|  |  |
| --- | --- |
|  | Warszawa, dnia 03 września 2020 r. |
| [AB/XYZ/000/2020] |  |

Pan

Wiktor Książek

stażysta

System zdalnego pomiaru temperatury z zastosowaniem płytki uruchomieniowej Arduino UNO

Projekt wykonano we współpracy z Panem Sebastianem Buczkiem

Z wyrazami szacunku

Opiekun

dr Dariusz Świerczyński

Główny Specjalista

Spis treści

[1. Ogólny opis płytki uruchomieniowej Arduino Uno 3](#_Toc49459120)

[2. Instalowanie środowiska Arduino IDE oraz konfiguracja płytki Arduino. 3](#_Toc49459121)

[3. Pomiary temperatur 5](#_Toc49459122)

[4. Zdalne monitorowanie temperatury za pomocą prostej strony HTML 8](#_Toc49459123)

[5. Podłączenie płytki Arduino Uno do wyświetlacza LCD przez konwerter I2C 14](#_Toc49459124)

[6. Optymalizacja oprogramowania z użyciem przerwań 19](#_Toc49459125)

[7. Wprowadzenie komunikacji UDP 23](#_Toc49459126)

# 1. Ogólny opis płytki uruchomieniowej Arduino Uno

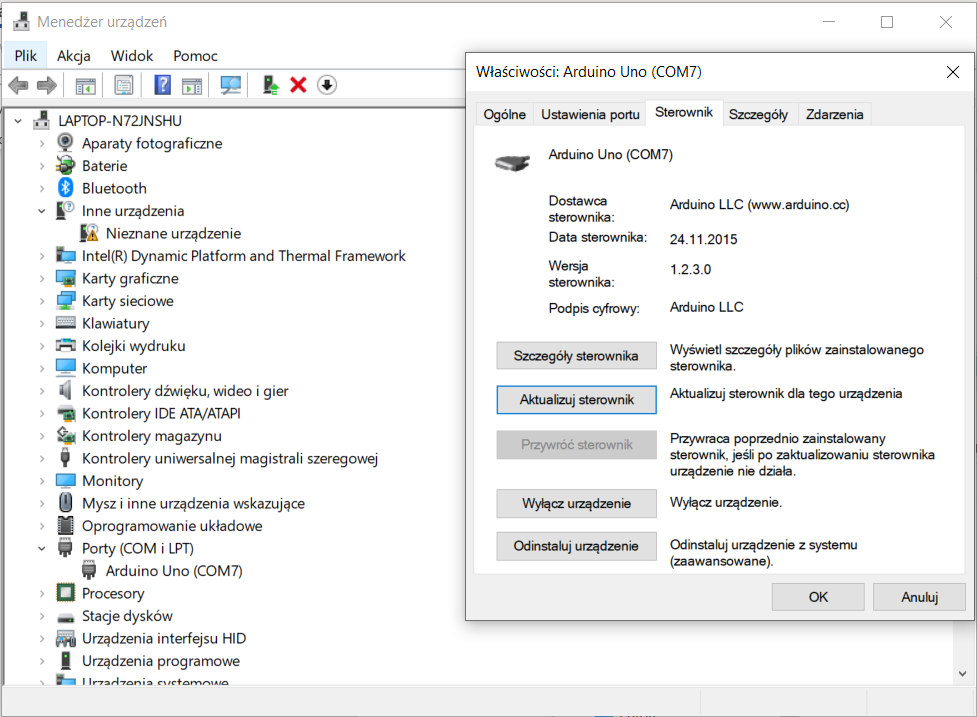
W projekcie wykorzystana zostanie płytka uruchomieniowa Arduino Uno. Z racji zaistnienia problemów z kompatybilnością zrezygnowano w późniejszych rozdziałach z platformy Arduino Due.

Arduino Uno jest to płytka uruchomieniowa wykorzystująca 8-bitowy mikrokontroler AVR ATmega328. Zasilana jest ona napięciem stałym o wartości od 6 do 20 woltów. Mikrokontroler posiada 2kB pamięci SRAM, 16kB pamięci Flash oraz 1kB pamięci EEPROM. Mikrokontroler taktowany jest z częstotliwością 16 MHz, która jest uzyskiwana z pomocą zewnętrznego rezonatora kwarcowego. Płytka posiada 14 wyjść cyfrowych o ogólnym przeznaczeniu, 6 wyjść analogowych (każdy z tych 20 pinów ma wydajność prądową na poziomie około 40mA) oraz piny służące do zasilania napięciem 5V oraz 3,3V. Poziomy napięcia w wyjściach logicznych wynoszą 5V. Układ można resetować hardwarowo z naciskając czerwony przycisk umieszczony w sąsiedztwie portu USB płytki. Układ jest dodatkowo wyposażony w zespół diód, które między innymi informują nas o wymianie danych przez port USB (diody oznaczone TX i RX) oraz umożliwiają sygnalizację poprawności załadowania kodu (programowalna dioda L). Układ posiada również header ICSP umożliwiający zastosowanie alternatywnej metody programowania mikrokontrolera.

Do najważniejszych zalet tej płytki uruchomieniowej należą: bardzo niska cena (dostępne w cenach poniżej 20 złotych), mała waga (ok 25 gram), duża różnorodność zastosowań oraz duża popularność, która skutkuje powstaniem wielu układów do współpracy z tym mikrokontrolerem oraz dużą ilością poradników oraz bibliotek ułatwiających pracę z płytką. Układ pozwala uzyskiwać komunikację w wielu standardach takich jak: SPI, 1wire, I2C, UART i wiele innych.

# 2. Instalowanie środowiska Arduino IDE oraz konfiguracja płytki Arduino.

W celu instalacji środowiska programistycznego dla systemu Arduino konieczne będzie pobranie pliku instalacyjnego ze strony producenta: https://www.arduino.cc/en/main/software. W projekcie użyto środowiska w wersji 1.8.13. Po uruchomieniu programu instalacyjnego należy zaakceptować umowę licencyjną oraz zaznaczyć wszystkie dostępne opcje instalacyjne. Po zakończeniu procesu instalacji należy uruchomić program oraz podłączyć do portu USB komputera płytkę Arduino. Przy pierwszym uruchomieniu oprogramowania na danym komputerze PC nie będzie on widoczny z poziomu środowiska Arduino IDE, co można sprawdzić przez wyszukanie dostępnych portów w zakładce narzędzia. W sytuacji, w której nie wykryto żadnego portu, do którego podłączono Arduino należy zainstalować sterowniki dla danej płytki dostępne w folderze „drivers” w folderze zainstalowanego środowiska (domyślna lokalizacja pliku: C:\Program Files (x86)\Arduino\drivers). Aby zainstalować sterownik w systemie Windows należy otworzyć manager urządzeń i w zakładce „Porty” wyszukać „nieznane urządzenie” lub odpowiednią nazwę płytki.



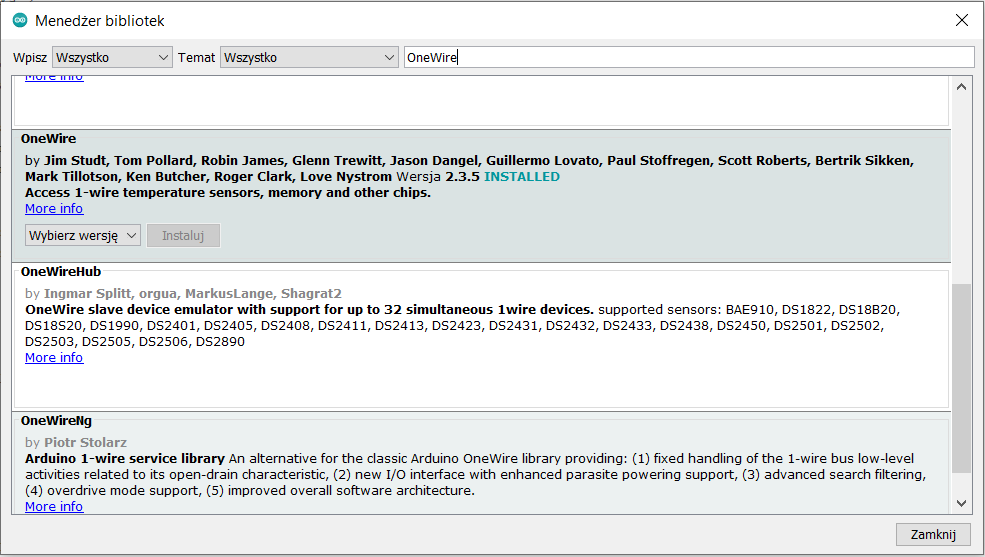
Następnie należy kliknąć lewym przyciskiem myszy na ikonkę urządzenia. Otworzone zostanie okienko „Właściwości” danego urządzenia. Należy nacisnąć przycisk „aktualizuj sterownik” w zakładce „Steronik” a następnie w nowym okienku dialogowym wybrać opcję ręcznej lokalizacji sterownika urządzenia. W nowym okienku należy podać ścieżkę folderu „drivers” (przykładowo: C:\Program Files (x86)\Arduino\drivers). Po odnalezieniu przez system odpowiedniego sterownika należy go zainstalować, co dzieje się automatycznie. W tym momencie po ponownym uruchomieniu środowiska Arduino IDE w zakładce „Narzędzia” powinien pojawić się port odpowiadający danej płytce Arduino. Należy wybrać ten port oraz wybrać odpowiednią płytkę Arduino w tej samej zakłace. W tym momencie płytka powinna być gotowa do programowania, co można sprawdzić przesyłając przykładowy program (domyślne przykłady dostępne są w zakładce Plik/Przykłady), taki jak program BLINK z kategorii Basics. Poprawna odpowiedź płytki potwierdza poprawną konfigurację środowiska programistycznego oraz płytki.

# 3. Pomiary temperatur

Do wykonania pomiarów wykorzystane będą czujniki DS18B20 oraz DHT11 pracujące równolegle. Stosowanie dwóch niezależnych czujników pozwala na zwiększenie wiarygodności wykonywanych pomiarów. Dane odbierane przez mikrokontroler dostępne są do wglądu w czasie rzeczywistym za pośrednictwem monitora portu szeregowego dostępnego po naciśnięciu przycisku w prawym górnym rogu Arduino IDE.

W pierwszym etapie prac należy połączyć czujnik DS18B20 z płytką Arduino Due. Wykorzystano do tego porty GND , numer 3 (służący do komunikacji cyfrowej) oraz 5V, które połączono odpowiednio z wyprowadzeniami 1(GND), 2(DQ) i 3(VDD) czujnika (odpowiednio od lewej). Ponieważ czujnik korzysta z protokołu komunikacyjnego 1Wire konieczne było dodanie rezystora podciągającego pomiędzy wyprowadzenie 5V i komunikacyjne DQ o wartości 4,7kOhm, co opisuje standard tego protokołu.

Po odpowiednim połączeniu czujnika pobrano bibliotekę OneWire.h oraz DallasTemperature.h pozwalające w wygodny sposób komunikować się z czujnikiem. Obie biblioteki dostępne są z poziomu menagera bibliotek w Arduino IDE, dostępnego w zakładce „Narzędzia”. Instalacja bibliotek wykonywane jest automatycznie po naciśnięciu przycisku instaluj:



Proces instalacji bibliotek jest analogiczny dla wszystkich bibliotek dostępnych za pomocą środowiska Arduino IDE. Następnie wykorzystano skrypt (nazywany w środowisku Arduino IDE Sketch (‘schemat’)) pozwalający na przetestowanie odpowiedniego połączenia z czujnikiem DS18B20. Jest on dostępny jako przykład dla biblioteki DallasTemperature.h pod nazwą „Single” (pamiętać należy o wyborze odpowiedniego pinu danych dla naszego czujnika w linii: „#define ONE\_WIRE\_BUS;” dla naszego przykładu wybrano wartość 3, która odpowiada fizycznemu połączeniu pinu 3 Arduino z wyjściem DQ czujnika). Program poprawnie zwrócił informację zwrotną potwierdzającą odpowiednie działanie układu w postaci adresu czujnika wykorzystywanego w protokole OneWire (64 bitowego numeru seryjnego, który dla naszego czujnika stanowi 0x28, 0xFF, 0xE6, 0xCC, 0x30, 0x17, 0x3, 0x9B).

Następnie wykorzystano prosty kod oparty na bibliotekach OneWire.h oraz DallasTemperature.h pozwalający na pomiar temperatury z wykorzystywanym czujnikiem. Ustawiono czułość czujnika na maksymalną (co dzieje się kosztem prędkości działania czujnika) oraz uzyskano pierwszy pomiar. Kod ten z dołączoną obsługą czujnika DHT11 jest przedstawiony poniżej.

W drugim etapie prac podłączono czujnik DHT11 zgodnie z wymaganiami stawianymi nam przez protokół komunikacyjny UART wykorzystywany przez czujnik. Napięcie 5V z płytki Arduino Due podłączono do wyprowadzenia „+”, wyprowadzenie „-” podłączono do masy, a wyprowadzenie DATA podłączono do portu numer 2 (służącego do komunikacji cyfrowej)

Po odpowiednim połączeniu czujnika pobrano bibliotekę DHT.h, która umożliwia łatwe wykonanie pomiaru. Jest ona dostępna w serwisie GitHub pod adresem: https://github.com/markruys/arduino-DHT. Proces instalacji bibliotek niestandardowych polega na rozpakowaniu zapakowanej biblioteki w specjalnie przeznaczonej do tego celu lokalizacji dostępnej domyślnie w folderze libraries środowiska Arduino IDE. Domyślnie znajduje się ona w folderze Arduino w Moich Dokumentach (przykładowo C:\Users\WK\Documents\Arduino\libraries). Po rozpakowaniu biblioteki i ponownym uruchomieniu środowiska powinna stać się ona dostępna, co można sprawdzić szukając przykładów dla tej biblioteki w środowisku Arduino IDE. Metody z tej biblioteki wykorzystano we wcześniejszym kodzie pobierającym dane z czujnika DS18B20 oraz przerzucono na port szeregowy COM5, który umożliwił obserwację pomiarów temperatury w czasie rzeczywistym. Napisany kod przedstawiono poniżej:

// Program odczytuje temperaturę z czujnika DS18B20 i DTH11

#include <OneWire.h>

#include <DallasTemperature.h>

#include <DHT.h>

#define DHT11\_PIN 2 // !!! podaj pin na ktorym DTH11 jest wpiety

float temp1 = 0.0;

int temp2;

int oneWireBus = 3; // !!! podaj pin na ktorym DS18B20 jest wpiety

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

DHT dht;

void setup(void) {

Serial.begin(57600);

Serial.println("Odczyt temperatury z czujnika DS18B20 i DHT11");

sensors.begin();

sensors.setResolution(12);

dht.setup(DHT11\_PIN);

delay(2000);

}

void loop(void) {

sensors.requestTemperatures();

temp1 = sensors.getTempCByIndex(0);

temp2 = dht.getTemperature();

Serial.print("Wartosc temperatury DS18B20:");

Serial.print(temp1);

Serial.print("\*C");

if (temp2>10){

Serial.print(" oraz DHT11:");

Serial.print(temp2);

Serial.println("\*C");

}

else Serial.println("");

delay(500);

}

Program jest uniwersalny tj. działa zarówno na platformie Arduino Uno, jak i Due.

# 4. Zdalne monitorowanie temperatury za pomocą prostej strony HTML

W tym etapie pracy dodano zdalną komunikację Ethernet z platformą Arduino. Do komunikacji wykorzystano moduł Ethernet ENC28J60.

Schemat poprawnego połączenia modułu ENC28J60 odnaleziono na stronie Botland.pl i zaprezentowano poniżej (https://botland.com.pl/pl/content/59-polaczenie-arduino-z-modulem-ethernet-enc28j60-).



Do poprawnego działania modułu konieczne jest wykonanie 5 połączeń: GND z masą Arduino, Vcc z źródłem napięcia 3,3V na platformie Arduino oraz odpowiednio połączenie CS, SI, SO oraz SCK z portami 10, 11, 12 i 13 platformy Arduino (jest to niezbędne do poprawnej komunikacji z modułem przy użyciu wykorzystanej biblioteki o nazwie EtherCard.h). EtherCard.h jest biblioteką dostępną z menadżera bibliotek Arduino IDE, co sprawia, że jej instalacja jest trywialna i została omówiona wcześniej na innym przykładzie(patrz rozdział 2). Po odpowiednim podłączeniu modułu i zainstalowaniu biblioteki okazało się, że platforma Arduino Uno nie obsługuje EtherCard.h, przez co zdecydowano się na zmianę platformy do obsługi systemu pomiarowego oraz połączenia modułu Ethernet na Arduino Uno. Po przełączeniu całego układu napisano program obsługujący połączenie z siecią Ethernet:

#include <EtherCard.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#include <DHT.h>

#define STATIC 1 // set to 1 to disable DHCP (adjust myip/gwip values below)

#define DHT11\_PIN 2

float temp1 = 0.0;

int oneWireBus = 3;

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

DHT dht;

char zxc[5];

#if STATIC

// ethernet interface ip address

static byte myip[] = { 192,168,107,7 };

// gateway ip address

static byte gwip[] = { 192,168, 107,254 };

#endif

// ethernet mac address - must be unique on your network

static byte mymac[] = { 0x2,0x8,0x25,0x7,0x44,0x11 };

byte Ethernet::buffer[500]; // tcp/ip send and receive buffer

const char page[] PROGMEM =

"HTTP/1.0 503 Service Unavailable\r\n"

"Content-Type: text/html\r\n"

"Retry-After: 600\r\n"

"\r\n"

"<html>"

"<head><title>"

"Service Temporarily Unavailable"

"</title></head>"

"<body>"

"<h3>Temperature measurement</h3>"

"<p><em>"

"Temperature1(DS18B20):<br />"

"Provided to you by WK."

"</em></p>"

"</body>"

"</html>"

;

void setup(){

Serial.begin(57600);

Serial.println("Odczyt temperatury z czujnika DS18B20 i DHT11");

sensors.begin();

sensors.setResolution(12);

dht.setup(DHT11\_PIN);

Serial.println("\n[backSoon]");

// Change 'SS' to your Slave Select pin, if you arn't using the default pin

if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0)

Serial.println( "Failed to access Ethernet controller");

#if STATIC

ether.staticSetup(myip, gwip);

#else

if (!ether.dhcpSetup())

Serial.println("DHCP failed");

#endif

ether.printIp("IP: ", ether.myip);

ether.printIp("GW: ", ether.gwip);

ether.printIp("DNS: ", ether.dnsip);

}

BufferFiller bfill;

static word homePage(float temp1, int temp2) {

bfill = ether.tcpOffset();

bfill.emit\_p(PSTR(

//"HTTP/1.0 503 Service Unavailable\r\n"

//"Content-Type: text/html\r\n"

//"Retry-After: 600\r\n"

//"\r\n"

"<html>"

"<head><title>"

"Temperature Measurement"

"</title></head>"

"<body>"

"<h3>Temperature measurement</h3>"

"<p><em>"

"Temperature1(DS18B20):$S<br />"

"Temperature2(DHT11):$D<br />"

"Provided to you by WK."

"</em></p>"

"</body>"

"</html>"),

dtostrf(temp1,3,2,zxc), temp2);

return bfill.position();

}

void loop(){

sensors.requestTemperatures();

temp1 = sensors.getTempCByIndex(0);

int temp2 = dht.getTemperature();

Serial.print("Wartosc temperatury DS18B20:");

Serial.print(temp1);

Serial.print("\*C");

if (temp2>10){

Serial.print(" oraz DHT11:");

Serial.print(temp2);

Serial.println("\*C");

}

else Serial.println("");

delay(100);

// wait for an incoming TCP packet, but ignore its contents

if (ether.packetLoop(ether.packetReceive())) {

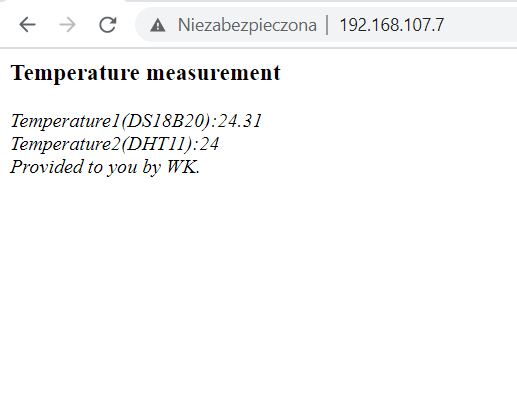
//memcpy\_P(ether.tcpOffset(), page, sizeof page);

ether.httpServerReply(homePage(temp1,temp2)); //sizeof page - 1

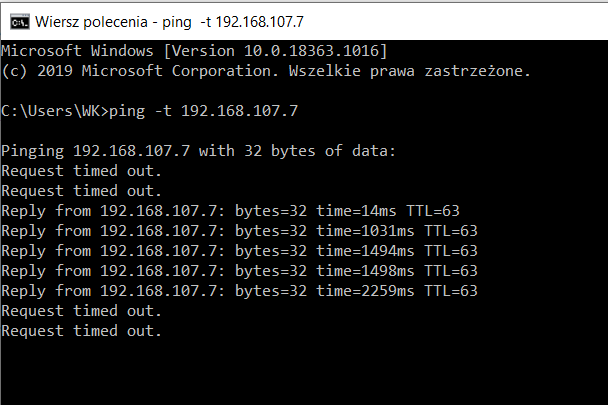
}

}

W ramach komunikacji utworzono małą HTMLową stronę publikującą w czasie rzeczywistym wyniki pomiaru temperatury, dostępną domyślnie pod adresem: http://192.168.107.7/. Treść strony zaprezentowano poniżej:



Należy zwrócić szczególną uwagę na to, że w programie adres IP oraz bramę sieciową ustawiono statycznie. Należy ją podać manualnie, ponieważ komunikacja z serwerem DHCP może okazać się zbyt niestabilna. Adres MAC przyznano urządzeniu losowo, należy tylko uważać, aby adres MAC nie powtórzył się w sieci, z której korzystamy. Kod podany wyżej funkcjonuje poprawnie, lecz jest wysoko niewydajny, ponieważ procesor może obsłużyć połączenie wyłącznie w krótkim fragmencie pętli, w którym nie obsługuje on czujników lub odczekania. Skutkuje to mało wydajnym połączeniem z mikrokontrolerem oraz długim i niestabilnym pingiem, co zaprezentowano poniżej:

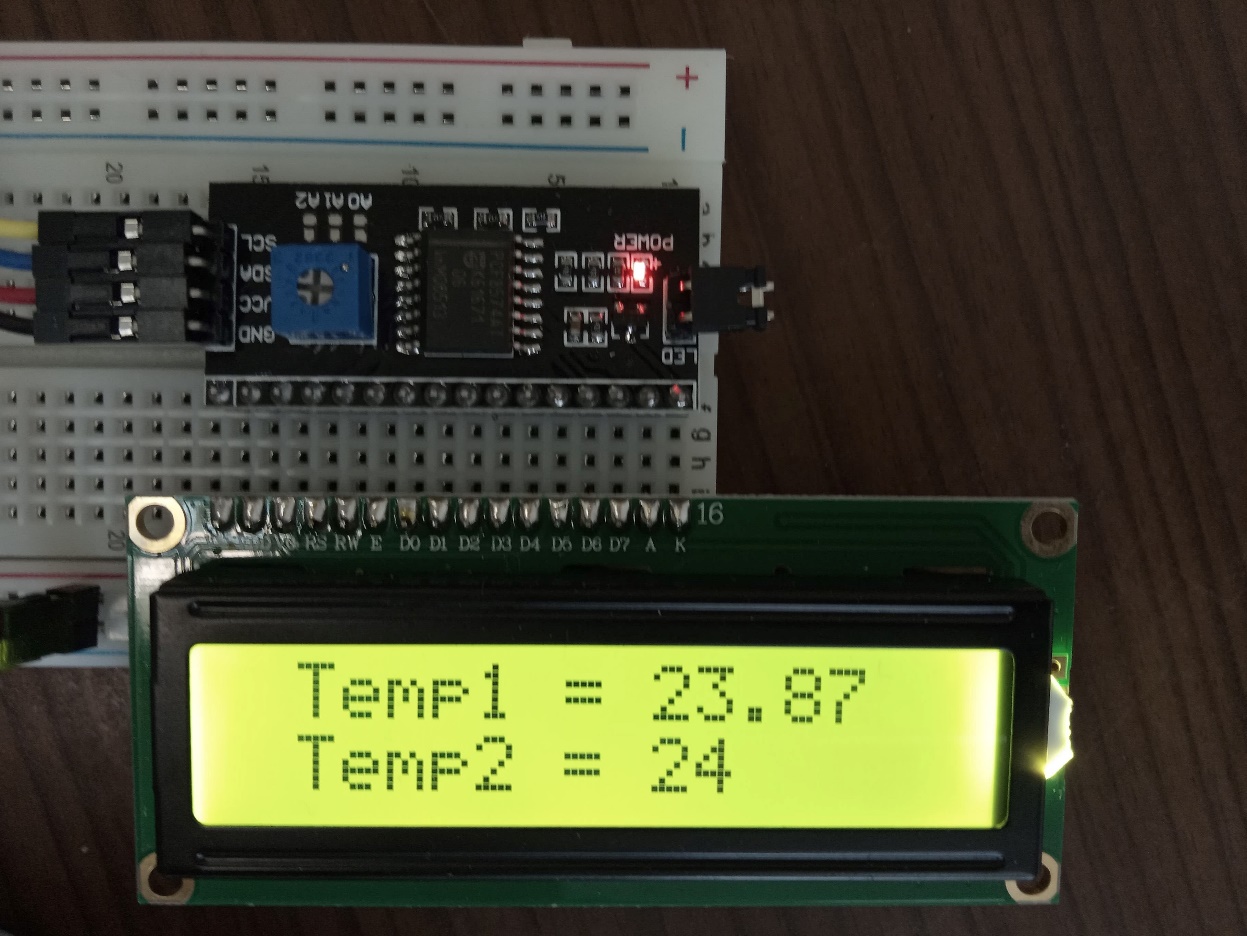


Problem rozwiązano w przyszłości przenosząc obsługę Ethernetu poza główną pętle do przerwania sterowanego timerem 1 (aktywowanym programowo co 100ms).

# 5. Podłączenie płytki Arduino Uno do wyświetlacza LCD przez konwerter I2C

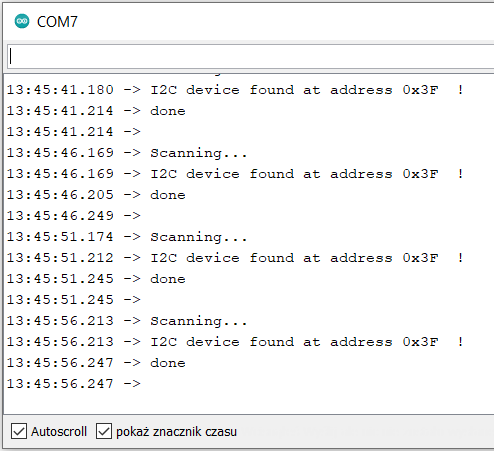
W tym etapie prac zdecydowano się dołączyć wyświetlacz LCD HD44780 o ilości pól 2\*16. Umożliwi on na lokalny monitoring pomiarów wykonywanych przez czujniki w czasie rzeczywistym bez potrzeby użycia komputera PC. Aby zminimalizować ilość przewodów koniecznych do sterowania wyświetlaczem zdecydowano się na wykorzystanie konwertera I2C LM1602 do formy akceptowanej przez moduł LCD. Do oprogramowania obsługującego wyświetlacz oraz komunikację z użyciem protokołu I2C wykorzystano bibliotekę Wire.h oraz LiquidCrystal\_I2C.h. Biblioteka Wire.h jest domyślnie wbudowana w środowisko Arduino IDE i nie wymaga dodatkowej instalacji. Biblioteka LiquidCrystal\_I2C.h jest biblioteką niestandardową, co oznacz, że jej instalacja musi być wykonana manualnie, analogicznie do przykładów podanych we wcześniejszych rozdziałach. LiquidCrystal\_I2C.h jest dostępna do pobrania w serwisie GitHub pod adresem: <https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library>.

Poprawne podłączenie modułu LCD do konwertera jest zadaniem trywialnym, które wymaga jedynie fizycznego połączenia każdego z jego 16 pinów do odpowiadającego mu pinu konwertera, co zaprezentowano na poniższym zdjęciu:



Połączenie tak przygotowanego układu do płytki uruchomieniowej Arduino Uno wymaga wykonania 4 połączeń: GND modułu z masą Arduino, VCC z zasilaniem 5V, SDA z pinem A4 Arduino oraz SCL z pinem A5 Arduino. Wybór pinu A4 oraz A5 wynika z faktu, że biblioteka LiquidCrystal\_I2C.h wymaga właśnie ich do odpowiedniego działania.

Po odpowiednim połączeniu układu konieczne jest uzyskanie adresu wyświetlacza LCD. Adres ten można uzyskać wykorzystując jeden z przykładów do biblioteki Wire.h o nazwie i2c\_scanner. W wykonanym przykładowym układzie adres wyświetlacza wynosi 0x3F.



Po uzyskaniu adresu wyświetlacza zalecane jest przetestowanie poprawnego działania układu. Do tego celu można wykorzystać przykład z biblioteki LiquidCrystal\_I2C.h o nazwie „Hello World”. Należy pamiętać, aby w przykładowym kodzie wstawić adres przyłączanego urządzenia. Po pomyślnym załadowaniu programu układ powinien odpowiednio wyświetlać komunikaty. W razie pojawienia się dodatkowych problemów należy sprawdzić, czy jasność wyświetlacza jest ustawiona no odpowiednim poziomie. Jest to wykonywane za pomocą przekręcania wału potencjometru znajdującego się na konwerterze I2C. Po pomyślnym przetestowaniu układu przystępujemy do tworzenia programu wyświetlającego wyniki pomiarów w czasie rzeczywistym na wyświetalczu (program jest zintegrowany z wcześniejszymi wersjami i zawiera oprócz pomiaru temperatur i obsługi LCD komunikację za pomocą modułu Ethernet):

// DS18B20 podlaczyc do pinu 3, DHT11 do pinu 2 w Arduino. Czujniki zasilane sa napieciem 5V.

#include <EtherCard.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#include <DHT.h>

#include <Wire.h> //biblioteki lcd

#include <LiquidCrystal\_I2C.h> //

#define STATIC 1 // set to 1 to disable DHCP (adjust myip/gwip values below) - problemy z polaczeniem (w takim wypadku zamienic na 1)

#define DHT11\_PIN 2

float temp1 = 0.0;

int oneWireBus = 3;

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

DHT dht;

LiquidCrystal\_I2C lcd(0x3F, 16, 2); // adres modulu lcd

char zxc[5];

#if STATIC

// ethernet interface ip address

static byte myip[] = {192,168,107,7 };

// gateway ip address

static byte gwip[] = {192,168,107,254};

#endif

// ethernet mac address - must be unique on your network

static byte mymac[] = { 0x2,0x8,0x25,0x7,0x44,0x11 };

byte Ethernet::buffer[500]; // tcp/ip send and receive buffer

void setup(){

Serial.begin(57600);

Serial.println("Odczyt temperatury z czujnika DS18B20 i DHT11");

sensors.begin();

sensors.setResolution(12);

dht.setup(DHT11\_PIN);

// Change 'SS' to your Slave Select pin, if you arn't using the default pin

if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0)

Serial.println( "Failed to access Ethernet controller");

#if STATIC

ether.staticSetup(myip, gwip);

#else

if (!ether.dhcpSetup())

Serial.println("DHCP failed");

#endif

ether.printIp("IP: ", ether.myip);

ether.printIp("GW: ", ether.gwip);

ether.printIp("DNS: ", ether.dnsip);

lcd.begin(); //inicjalizacja LCD

lcd.backlight(); //

}

BufferFiller bfill;

static word homePage(float temp1, int temp2) {

bfill = ether.tcpOffset();

bfill.emit\_p(PSTR(

"<html>"

"<head></head>"

"<body>"

"$S<br />"

"$D<br />"

"</body>"

"</html>"),

dtostrf(temp1,3,2,zxc), temp2);

return bfill.position();

}

void loop(){

sensors.requestTemperatures();

temp1 = sensors.getTempCByIndex(0);

int temp2 = dht.getTemperature();

Serial.print("Wartosc temperatury DS18B20:");

Serial.print(temp1);

Serial.print("\*C");

if (temp2>10){

Serial.print(" oraz DHT11:");

Serial.print(temp2);

Serial.println("\*C");

}

else Serial.println("");

delay(100);

lcd.setCursor(1,0);

lcd.print("Temp1 = ");

lcd.print(temp1);

lcd.setCursor(1,1);

lcd.print("Temp2 = ");

lcd.print(temp2);

// wait for an incoming TCP packet, but ignore its contents

if (ether.packetLoop(ether.packetReceive())) {

//memcpy\_P(ether.tcpOffset(), homePage(temp1,temp2), sizeof homePage(temp1,temp2));

ether.httpServerReply(homePage(temp1,temp2)); //sizeof page - 1

}

}

Napisany program z powodzeniem prezentuje pomiary z obydwu czujników, gdzie zmienna temp1 reprezentuje czujnik DS18B20, a temp 2 czujnik DHT11. Wyświetlane dane pomiarowe odzwierciedlają dokładność pomiarową czujników wynoszącą dla DS18B20 setne części stopnia Celsjusza, a dla DHT11 +/- 1 stopień Celsjusza.

# 6. Optymalizacja oprogramowania z użyciem przerwań

Program obsługujący wszystkie peryferia dołączone do płytki uruchomieniowej Arduino przedstawiony w rozdziale 5 posiada poważne wady. Podstawowym problemem jest to, że każde z peryferiów obsługiwane jest „jedno po drugim” w pętli głównej „loop”, której wykonanie ze względu na niekrótki czas pomiarów, obsługi wyświetlacza i komunikacji zajmuje łącznie około sekundy. Oznacza to, że żadne z peryferiów nie może komunikować się z mikrokontrolerem częściej niż co jedną sekundę. O ile nie zakłóca to jeszcze działania systemu akwizycji danych, czujników oraz wyświetlacza, to komunikacja z systemem przez moduł Ethernet działa już bardzo słabo. Moduł „nasłuchuje” zapytań tylko przez parę milisekund w sekundzie, co w konsekwencji oznacza, że większość prób połączenia się z modułem będzie nieudana, a przesłanie jakiejkolwiek większej ilości danych zajmie bardzo dużo czasu. Moduł Ethernet może działać wydajnie tylko wtedy kiedy nasłuchuje przez większość czasu oraz pracuje niezależnie od obciążenia mikrokontrolera przez pozostałe peryferia.

Rozwiązaniem tego problemu jest utworzenie nowej pętli działającej równolegle z pętla główną, zwaną przerwaniem. Zastosowane przez nas przerwania aktywowane są wewnętrznym timerem (w tym przypadku o numerze 1, co wynika z wykorzystywania w programie biblioteki TimerOne.h, która wymaga użycia właśnie tego timera) płytki uruchomieniowej Arduino Uno z częstotliwością około raz na 100ms. Zasada działania przerwania polega na tym, że układ chwilowo przerywa aktualne wykonywane instrukcje z pętli głównej i przechodzi do pętli aktywowanej przerwaniem. Po wykonaniu instrukcji w tej drugiej pętli procesor wraca do instrukcji wcześniej przerwanych. Pozwala to zapewnić, że komunikacja z modułem Ethernet przebiegać będzie płynnie i bez zakłóceń. Częstotliwość występowania przerwań została starannie dobrana. Zbyt częste przerwania powodują powstanie zakłóceń w procesie sczytywania danych z czujników oraz obsługi wyświetlacza, zbyt rzadkie powodują zakłócenia w komunikacji.

Praktyczna implementacja przerwań aktywowanych timerami wymaga od nas użycia dodatkowej biblioteki TimerOne.h. Jest to jedna z bibliotek dostępnych domyślnie z poziomu managera biblioteki, a jej proces instalacyjny jest analogiczny do pozostałych bibliotek dostępnych z tego poziomu.

Program z rozdziału 6 zmodyfikowany do obsługi przerwań przedstawiono poniżej:

// DS18B20 podlaczyc do pinu 3, DHT11 do pinu 2 w Arduino. Czujniki zasilane sa napieciem 5V.

#include <OneWire.h> // komunikacja z czujnikiem 1 (1Wire)

#include <DallasTemperature.h> // obsluga czujnika 1

#include <DHT.h> // obsluga czujnika 2

#include <Wire.h> // biblioteki lcd (komunikacja I2C)

#include <LiquidCrystal\_I2C.h> // biblioteki lcd

#include <EtherCard.h> // biblioteka do obslugi modulu ethernet ENC28J80

#include <TimerOne.h> // obsluga przerwania - ethernet

#define STATIC 1 // set to 1 to disable DHCP (adjust myip/gwip values below) - problemy z polaczeniem (w takim wypadku zamienic na 1)

#define DHT11\_PIN 2 // przypisanie pinu 2 czujnikowi 2 (DHT11)

int oneWireBus = 3; // przypisanie pinu 3 czujnikowi 3 (DS18B20)

volatile float temp1 = 0.0; // zadelarowanie zmiennej temp1 (DS18B20)

volatile int temp2 = 0; // zadelarowanie zmiennej temp2 (DHT11)

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

DHT dht;

LiquidCrystal\_I2C lcd(0x3F, 16, 2); // adres modulu LCD

char zxc[5];

#if STATIC

static byte myip[] = {192,168,107,7}; // ethernet interface ip address

static byte gwip[] = {192,168,107,254}; // gateway ip address

#endif

static byte mymac[] = { 0x2,0x8,0x25,0x7,0x44,0x11 }; // ethernet mac address - must be unique on your network

byte Ethernet::buffer[500]; // tcp/ip send and receive buffer

void setup(){

Serial.begin(57600);

Serial.println("Odczyt temperatury z czujnika DS18B20 i DHT11");

sensors.begin();

sensors.setResolution(12);

dht.setup(DHT11\_PIN);

if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0)

Serial.println( "Failed to access Ethernet controller");

#if STATIC

ether.staticSetup(myip, gwip);

#else

if (!ether.dhcpSetup())

Serial.println("DHCP failed");

#endif

ether.printIp("IP: ", ether.myip);

ether.printIp("GW: ", ether.gwip);

ether.printIp("DNS: ", ether.dnsip);

lcd.begin(); //inicjalizacja LCD

lcd.backlight(); //

Timer1.initialize(100000); // obsluga przerwania - ethernet (ping do ok 100ms - zbyt duża czestosc przerwan moze zakłócać pracę pozostałych peryferiów)

Timer1.attachInterrupt(ethercomms); // obsluga przerwania - ethernet

}

BufferFiller bfill;

static word homePage(float temp1, int temp2) {

bfill = ether.tcpOffset();

bfill.emit\_p(PSTR(

"<html>"

"<head></head>"

"<body>"

"$S<br />"

"$D<br />"

"</body>"

"</html>"),

dtostrf(temp1,3,2,zxc), temp2);

return bfill.position();

}

void loop(){

sensors.requestTemperatures();

temp1 = sensors.getTempCByIndex(0);

temp2 = dht.getTemperature();

Serial.print("Wartosc temperatury DS18B20:");

Serial.print(temp1);

Serial.print("\*C");

if (temp2>10){

Serial.print(" oraz DHT11:");

Serial.print(temp2);

Serial.println("\*C");

}

else Serial.println("");

delay(100);

lcd.setCursor(1,0);

lcd.print("Temp1 = ");

lcd.print(temp1);

lcd.setCursor(1,1);

lcd.print("Temp2 = ");

lcd.print(temp2);

}

void ethercomms(){

// obsluga modulu ethernetowego w przerwaniu triggerowanym timerem pozwala na

// jego plynne dzialanie niezaleznie od innych wykonywanych przez mikrokontroler zadan

//

// wait for an incoming TCP packet, but ignore its contents

if (ether.packetLoop(ether.packetReceive())) {

ether.httpServerReply(homePage(temp1,temp2));

}

}

Używając komendy ping -t 192.168.107.7 z poziomu „wiersza poleceń” systemu Windows 10 przekonujemy się, że program implementujący komunikację z przerwaniami pozwala otrzymać wymianę danych o znacznie mniejszej wartości opóźnień (ping) oraz o znacznie większej stabilności. Zwiększoną jakość komunikacji można również zbadać poprzez próbę odczytu danych pomiarowych za pomocą stworzonej wcześniej strony html, której długość ładownia oraz częstotliwość występowania błędów w połączeniu znacznie się obniżyła.

# 7. Wprowadzenie komunikacji UDP

Dalszym etap rozwoju projektu polega na bezpośrednim przesłaniu danych pomiarowych do zewnętrznej bazy danych, gdzie zostaną one przetworzone oraz przedstawione w postaci multimedialnej umożliwiającej analizę zmienności temperatury w badanym środowisku w zadanym czasie. W celu uzyskania wydajnej oraz stabilnej komunikacji z zewnętrznym serwerem konieczna jest zmiana sposobu komunikacji systemu pomiarowego ze światem zewnętrznym. Zaprezentowana we wcześniejszych rozdziałach metoda zdalnego monitoringu temperatury w czasie rzeczywistym za pomocą stronki html zostanie zamieniona na model bazujący na protokole UDP.

UDP (User Datagram Protocol) jest to protokół komunikacyjny warstwy transportowej w systemie OSI stanowiący alternatywę dla częściej stosowanego protokołu TCP. Jest on bezpołączeniowy co oznacza, że raz wysłana porcja informacji (nazywana datagramem), nie wiąże się już z potrzebą żadnej dodatkowej komunikacji między użytkownikami. Stanowi to istotną różnicę w porównaniu do protokołu TCP, który wymaga przesłania od obiorcy do nadawcy po otrzymaniu datagramu sumy kontrolnej potwierdzającej pomyślną transmisję danych. Sprawia to, że protokół UDP jest mniej pewny w porównaniu do protokołu TCP, ponieważ w sytuacji niedostarczenia datagramu lub jego uszkodzenia nie istnieje żaden mechanizm, który pozwoliłby na wykrycie tego błędu oraz ponowne wykonanie przesłania tego samego pakietu. Do zalet protokołu UDP należy zaliczyć jego prostotę oraz dużą prędkość transmisji w porównaniu do protokołu TCP. Oznacza to, że protokół UDP używany jest w zastosowaniach niewymagających bezbłędnej transmisji, które wymagają dużych prędkości wymiany danych takich jak na przykład streaming wideo.

Ze względu na właśnie prostotę tej technologii zdecydowano się na jej zastosowanie w komunikacji pomiędzy systemem pomiarowym, a zewnętrzną bazą danych. Niestabilność połączenia systemu z siecią o podłożu hardwareowym bardzo utrudniałaby pomyślną wymianę danych w protokole TCP, która do odpowiedniego przesłania pojedynczego pakietu wymaga wielokrotnego połączenia między odbiorcą, a nadawcą.

Do implementacji modyfikacji koniecznych do komunikacji za pomocą protokołu UDP zastosowano bibliotekę IPAddress.h. Jest to biblioteka domyślnie dostępna z poziomu Arduino IDE i nie wymaga ona żadnej instalacji.

Poniżej zaprezentowano modyfikację programu z rozdziału 6, który zamiast komunikacji opartej na TCP stosuje protokół UDP:

// DS18B20 podlaczyc do pinu 3, DHT11 do pinu 2 w Arduino. Czujniki zasilane sa napieciem 5V.

#include <OneWire.h> // komunikacja z czujnikiem 1 (1Wire)

#include <DallasTemperature.h> // obsluga czujnika 1

#include <DHT.h> // obsluga czujnika 2

#include <Wire.h> // biblioteki lcd (komunikacja I2C)

#include <LiquidCrystal\_I2C.h> // biblioteki lcd

#include <EtherCard.h> // biblioteka do obslugi modulu ethernet ENC28J80

#include <IPAddress.h>

#include <TimerOne.h> // obsluga przerwania - ethernet

#define STATIC 1 // set to 1 to disable DHCP (adjust myip/gwip values below) - problemy z polaczeniem (w takim wypadku zamienic na 1)

#define DHT11\_PIN 2 // przypisanie pinu 2 czujnikowi 2 (DHT11)

int oneWireBus = 3; // przypisanie pinu 3 czujnikowi 3 (DS18B20)

volatile float temp1 =0.0; // zadelarowanie zmiennej temp1 (DS18B20)

volatile int temp2 = 0; // zadelarowanie zmiennej temp2 (DHT11)

char buf1[5];

char buf2[5];

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

DHT dht;

LiquidCrystal\_I2C lcd(0x3F, 16, 2); // adres modulu LCD

char zxc[5];

#if STATIC

static byte myip[] = {192,168,107,7}; // ethernet interface ip address

static byte gwip[] = {192,168,107,254}; // gateway ip address

unsigned int myport = 8888;

#endif

static byte mymac[] = { 0x2,0x8,0x25,0x7,0x44,0x11 }; // ethernet mac address - must be unique on your network

byte Ethernet::buffer[500]; // tcp/ip send and receive buffer

void udpSerialPrint(uint16\_t port, uint8\_t ip[4],uint16\_t src\_port, const char \*data, uint16\_t len){

IPAddress src(ip[0], ip[1], ip[2], ip[3]);

Serial.print("Adres zrodlowy: ");

Serial.println(src);

Serial.print("Port zrodlowy: ");

Serial.println(src\_port);

Serial.print("Jakis port: : ");

Serial.println(port);

Serial.print("Wiadomosc: ");

Serial.println(data);

if(\*data=='1') ether.makeUdpReply(buf1, sizeof buf1, src\_port);

else ether.makeUdpReply(buf2, sizeof buf2, src\_port);

}

void setup(){

Serial.begin(57600);

Serial.println("Odczyt temperatury z czujnika DS18B20 i DHT11");

sensors.begin();

sensors.setResolution(12);

dht.setup(DHT11\_PIN);

if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0)

Serial.println( "Failed to access Ethernet controller");

#if STATIC

ether.staticSetup(myip, gwip);

#else

if (!ether.dhcpSetup())

Serial.println("DHCP failed");

#endif

ether.printIp("IP: ", ether.myip);

ether.printIp("GW: ", ether.gwip);

ether.printIp("DNS: ", ether.dnsip);

lcd.begin(); //inicjalizacja LCD

lcd.backlight(); //

Timer1.initialize(100000); // obsluga przerwania - ethernet (ping do ok 100ms - zbyt duża czestosc przerwan moze zakłócać pracę pozostałych peryferiów)

Timer1.attachInterrupt(ethercomms); // obsluga przerwania - ethernet

if(ether.begin(sizeof Ethernet::buffer, mymac) == 0)

Serial.println("Nie udalo sie polaczyc z kontrolerem");

#if STATIC

ether.staticSetup(myip,gwip);

#endif

ether.udpServerListenOnPort(&udpSerialPrint, myport);

}

BufferFiller bfill;

void loop(){

sensors.requestTemperatures();

temp1 = sensors.getTempCByIndex(0);

temp2 = dht.getTemperature();

int temp1mod = (int)(temp1\*100);

sprintf(buf1,"%i",temp1mod);

sprintf(buf2,"%i",temp2);

Serial.println(buf1);

Serial.println(buf2);

Serial.print("Wartosc temperatury DS18B20:");

Serial.print(temp1);

Serial.print("\*C");

if (temp2>10){

Serial.print(" oraz DHT11:");

Serial.print(temp2);

Serial.println("\*C");

}

else Serial.println("");

delay(100);

lcd.setCursor(1,0);

lcd.print("Temp1 = ");

lcd.print(temp1);

lcd.setCursor(1,1);

lcd.print("Temp2 = ");

lcd.print(temp2);

}

void ethercomms(){

// obsluga modulu ethernetowego w przerwaniu triggerowanym timerem pozwala na

// jego plynne dzialanie niezaleznie od innych wykonywanych przez mikrokontroler zadan

ether.packetLoop(ether.packetReceive());

}

Program wysyła pomiar z jednego z dwóch dostępnych czujników: DS18B20 dostępnego ze zmiennej zmiennoprzecinkowej temp1 oraz DHT11 dostępnego spod zmiennej typu int temp2. W celu łatwego przesyłu danych zmienne temp1 oraz temp2 są po wykonaniu pomiarów konwertowane do postaci tablic znakowych i zapisane do zmiennych o nazwach odpowiednio buf1 oraz buf2.

Procedura przesyłu pomiaru temperatury z określonego czujnika przebiega w następujący sposób:

1. Zewnętrzny klient wysyła zapytanie do układu pomiarowego w celu uzyskania danych z konkretnego czujnika (w zależności od treści zapytania dostaje on różne odpowiedzi: dla zapytania o treści „1” otrzymuje on pomiar z czujnika pierwszego, a dla zapytania o treści „2” otrzymuje pomiar z czujnika drugiego; zapytania powinny być kodowane w systemie UTF-8). W celu wysłania zapytania konieczne jest określenie adresu IP Arduino Uno oraz portu docelowego. Wysłany datagram zawiera pełną informację o nadawcy.

2. Serwer UDP działający z poziomu płytki uruchomieniowej Arduino Uno odbiera wysłane wcześniej zapytanie i interpretuje je tak, aby w odpowiedzi wysłać datagram pochodzący z odpowiedniej zmiennej.

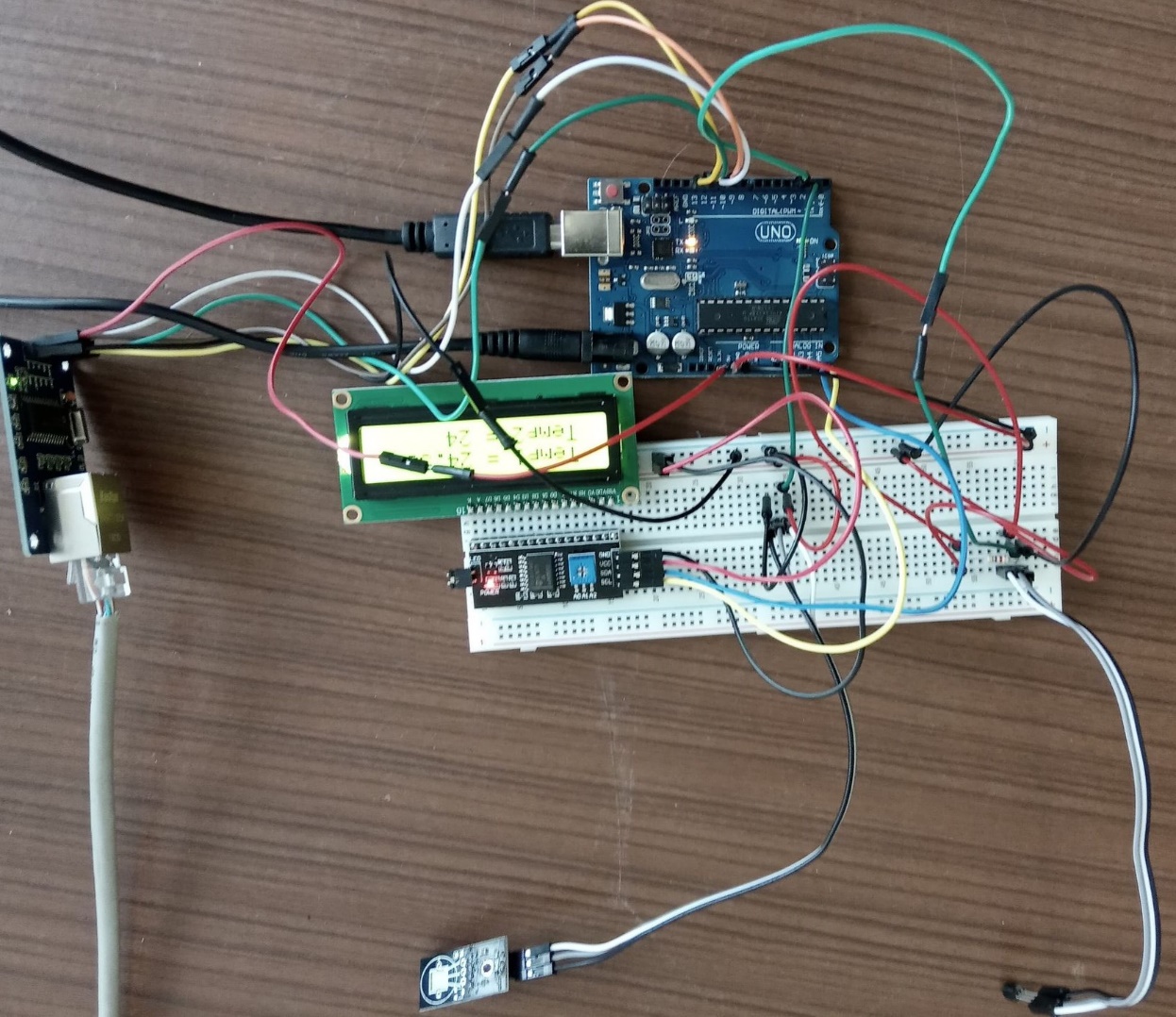
3. Serwer UDP wysyła odpowiedź do zewnętrznego klienta, którą adresuje na podstawie danych nadawcy umieszczonych w nagłówku odebranego wcześniej zapytania.

4. Zewnętrzny klient odbiera wysłaną odpowiedź oraz zapisuje ją w swojej bazie danych w celu późniejszego wykorzystania.

Zaimplementowana komunikacja UDP umożliwia uzyskanie stabilnego połączenia między zewnętrzną bazą danych a systemem akwizycji danych pomiarowych.

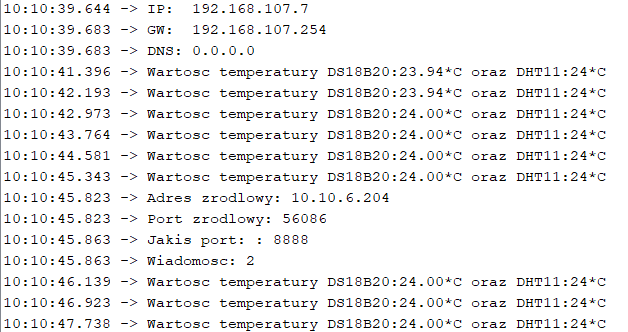
# 8. Podsumowanie

Wykonany system pomiarowy przedstawiono na zdjęciu poniżej poniżej:



Po lewej stronie zdjęcia widoczny jest moduł ENC28J60 podłączony do kabla Ethernetowego. W prawym dolnym rogu zdjęcia widoczny jest czujnik DS18B20, a po lewej od niego widoczny jest czujnik DHT11. Na zdjęciu widać również w jak prosty sposób można połączyć moduł LCD z konwerterem I2C. Połączenia elementów do płytki Arduino Uno wykonano z użyciem płytki typu breadboard.

Potwierdzenie odpowiedniego działania kodu z rozdziału 7, który stanowi finalną formę projektu przedstawiono na poniższym zrzucie prezentującym monitor portu szeregowego Arduino IDE:



Na zdjęciu widoczny jest moment uzyskania przez moduł ENC28J60 połączenia z siecią oraz parę pierwszych pomiarów wykonywanych przez czujnik. Po wykonaniu sześciu pomiarów płytka Arduino odbiera zapytanie od zewnętrznego klienta o pomiar z czujnika 2. W reakcji na to zapytanie mikrokontroler poprawnie wysłał odpowiedź.

Przedstawione powyżej zrzuty stanowią potwierdzenie poprawnego działania systemu pomiarowego opartego na płytce Arduino Uno, co było przedmiotem projektu.