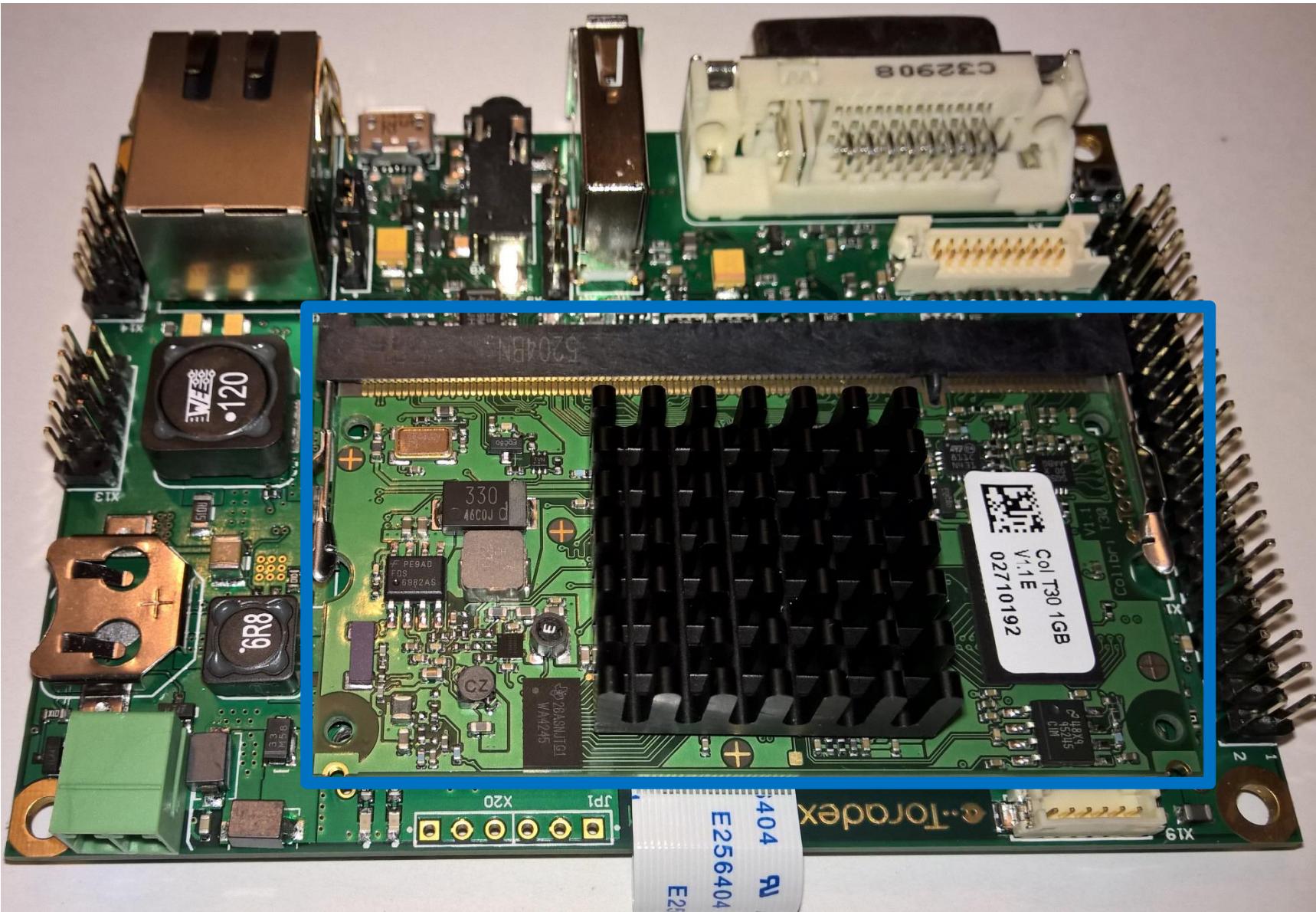


Bardziej niskopoziomowe,

# DragonBoard



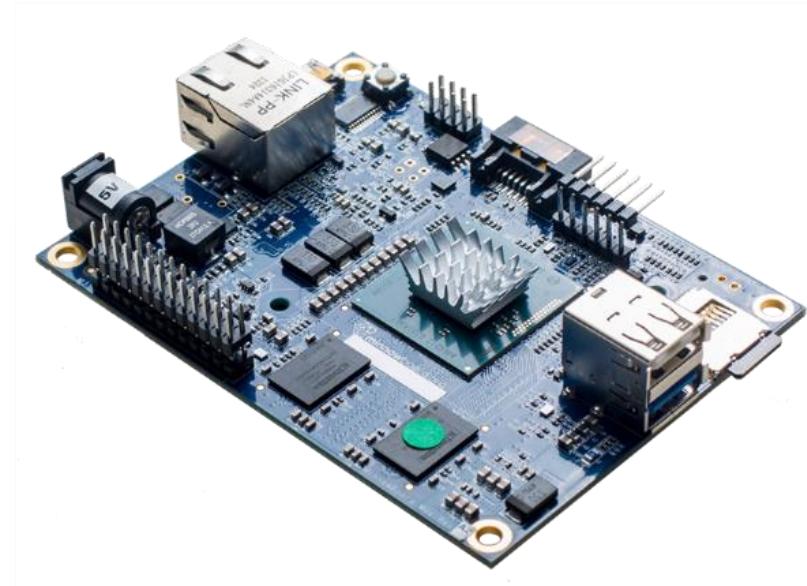
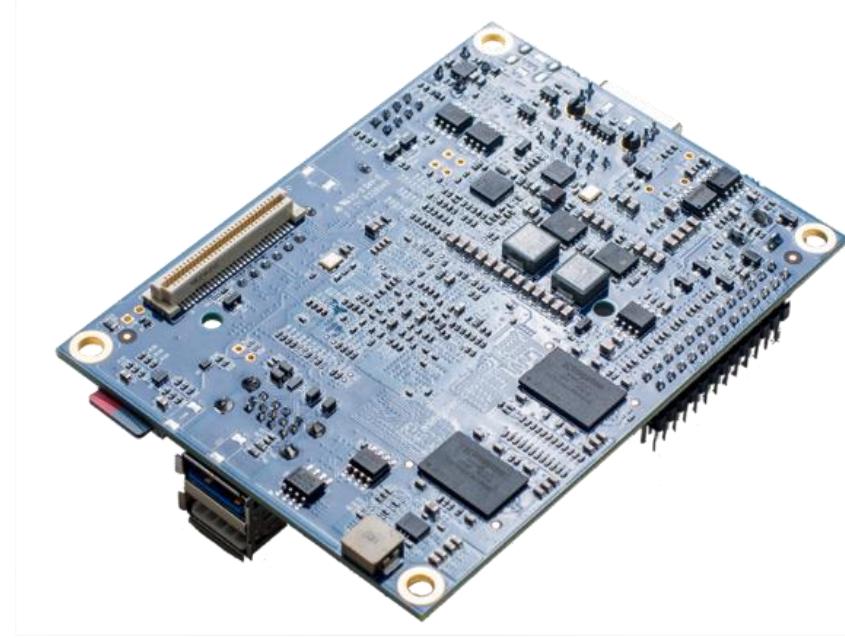
# Toradex



# Minnowboard Max

Płyta oparta o Intel® Atom™ E38xx (dwurdzeniowy, 1.33 Ghz),  
2GB RAM, Intel® HD Graphics, też  
PCI Express

Też może uruchomić „pełnego”  
Windows



# Raspberry Pi 3 (out of stock na razie)

RPi 2 +

1.2GHz 64-bit ARM Cortex-A53 CPU  
(10 x RPi1, 30% szybsze niż RPi2)

802.11n wireless LAN

Bluetooth 4.1



Na razie tylko w Windows Insider!

# Materiały na to spotkanie (GitHub)

<https://github.com/tkopacz/2016windowsiot-iothub-workshopV1>

Trochę psuje zabawę – bo wszystko działa, ale 😊

<https://github.com/tkopacz/2016windowsiot-iothub-genericsender>

Trochę dodatkowych narzędzi, nie wymagających urządzeń IoT

# Materiały inne

<https://github.com/Azure/azure-iot-sdks> - SDK do Azure IoT Hub dla różnych języków

<https://github.com/ms-iot/BusProviders.git> - Bus Providers dla PWM, ADC itp..

<https://github.com/pptierno/m2mqtt> - biblioteka MQTT dla .NET (w tym – UWP)

<http://ms-iot.github.io/content/en-US/win10/samples/HelloWorld.htm> - takie naprawdę Hello World

<http://ms-iot.github.io/content/en-US/win10/samples/BlinkyHeadless.htm> - Blinking Led

<https://github.com/ms-iot/adafruitsample> - przykłady lekcji dla Adafruit Starter Kit (bez IoT Hub), ale połączenia podobne

# Co budujemy

# Co budujemy?

## Urządzenia

12 „nadawców” sygnałów

Analogowy, cyfrowy, 2 pochodzące z I2C

12 „odbiorców” poleceń

Błyskanie diodą, aktualizacja UI, restart, ...

(kod symulatora jakby co...)

## Chmura

IoT Hub

Prywatny, per urządzenie

Event Hub: Jeden wspólny dla wszystkich

Stream Analytics

Prywatny

Jeden wspólny dla wszystkich

UI – jakiś (też konsola)

# Kroki do wykonania

1. Wgranie Windows IoT na kartę, uruchomienie urządzenia
2. Połączenie kabelków
3. 4 testowe aplikacje – czy dobrze kabelki są połączone
4. Założenie IoT Hub i Stream Analytics Job
5. Test działania IoT Hub
6. Główna aplikacja wysyłająca i odbierająca komunikaty z IoT Hub
7. Dodatkowe wyjście do Stream Analytics – centralny Event Hub
8. Wysyłanie komunikatów przy użyciu MQTT
9. Własna aplikacja do przetwarzania wiadomości

# Windows IoT - API (użytkowe)

## GpioPin

- Odczyt / zapis sygnału cyfrowego (1/0).

## I2cDevice

- Szyna do którejłączamy różne urządzenia
- 2 piny (+ masa), StandardMode: 100 kbit/s, FastMode: 1 mbit/s (są szybsze, ale to na razie to co jest dostępne)

## SpiDevice

- Szybsza szyna do którejłączamy różne urządzenia
- 4 piny (+ masa), prędkości większe niż I2C, do 100MHz, testy pokazują około 13 MB/s (~104 mbit/s)

## AdcChannel

- Przetwornik analogowo/cyfrowy (dołączony układ)

## Pwm

- Sygnał PWM (serwomechanizm, popiskiwania); Zakłada dołączone układy

# Raspberry Pi 2 - piny i ich znaczenie



3.3V PWR	1		2	5V PWR
I2C1 SDA	3		4	5V PWR
I2C1 SCL	5		6	GND
GPIO 4	7		8	Reserved
GND	9		10	Reserved
SPI1 CS0	11		12	GPIO 18
GPIO 27	13		14	GND
GPIO 22	15		16	GPIO 23
3.3V PWR	17		18	GPIO 24
SPI0 MOSI	19		20	GND
SPI0 MISO	21		22	GPIO 25
SPI0 SCLK	23		24	SPI0 CS0
GND	25		26	SPI0 CS1
Reserved	27		28	Reserved
GPIO 5	29		30	GND
GPIO 6	31		32	GPIO 12
GPIO 13	33		34	GND
SPI1 MISO	35		36	GPIO 16
GPIO 26	37		38	SPI1 MOSI
GND	39		40	SPI1 SCLK

GPIO

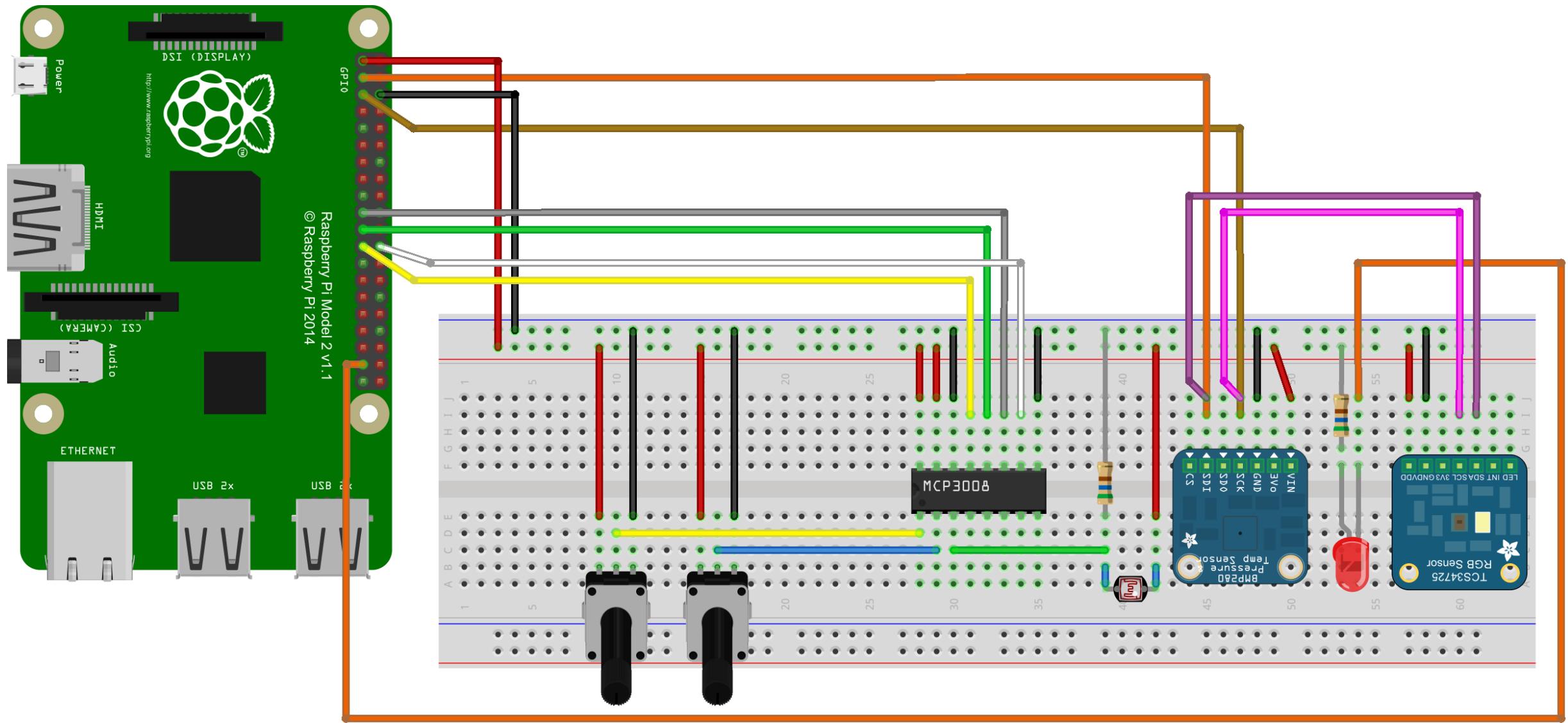
I2C: 3,5

SPI0: 19,21,23

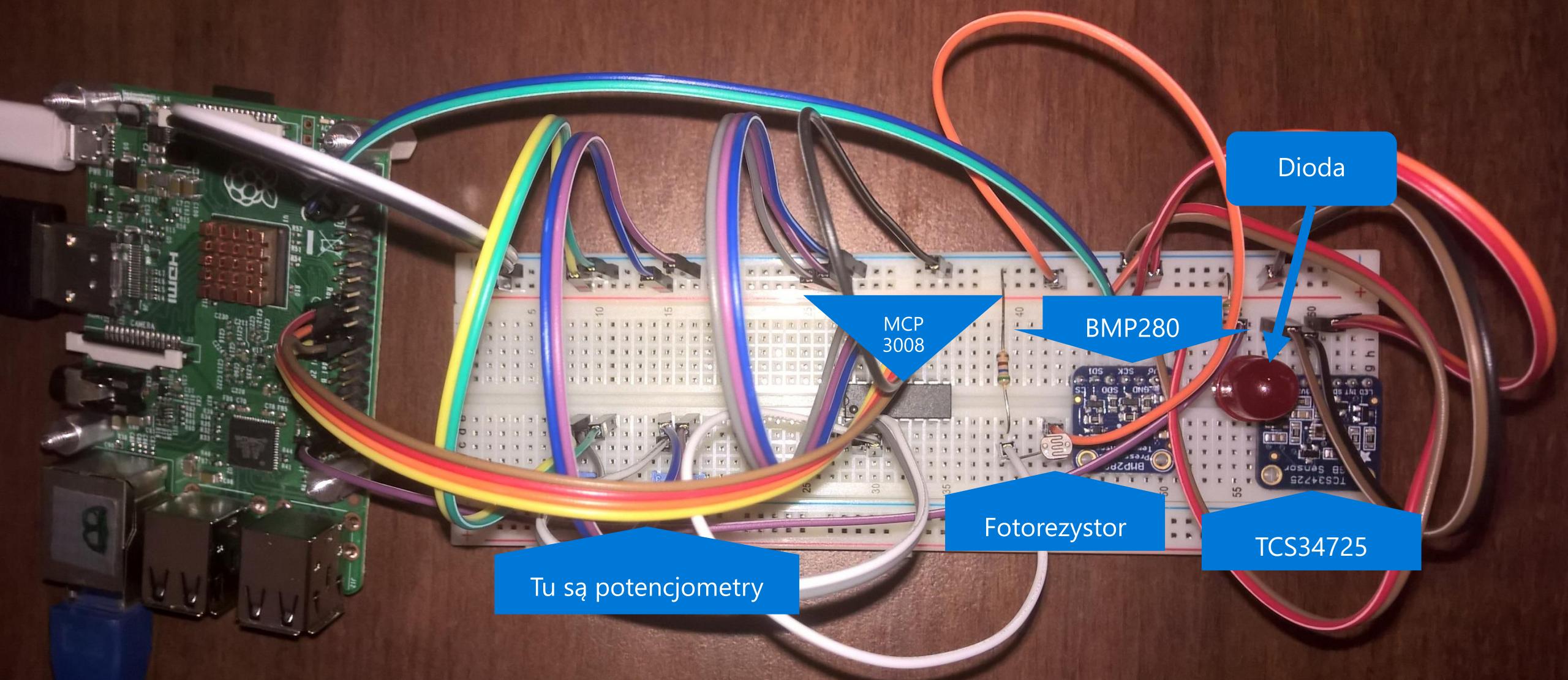
ew: 24,25

(SPI1 zajęte)

# Diagram (schemat połączeń)



# Sprzęt



SPI, programowanie –  
niskopoziomowe!

# Inicjalizacja SPI

```
private const string SPI_CONTROLLER_NAME = "SPI0";
private const Int32 SPI_CHIP_SELECT_LINE = 0; /* SPI CS0, pin 24 */
SpiDevice m_spiDev;

...
var settings = new SpiConnectionSettings(SPI_CHIP_SELECT_LINE);
settings.ClockFrequency = 3000000; // 3200000;3000000
settings.Mode = SpiMode.Mode0;

string spiAqs = SpiDevice.GetDeviceSelector(SPI_CONTROLLER_NAME);

var deviceInfo = await DeviceInformation.FindAllAsync(spiAqs);

m_spiDev = await SpiDevice.FromIdAsync(deviceInfo[0].Id, settings);
```

# Odczyt

```
byte[] m_readBuffer = new byte[2];
byte[] m_writeBufferCH0 = new byte[] { 0x68, 0x00}; //0 1 10 1 000
...
m_spiDev.TransferFullDuplex(m_writeBufferCH0, m_readBuffer);
int resCH0 = convertToInt(m_readBuffer);
```

---

```
byte[] m_writeBufferCH1 = new byte[] { 0x70, 0x00}; //0 1 11 0 000
...
m_spiDev.TransferFullDuplex(m_writeBufferCH1, m_readBuffer);
int resCH1 = convertToInt(m_readBuffer);
```

---

```
public int convertToInt(byte[] data) { //10 bitowe wejście, czyli 2 + 8 bitów
    int result = data[0] & 0x03; result <= 8; result += data[1];
    return result; //0 - 1023
}
```

# Dygresja: Wiele urządzeń

Raspberry PI 2 to tzw. SPI Master

Konieczny kanał Chip Select dla każdego urządzenia

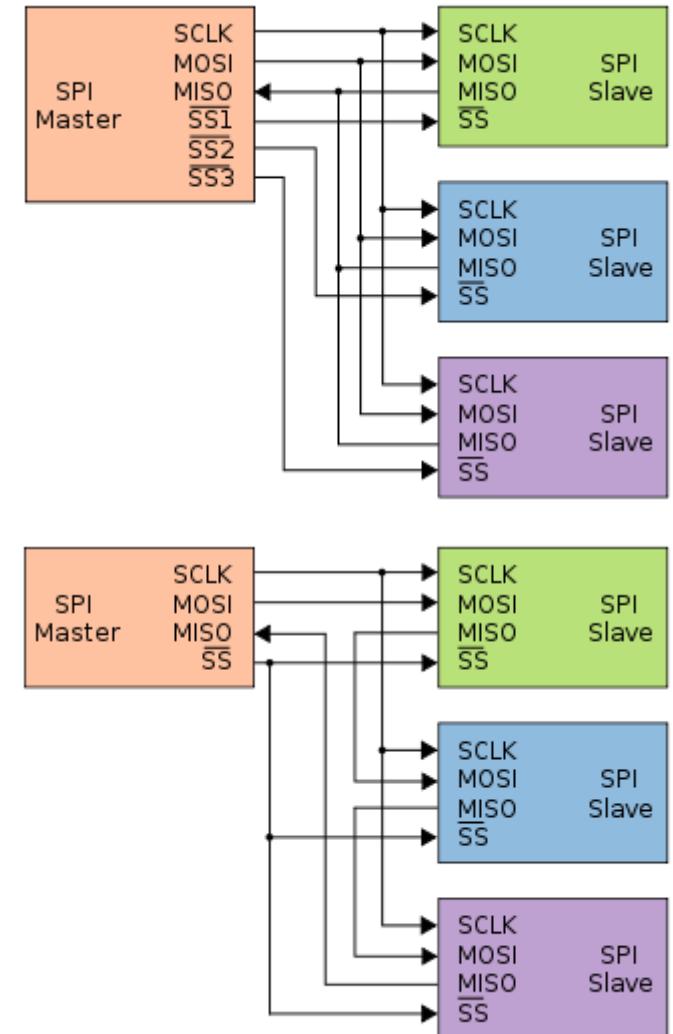
Chyba, że jak w dolnym rysunku, urządzenia ze sobą współpracują

Raspberry PI 2 ma 2 kanały „Chip Select”

W naszym przykładzie stała SPI\_CHIP\_SELECT\_LINE, wartość 0 lub 1 co odpowiada pinom 24 lub 26.

Inne interfejsy (np. I<sup>2</sup>C) mają znacznie wygodniejsze mechanizmy łączenia urządzeń w łańcuch

Ale są wolniejsze



I<sup>2</sup>C, programowanie –  
niskopoziomowe!

# Protokół I<sup>2</sup>C

Dobry opis: <https://learn.sparkfun.com/tutorials/i2c>

Węzeł **Master** generuje cykle zegara synchronizujące komunikację. Inicjuje komunikację do **slave**. Węzeł **Slave** pracuje zgodnie z otrzymanymi cyklami zegara. Ma adres (liczba 7-bitowa). Gdy otrzyma komunikat (bajt) może zwrotnie wysłać odpowiedź. Ale **master** musi wydać polecenie „Read”.

(W skrócie) sposób komunikacji:

**Master** wysyła do danego **slave** kod sterujący a **slave** wykonuje daną operację opcjonalnie dodatkowo zwraca jakieś wartości do Master.

Czyli, jeżeli 2 urządzenia **slave** chcą coś do siebie wysłać, musi koordynować to **master**.

Raspberry PI 2, MinnowBoard MAX, .NET Micro Framework to urządzenia typu master

Są przykłady softwareowej realizacji I2C

# Protokół I2C - z punktu widzenia Windows IoT:

Inicjalizacja:

```
string deviceSelector = I2cDevice.GetDeviceSelector();
var i2cDeviceControllers = await DeviceInformation.FindAllAsync(deviceSelector);
var i2cSettings = new I2cConnectionSettings(I2C_ADDR_MCP23008);
i2cMCP23008 = await I2cDevice.FromIdAsync(i2cDeviceControllers[0].Id, i2cSettings);
```

Wysłanie polecenia:

```
i2cMCP23008.Write(new byte[] { MCP23008_OLAT, olatRegister });
```

Wysłanie polecenia (kod MCP23008\_OLAT) i odebranie wyniku do bufora i2CReadBuffer:

```
i2cMCP23008.WriteRead(new byte[] { MCP23008_OLAT }, i2CReadBuffer);
```

Odebranie wyniku, gdy możemy otrzymać niepełną informację:

```
var result = i2cArduino.ReadPartial(rbuffer);
```

# W naszym przypadku - SPI

```
m_adc = (await AdcController.GetControllersAsync(AdcMcp3008Provider.GetAdcProvider()))[0];  
m_adc.OpenChannel(0).ReadRatio();
```

```
AdcMcp3008Provider  
GetControllers -> AdcMcp3008ControllerProvider
```

```
public int ReadValue(int channelNumber) {  
    byte command = (byte)channelNumber;  
    if (channelMode == ProviderAdcChannelMode.SingleEnded)  
        command |= MCP3008_SingleEnded;  
    command <<= 4;  
    byte[] commandBuf = new byte[] { 0x01, command, 0x00 };  
    byte[] readBuf = new byte[] { 0x00, 0x00, 0x00 };  
spiController.TransferFullDuplex(commandBuf, readBuf);  
    int sample = readBuf[2] + ((readBuf[1] & 0x03) << 8);  
    return sample; }
```

# W naszym przypadku – I<sup>2</sup>C

BMP280: <http://www.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf>

```
const byte BMP280_Address = 0x77;  
private async Task begin() ...  
private byte ReadByte(byte register) {  
    byte value = 0; byte[] writeBuffer = new byte[] { 0x00 }; byte[] readBuffer = new byte[] { 0x00 };  
    writeBuffer[0] = register;  
    bmp280.WriteRead(writeBuffer, readBuffer);  
    value = readBuffer[0]; return value; }
```

TCS34725: <http://www.adafruit.com/datasheets/TCS34725.pdf>

```
const byte TCS34725_Address = 0x29;  
private async Task begin() ...  
public async Task<ColorData> GetRawDataAsync() {  
    WriteBuffer[0] = TCS34725_CDATAL | TCS34725_COMMAND_BIT;  
    colorSensor.WriteRead(WriteBuffer, ReadBuffer);  
    colorData.Clear = ColorFromBuffer(ReadBuffer);
```

# Uwaga konstrukcyjna

# Źle (zamienione SDA / SCL, MOSI/MISO, ...)

Exception thrown: 'System.IO.FileNotFoundException' in mscorlib.ni.dll

WinRT information: Slave address was not acknowledged.

Exception thrown: 'System.IO.FileNotFoundException' in mscorlib.ni.dll

WinRT information: Slave address was not acknowledged.

Exception thrown: 'System.IO.FileNotFoundException' in mscorlib.ni.dll

WinRT information: Slave address was not acknowledged.

Dygresja – rola biernych elementów

# Bierne elementy

Rezystor (opornik, potencjometr)

Kondensator (potencjalnie można by użyć)

(cewka (dławik) – tu nie używane; też potencjalnie...)

(memristor – tu nie używane)

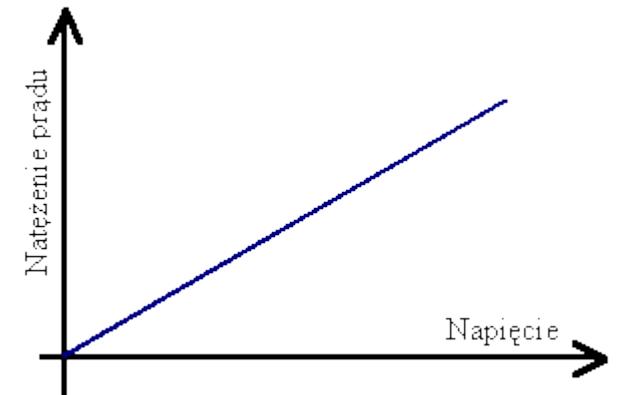
Prawo Ohma:

$$V = R * A$$

V – (U) napięcie

R – opór

A – (I) natężenie prądu



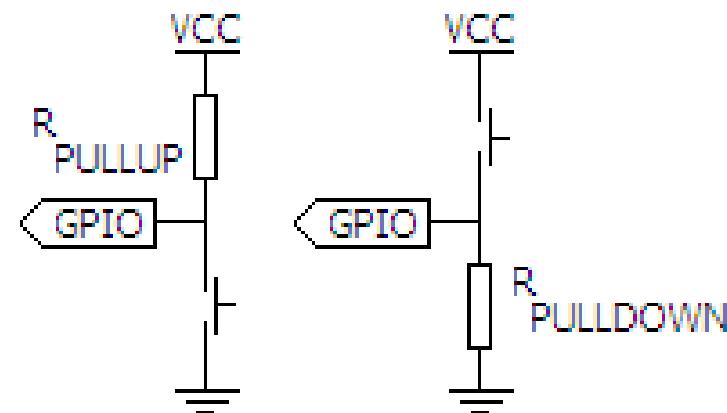
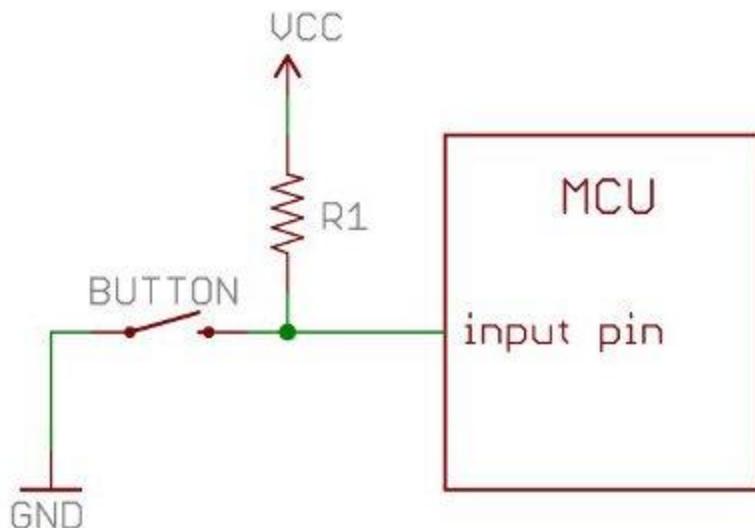
# GPIO: Pull-Up, Pull-Down

Zapewnia że PIN bez połączenia będzie w określonym stanie (0 lub 1)

Za: <https://learn.sparkfun.com/tutorials/pull-up-resistors>

Pull-Up: stan 1 gdy przycisk nie jest naciśnięty

Pull-Down: stan 0 gdy przycisk nie jest naciśnięty



# Rezystor (opornik)

A

GPIO 1 - dioda gaśnie

Ochrona przed maksymalnym prądem.

[http://kalkulator.majsterkowicza.pl/oblicz/rezystor do LED](http://kalkulator.majsterkowicza.pl/oblicz/rezystor%20do%20LED)

„Nadmiar” odłoży się na oporniku

Uszkadza za duży prąd, rzadko napięcie

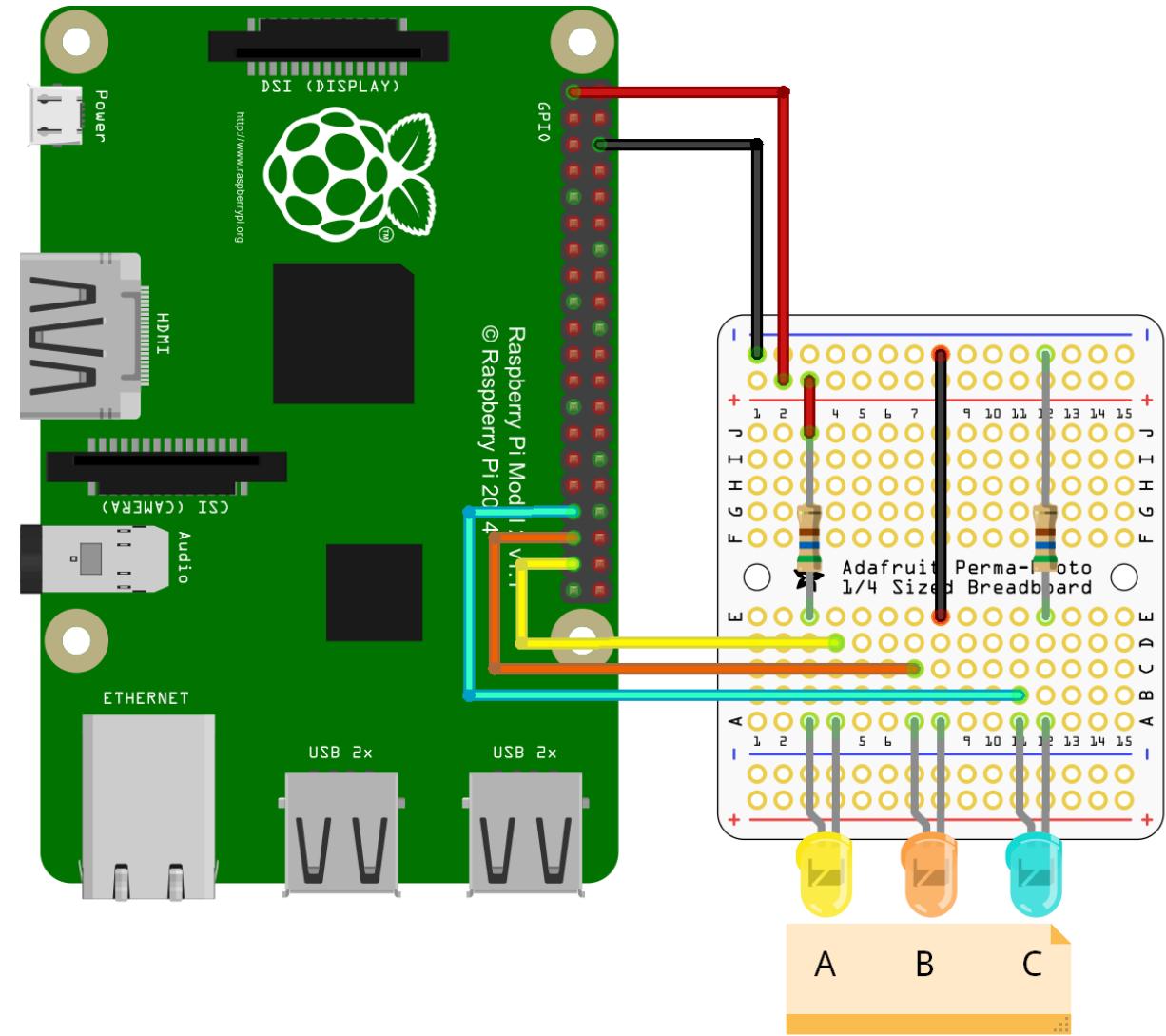
B

GPIO 1 – dioda się zapala

Brak zabezpieczeń

C

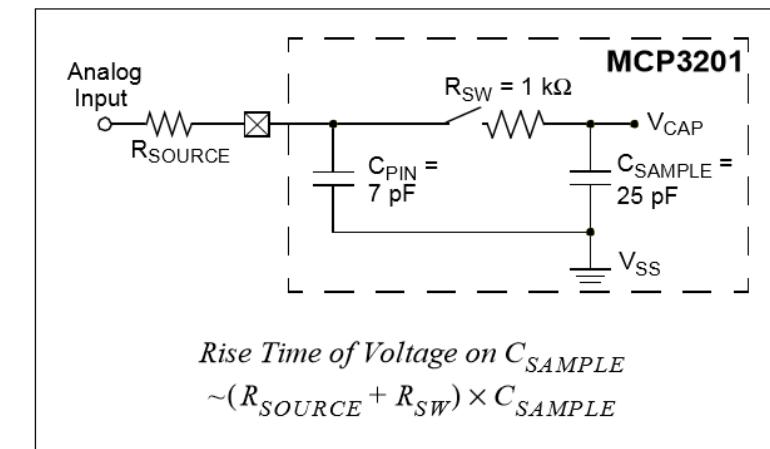
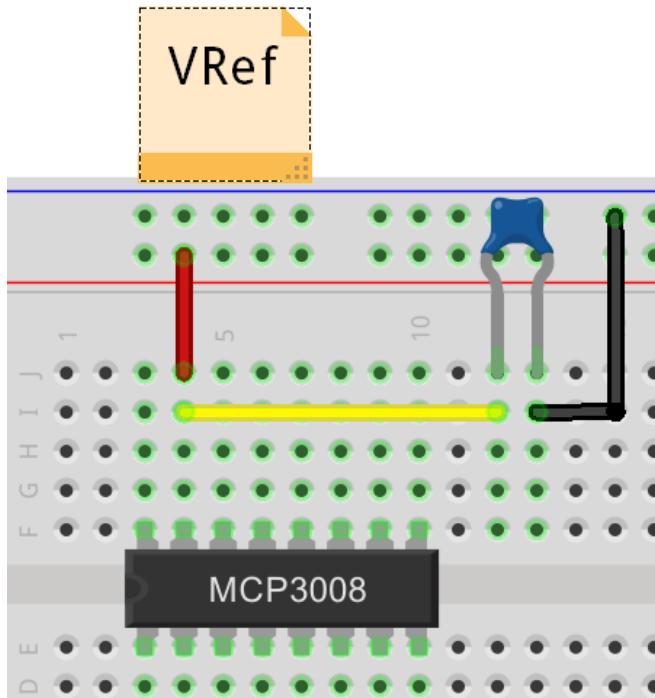
GPIO 1 – dioda się zapala



# Kondensator

Opcjonalnie: filtrowanie napięć

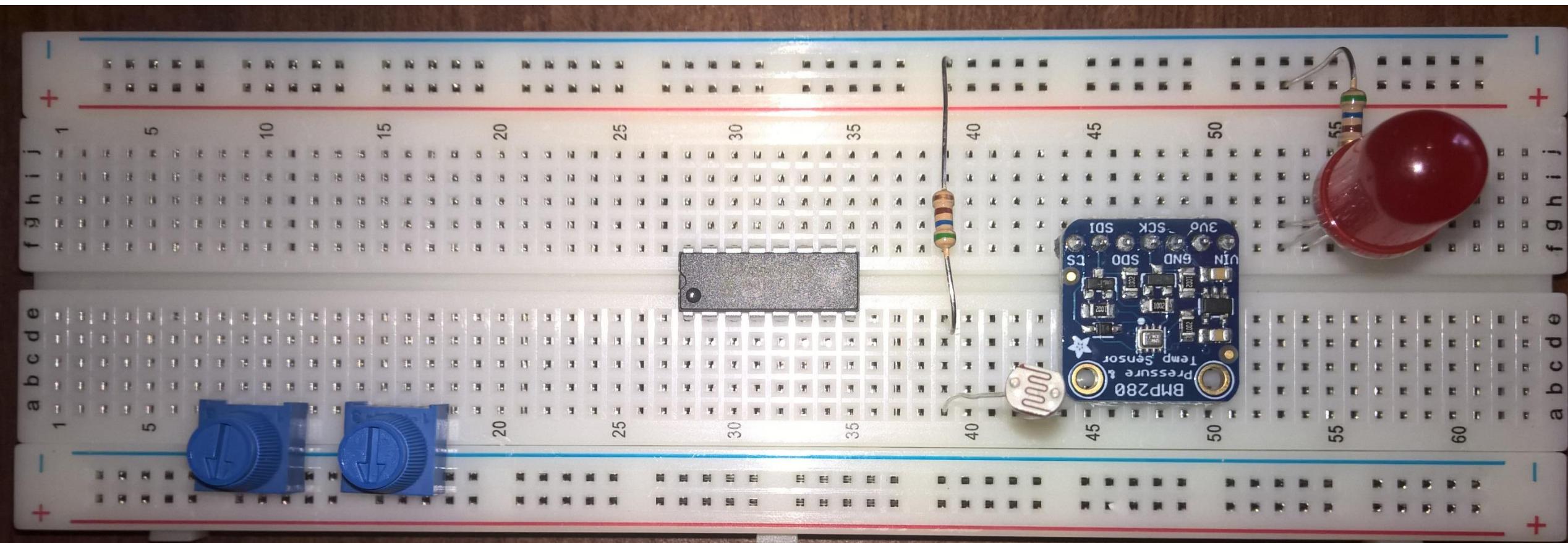
Podłączany zwykle dla  $V_{ref}$



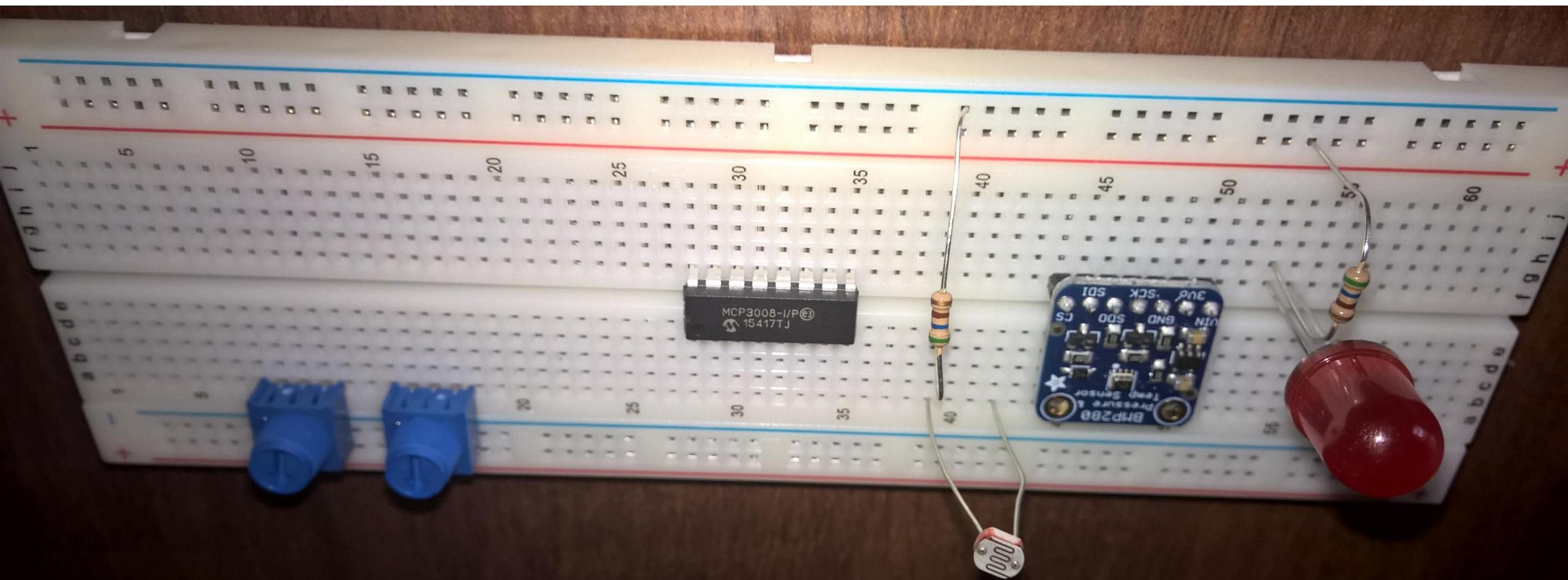
**FIGURE 3:** The model of the input stage of the MCP320X ADC can be reduced to a switch resistance and sample capacitor.

# Składanie urządzenia

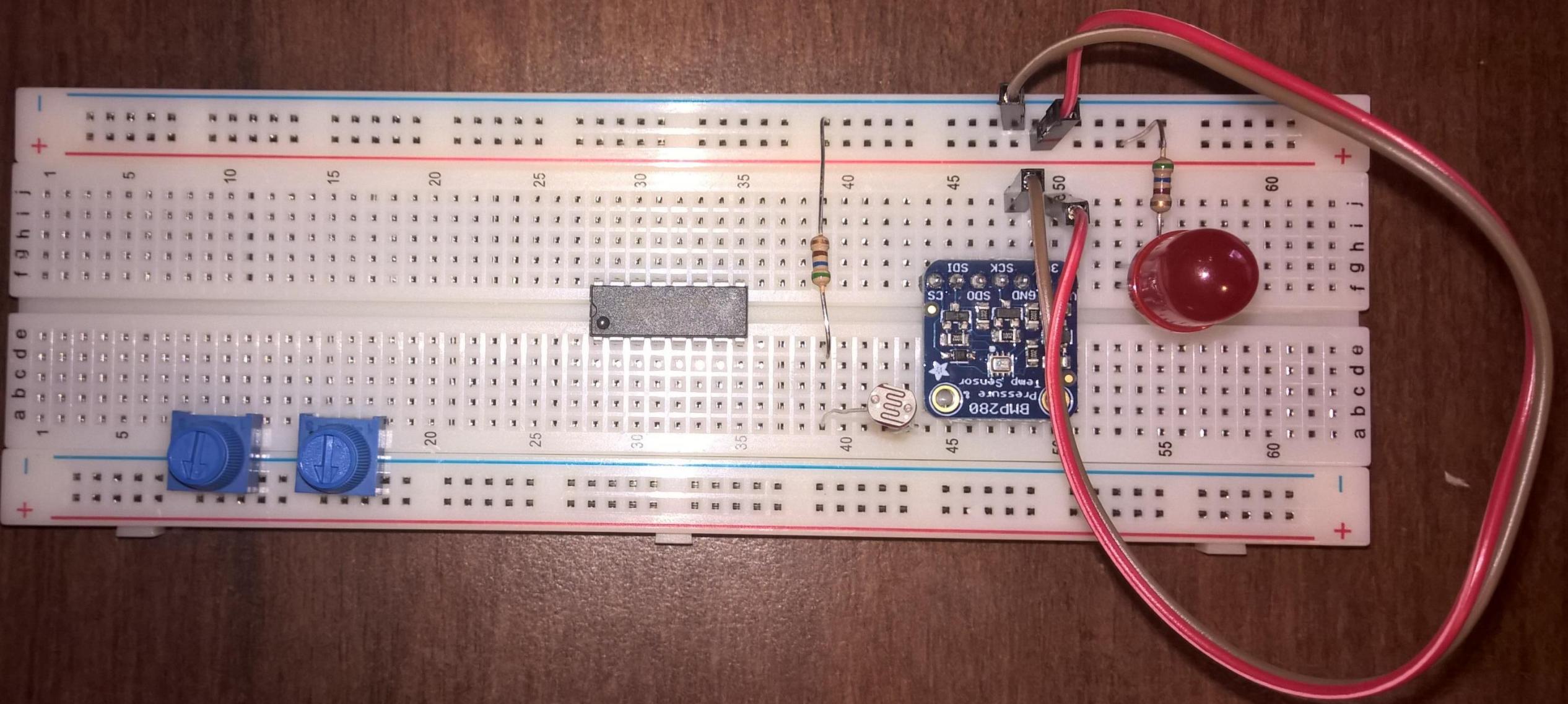
# Krok 01



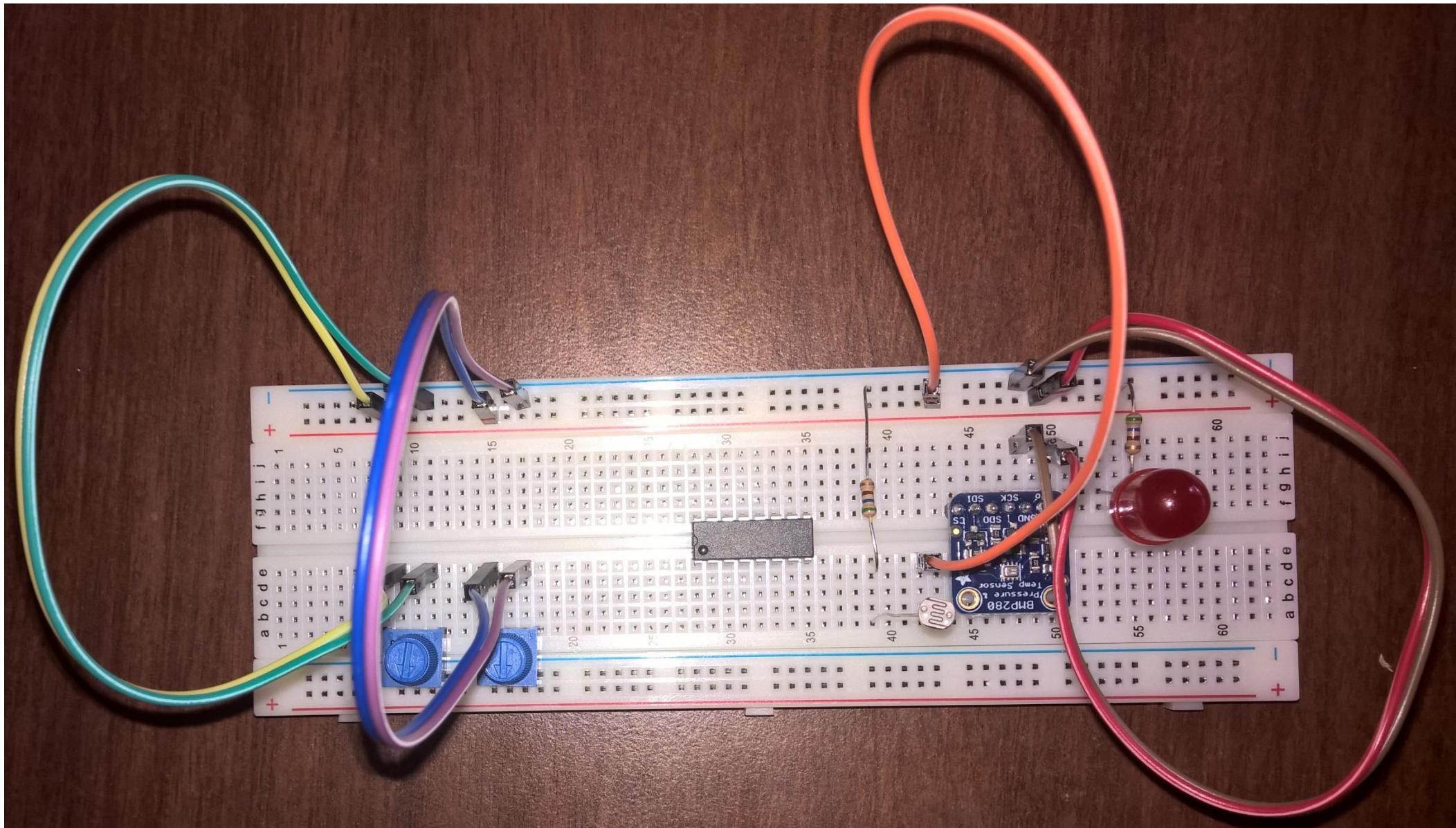
# Krok 01



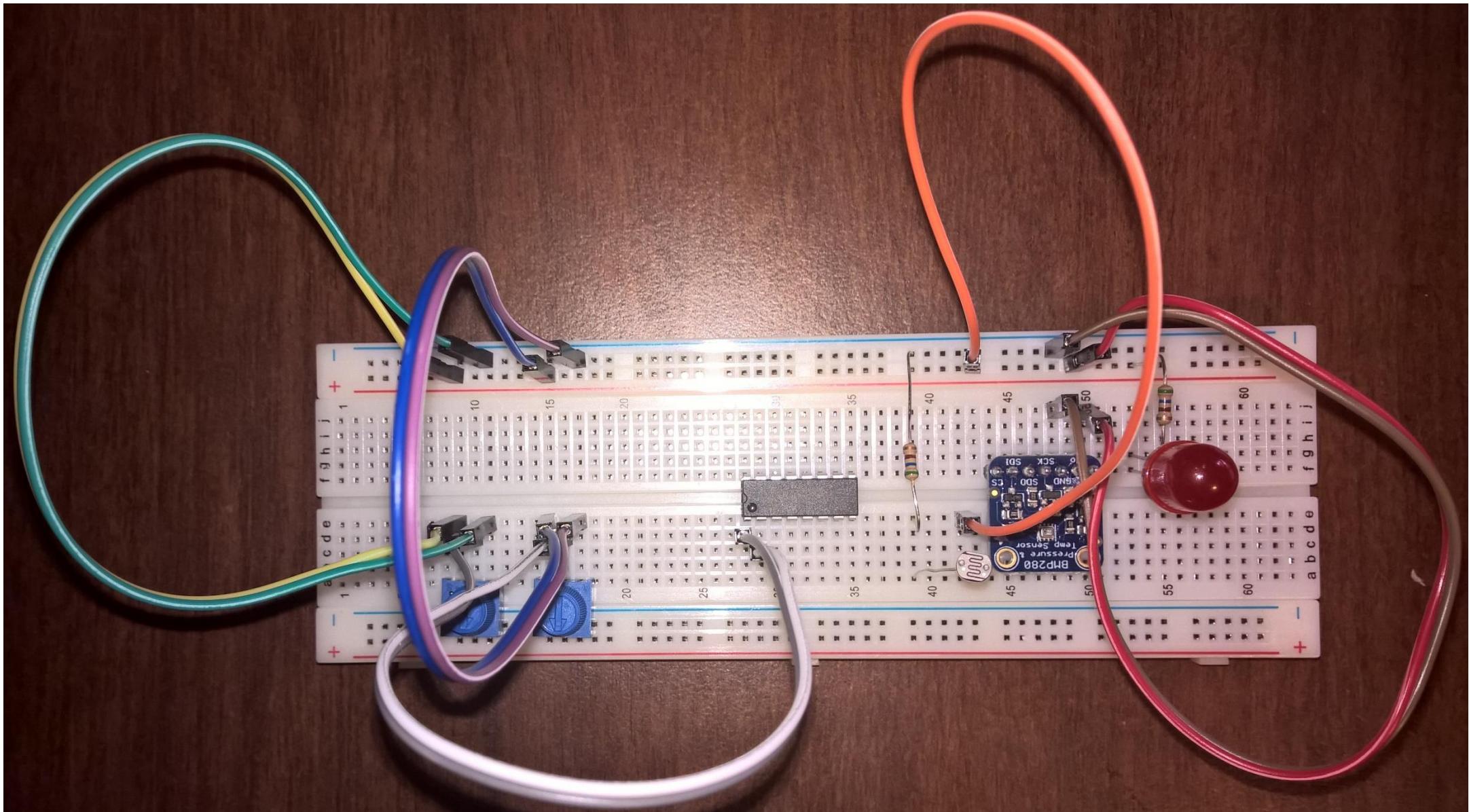
# Krok 02



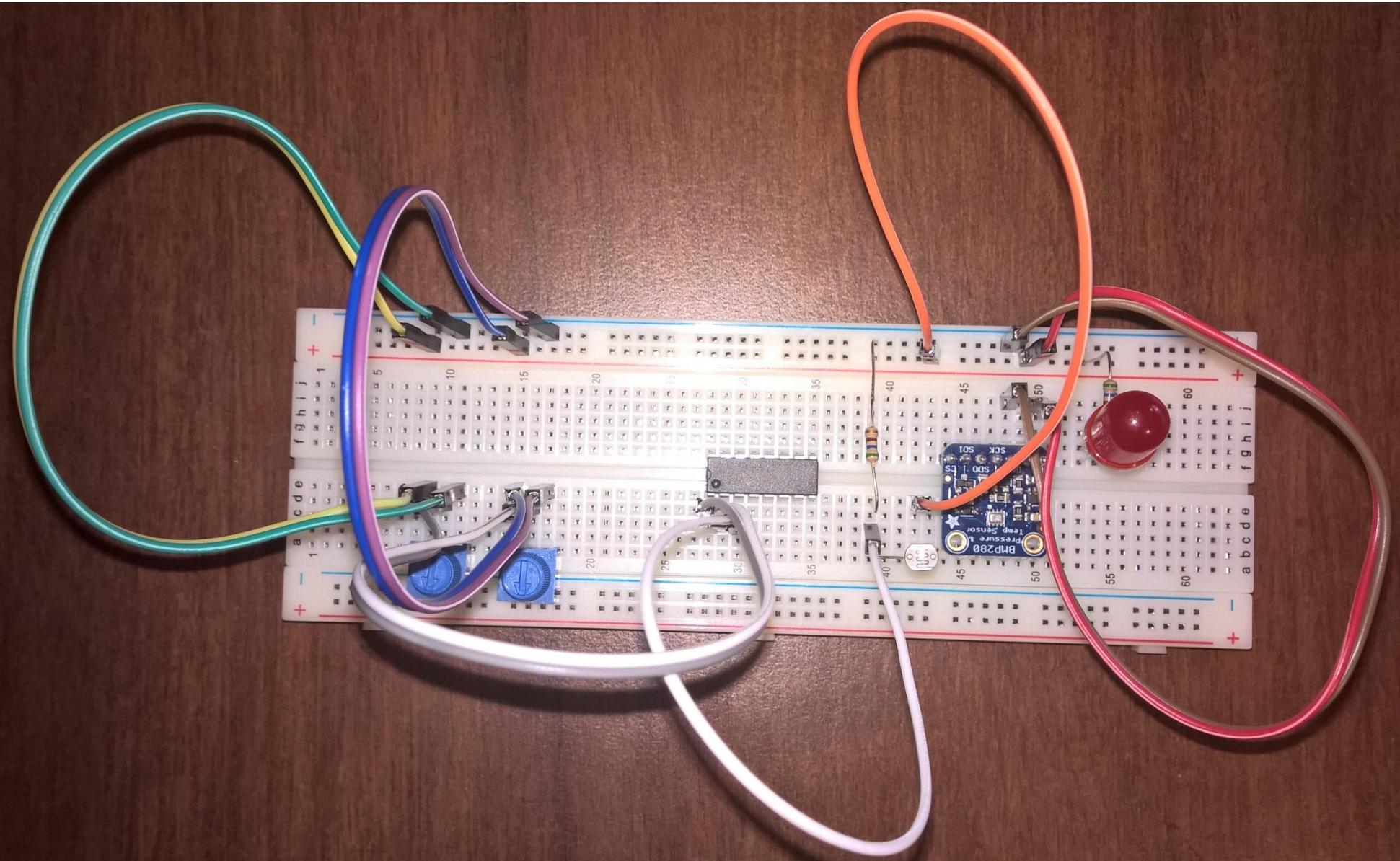
# Krok 03



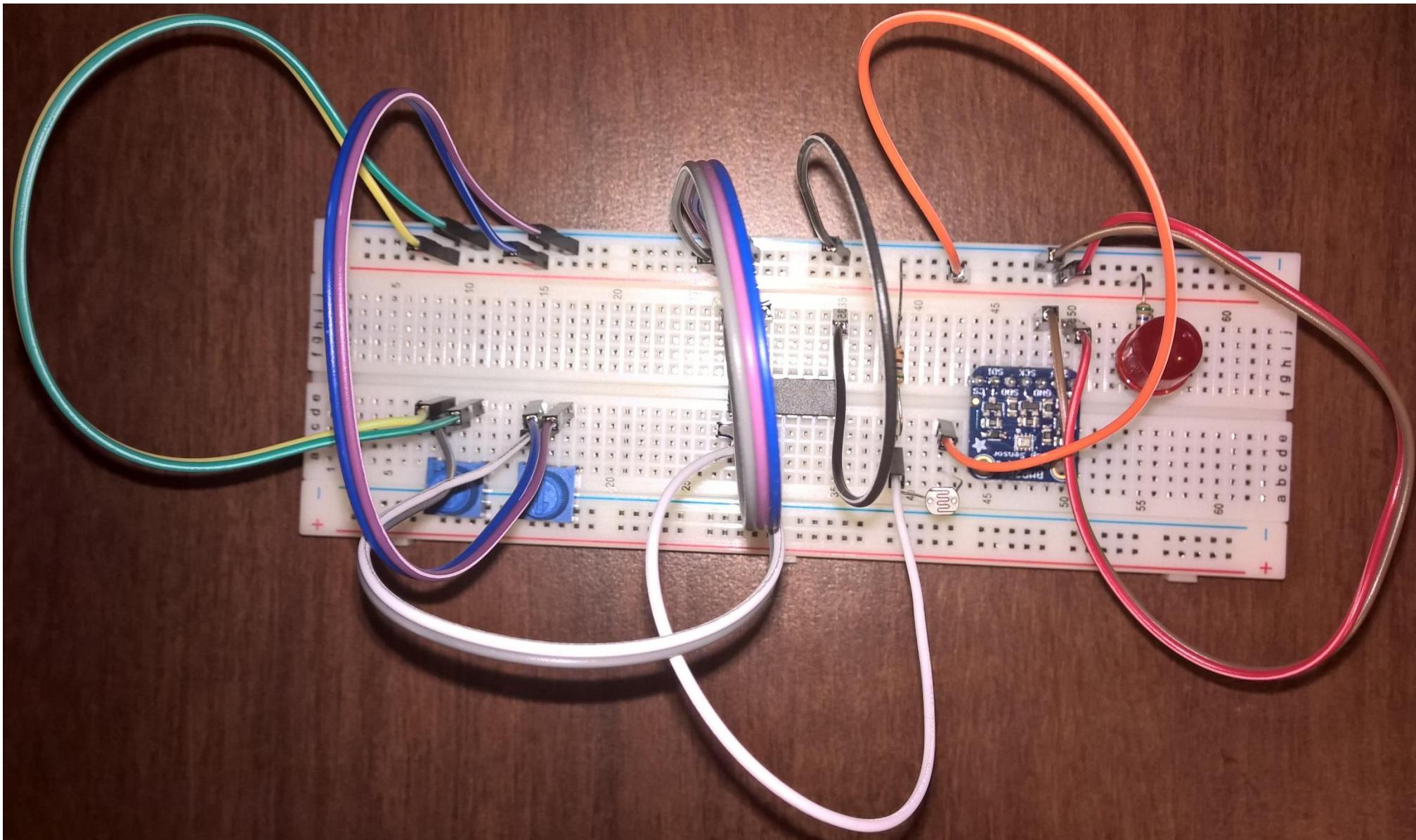
# Krok 04



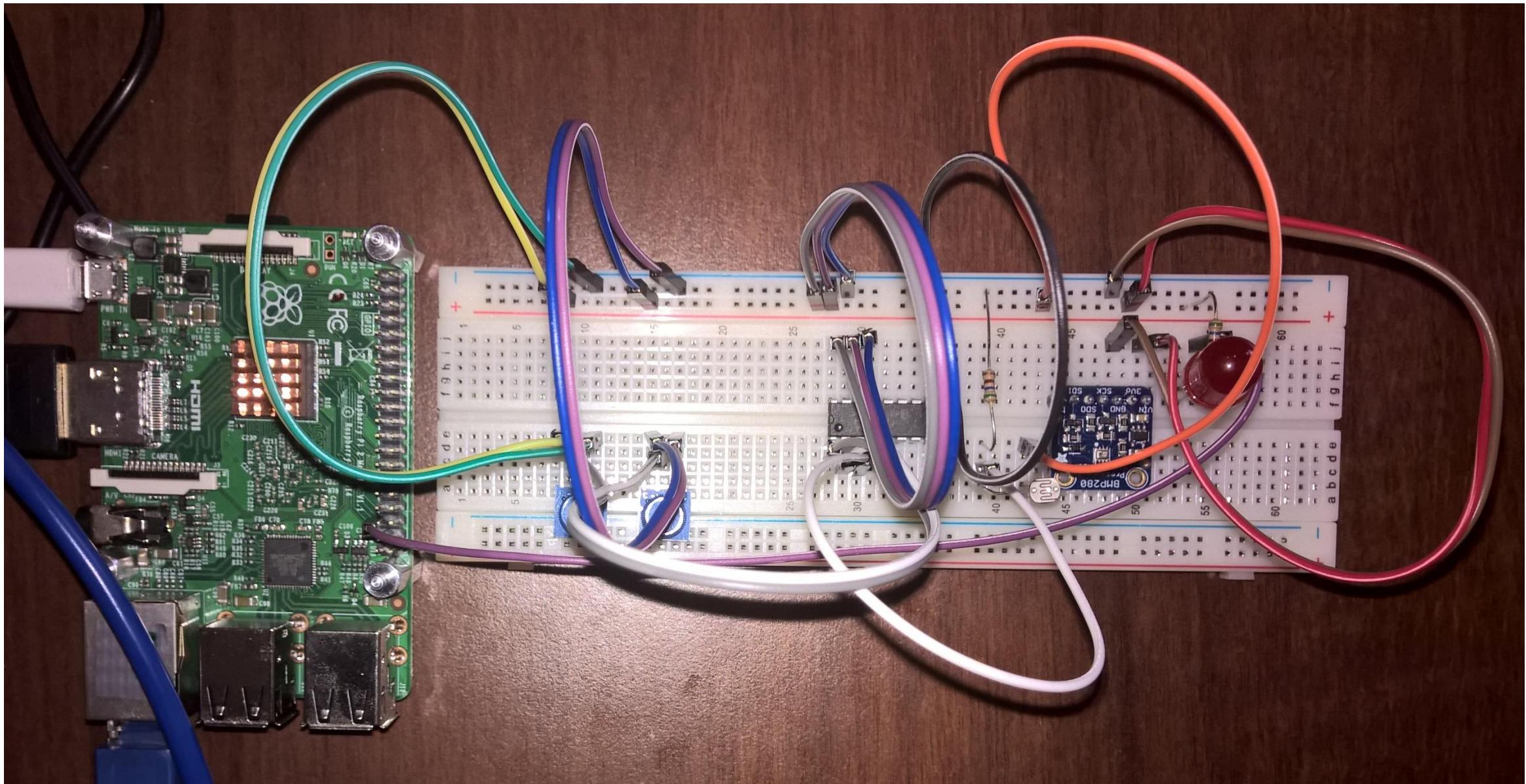
# Krok 05



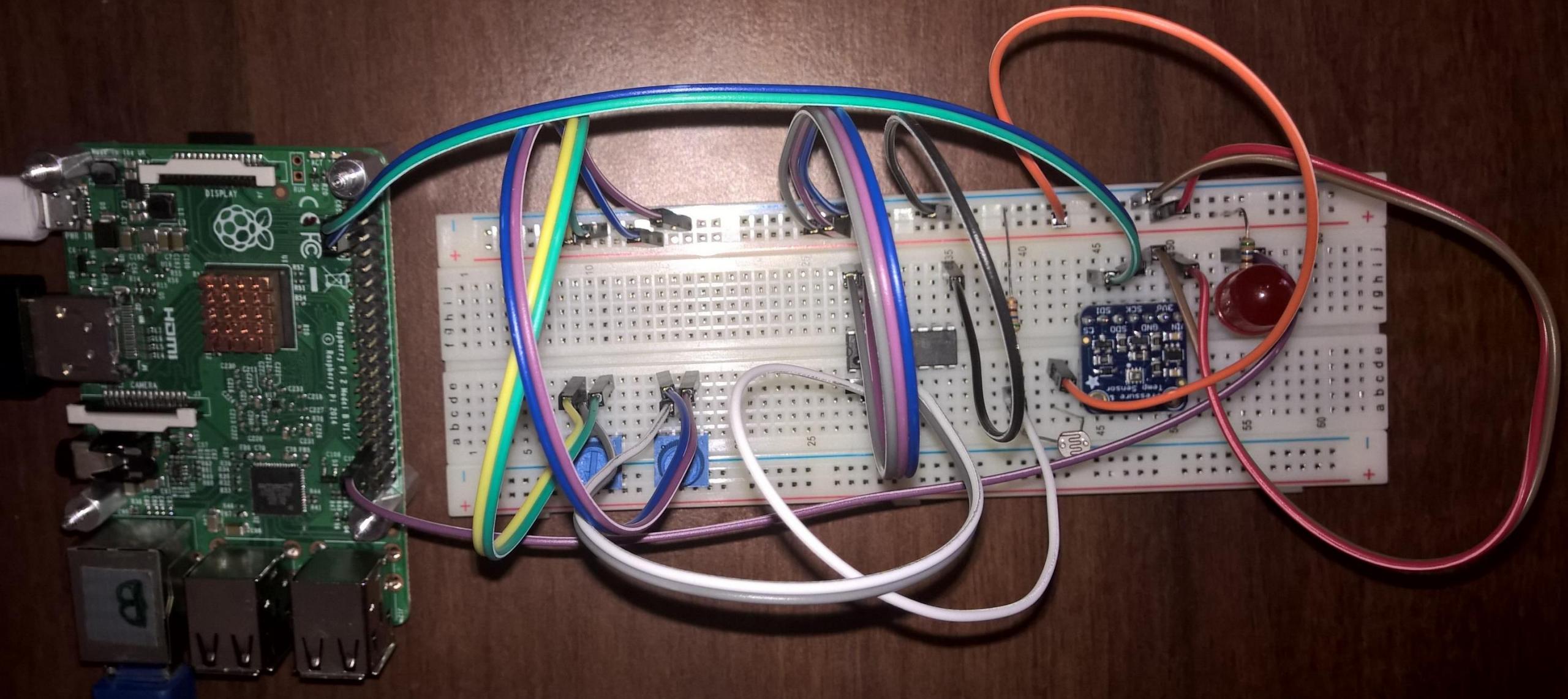
# Krok 06



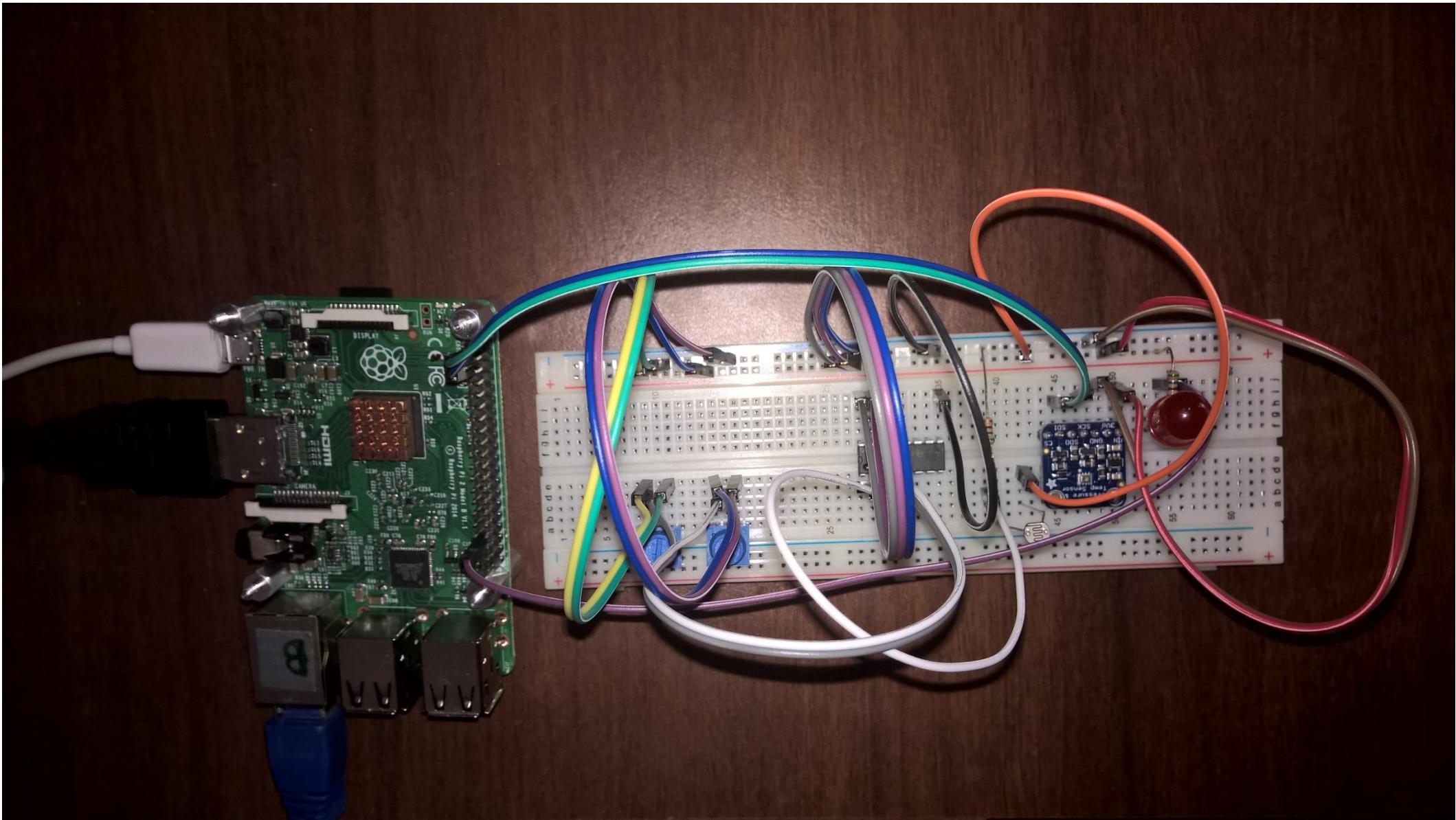
# Krok 07



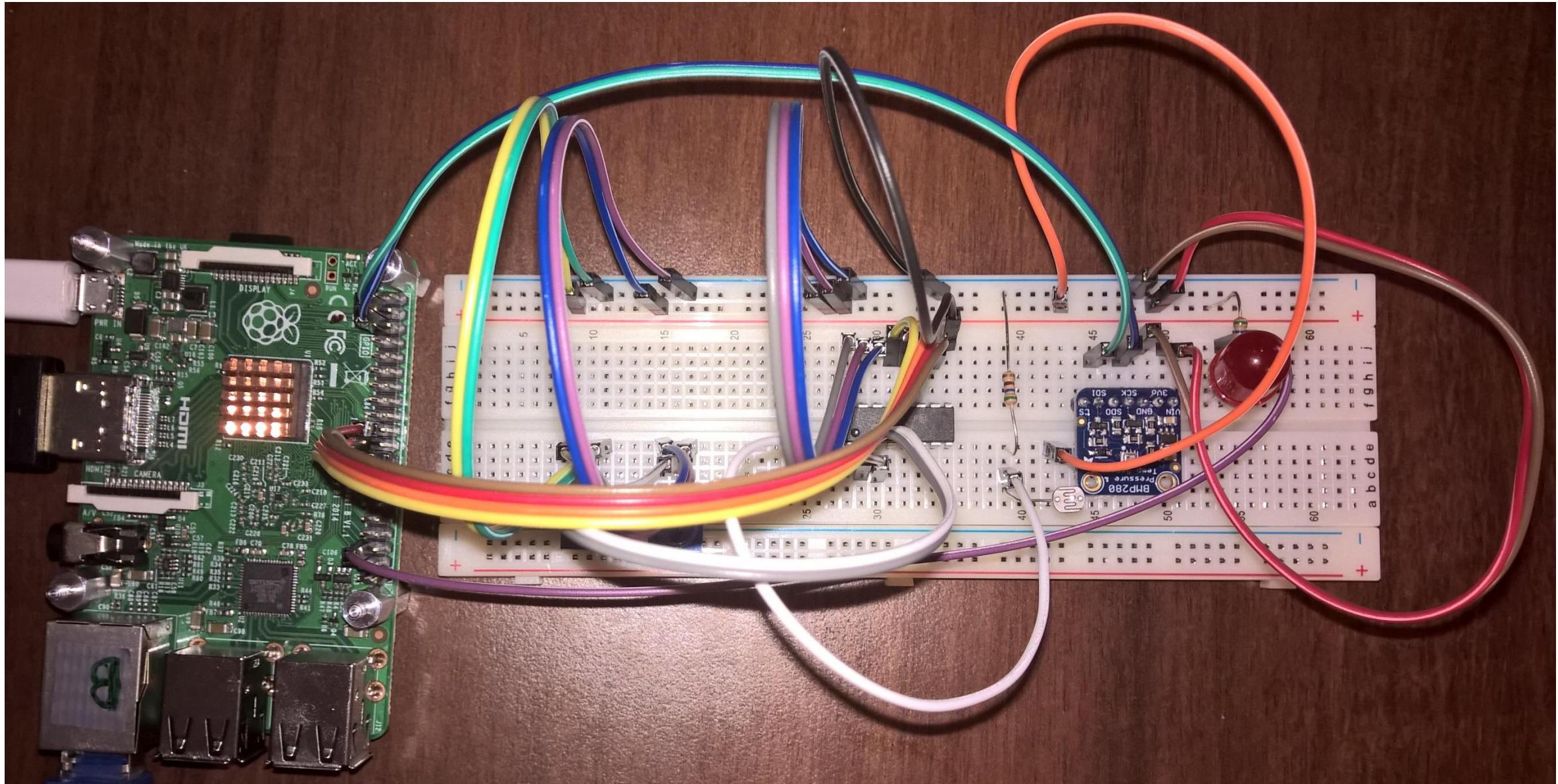
# Krok 07



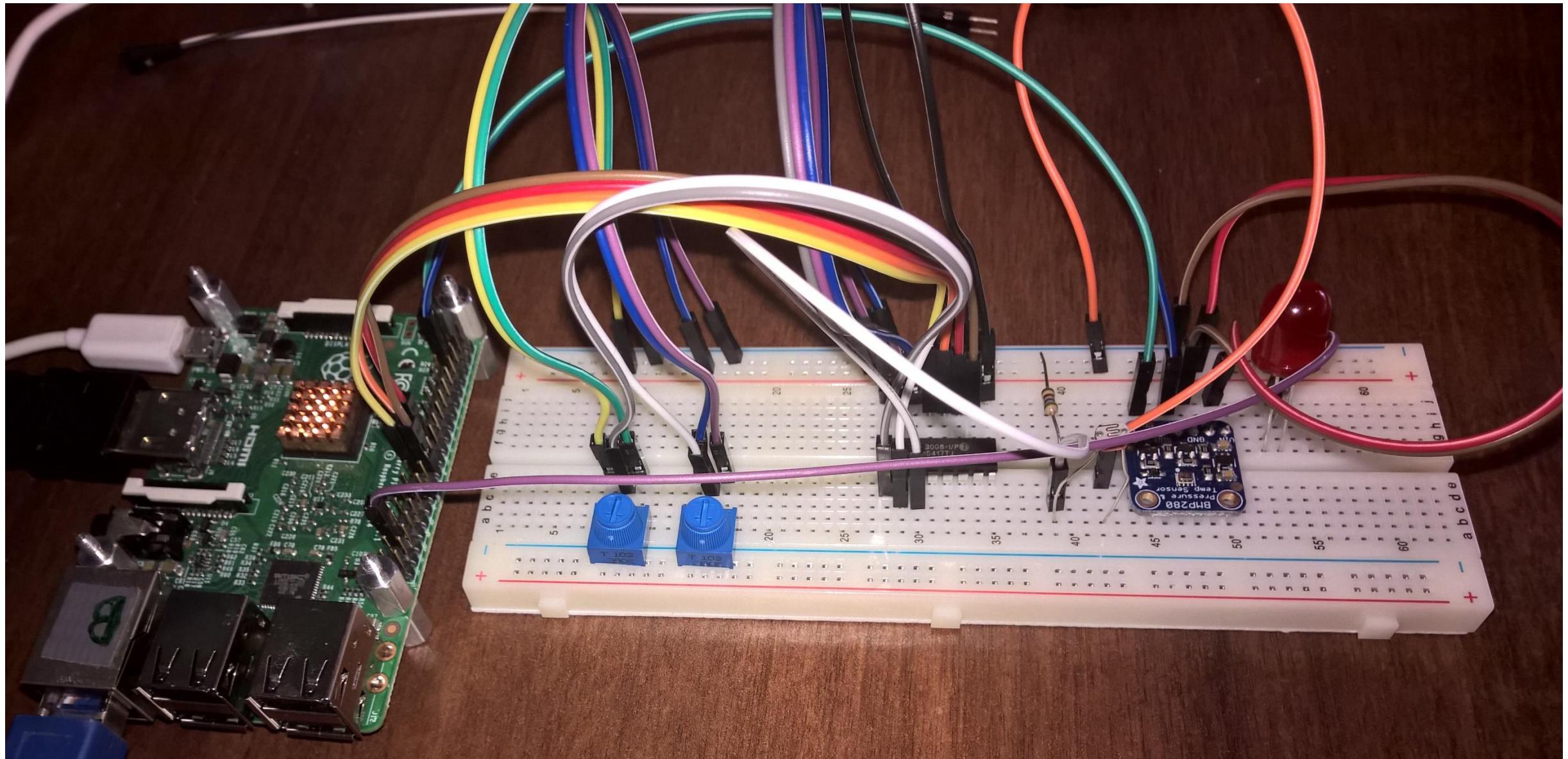
# Krok 08



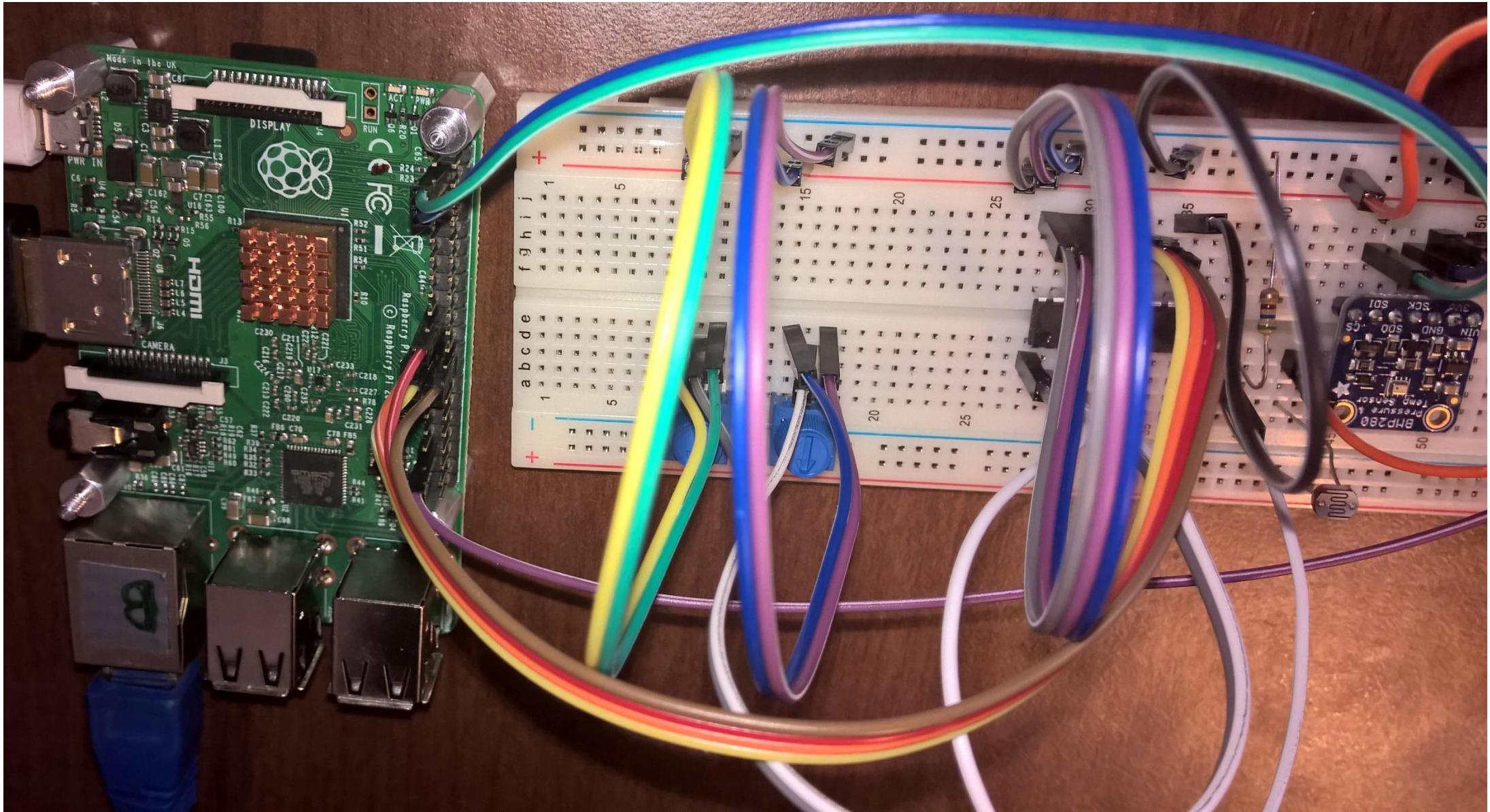
# Krok 09



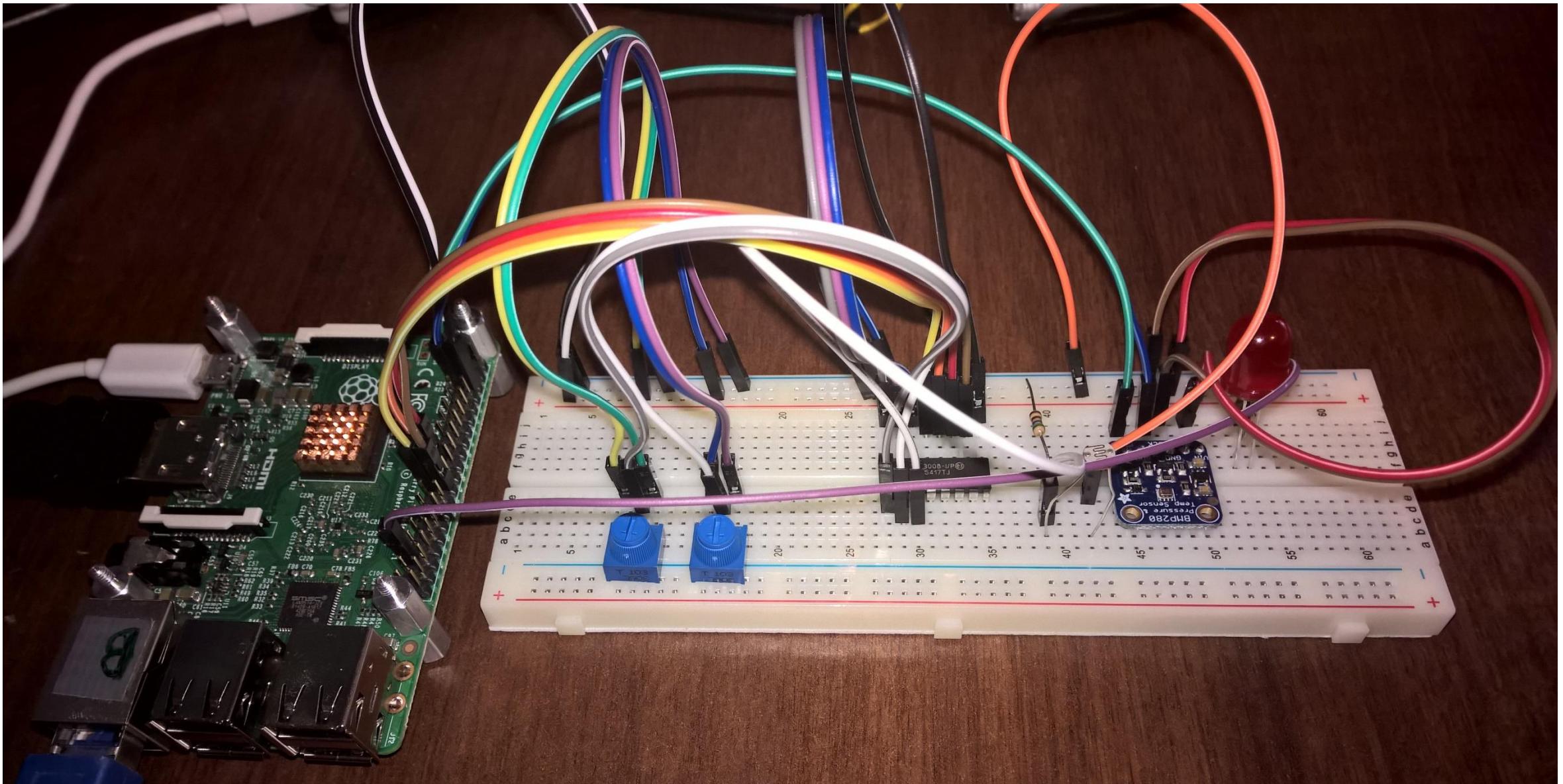
# Krok 09



# Krok 09



# Krok 10 (zasilanie)



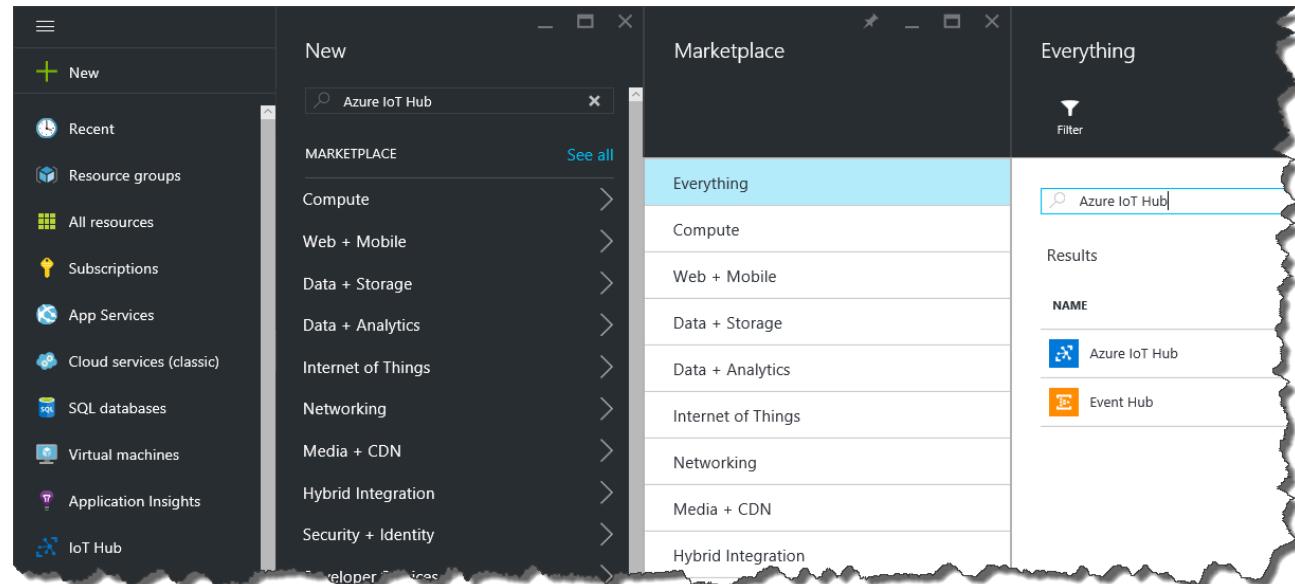
# Azure IoT Hub

# Co potrzebne?

Założony Azure IoT Hub

Skopiowane hasła (*iothubowner, service, device*)

Założenie Stream Analytics Job



# Rejestracja urządzenia

# Device Explorer

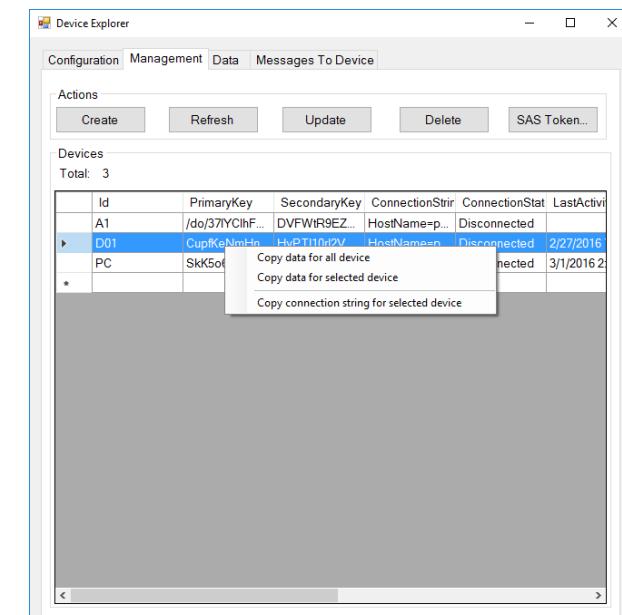
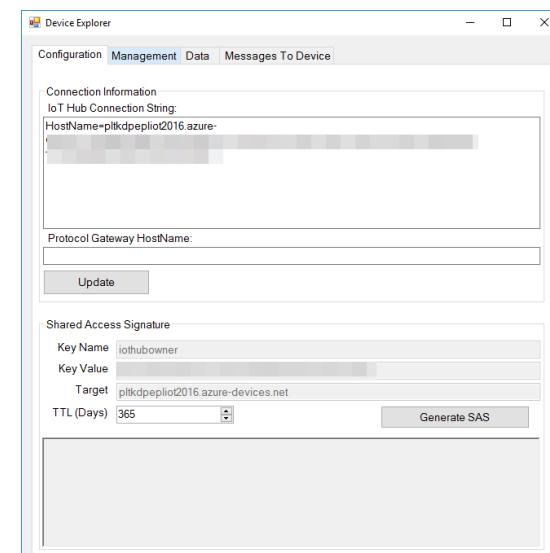
Git clone <https://github.com/tkopacz/2016windowsiot-iothub-genericsender>

READY\_TO\_RUN\DeviceExplorer.exe

Rejestracja nowego urządzenia

D<numer stanowiska> i PC

Skopiowanie haseł



# Test Azure IoT Hub

# Narzędzia

SamplePCClient\IoTClient\IoTClient.sln

Tools\setupAzureStorageExplorerCopy20160226.exe

READY\_TO\_RUN\DeviceExplorer.exe

Generowanie i podgląd komunikatów

# Aplikacja UWP do wysyłania komunikatów do IoT Hub

# Główne kroki

1. Dodać Windows IoT Extension
2. Dodać (NuGet): **Newtonsoft.Json**
3. Dodać (NuGet): **Microsoft.Azure.Devices.Client**
4. Dodać referencję z ADC Providers do ms-iot\ADC\AdcMcp3008\AdcMcp3008.csproj  
Źródło: <https://github.com/ms-iot/BusProviders.git>
5. Pobrać plik z driverem dla BMP280 – nieznacznie zmodyfikowany (poprawić przestrzenie nazw itp.)  
[https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test2\\_BMP280/Test2\\_BMP280/BMP280.cs](https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test2_BMP280/Test2_BMP280/BMP280.cs)
6. Pobrać plik z driverem do TCS34725 (poprawić przestrzenie nazw itp.):  
[https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test4\\_TCS34725/Test4\\_TCS34725/TCS34725.cs](https://raw.githubusercontent.com/tkopacz/2016windowsiot-iothub-workshopV1/master/Test4_TCS34725/Test4_TCS34725/TCS34725.cs)
7. <napisać / skopiować – projekt>

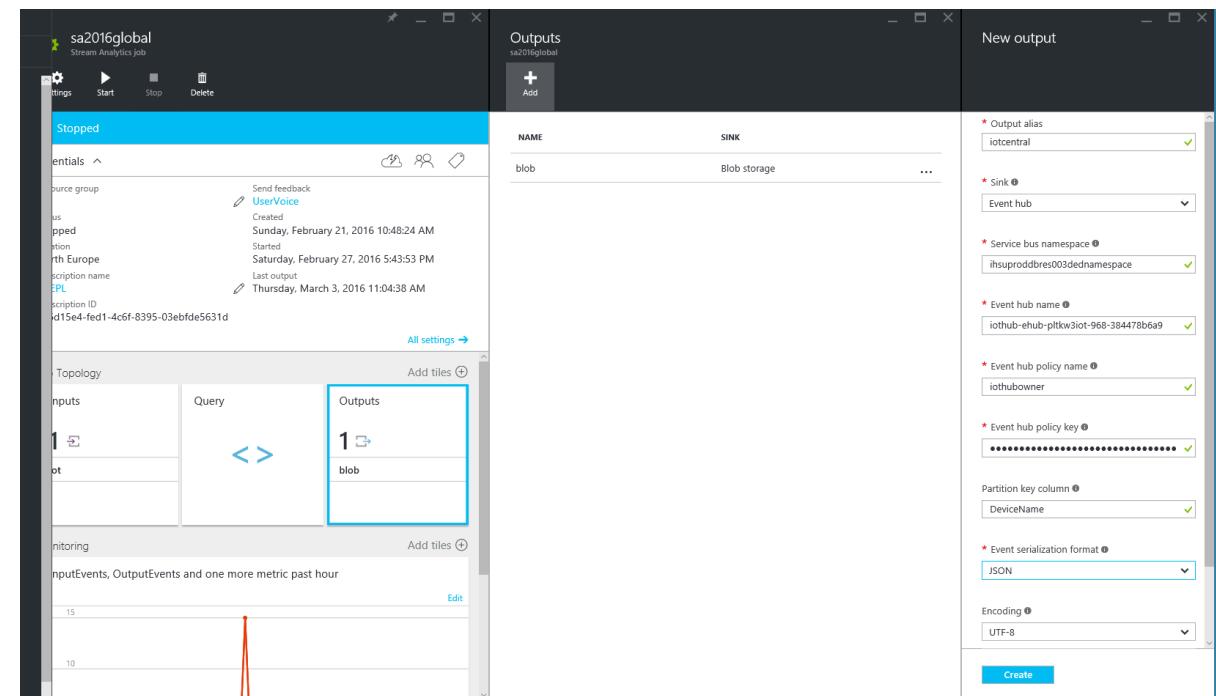
Dodanie do Stream Analytics  
dodatekowego wyjścia -  
centralnego „Event Hub”

# Zmiana kwerendy w Stream Analytics

## Kwerenda:

```
SELECT * INTO [blob] FROM [iot]
```

```
SELECT * INTO [iotcentral] FROM [iot] where MsgType='ALL'
```



Aplikacja UWP do wysyłania  
komunikatów do IoT Hub –  
MQTT

# Główne kroki

1. Skopiować projekt wykorzystujący IoT Hub Client
2. Usunąć referencję (NuGet) do `Microsoft.Azure.Devices.Client`
3. Dodać referencję (NuGet) do M2Mqtt (biblioteka dla Mqtt).
4. Zmienić wysyłanie wiadomości z Azure IoT Hub SDK – `m_clt` na `MqttClient`
5. Usunąć public async Task `ReceiveDataFromAzure()`

Własna aplikacja do odbierania  
i przetwarzania wiadomości z  
IoT Hub

# Główne kroki

1. Dodać nową aplikację konsolową ConsoleMonitor.
2. Dodac referencję (NuGet) do  
Microsoft.Azure.Devices, Microsoft.Azure.Amqp,  
Microsoft.AspNet.WebApi.Client
3. EventHubReceiver
4. eventHubReceiver.ReceiveAsync();