

Czujnik temperatury, Biblioteki

Opracował opiekun przedmiotu dr inż. Krzysztof Chudzik

Wydział Informatyki i Telekomunikacji
Politechnika Wrocławskiego

Wrocław, 2021.08.27 17:49:27

Spis treści

1 Czujnik temperatury	1
1.1 Czujnik DS18B20	1
1.2 Magistrala 1-Wire i podłączenie czujników w zestawie laboratoryjnym	2
1.3 Obsługa programowa czujnika DS18B20	3
2 Moduły i własne biblioteki programowe	4
2.1 Podział kodu na moduły	4
2.2 Biblioteki	6

Lista zadań

1 Program wykorzystujący czujniki DS18B20	4
2 Implementacja własnej biblioteki ze sterownikiem wybranego urządzenia	7

1 Czujnik temperatury

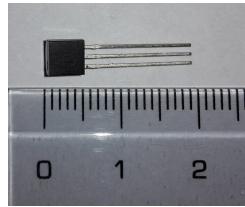
1.1 Czujnik DS18B20

Temperatura, jest jedną z najczęściej mierzonych wielkości fizycznych. Żyjemy w otoczeniu różnych termometrów. Znajdziemy je we wskaźnikach temperatury wewnętrz i na zewnątrz pomieszczeń, w układach klimatyzacji, sprzęcie powszechnego użytku takim jak czajniki czy piekarniki, sprzęcie medycznym, systemach przemysłowych, itd. Nawet wewnętrz układu scalonego mikrokontrolera ATmega328P z płytka Arduino UNO jest czujnik temperatury. Duża część z nich to elektroniczne czujniki temperatury.

Czujniki w postaci układów scalonych są chyba najwygodniejszymi w użyciu. Mogą one dostarczać sygnał analogowy, ale często wyposażone są przetwornik analogowo cyfrowy i możliwość wykorzystania magistrali komunikacyjnej. Ułatwia to ich integrację w systemach z mikrokontrolerami. Przykładem takiego układu może być wykorzystywany na laboratorium czujnik temperatury DS18B20.

Na witrynie producenta układu DS18B20 dostępna jest jego [karta katalogowa \(link\)](#). Specyfikacja ta podaje następujące podstawowe parametry układu. Zapewnia pomiary temperatury z rozdzielcością od 9 do 12 bitów i ma funkcję alarmu z nieulotnymi programowanymi przez użytkownika górnymi i dolnymi punktami wyzwalania. DS18B20 komunikuje się za pośrednictwem magistrali 1-Wire, która z definicji wymaga tylko jednej linii danych (i uziemienia) do komunikacji z centralnym mikroprocesorem. Ma zakres temperatur roboczych od -55°C do + 125°C. Każdy DS18B20 ma unikalny 64-bitowy numer seryjny, który pozwala wielu DS18B20 działać na tej samej magistrali 1-Wire. W ten sposób można łatwo użyć jednego mikroprocesora do sterowania wieloma DS18B20 rozmieszczonymi na dużym obszarze.

Układ scalony DS18B20 przedstawiony jest na fotografii 1. Taki układ zamontowany jest wewnątrz zestawu laboratoryjnego.



Fotografia 1: Czujnik temperatury DS18B20.

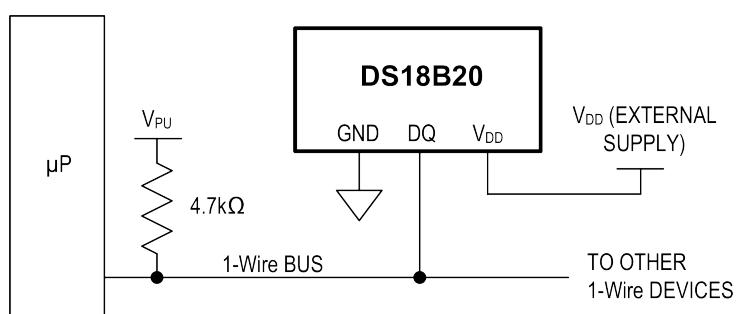
Wykorzystanie praktyczne wymaga w wielu przypadkach zabudowania układu, aby chronić go przed czynnikami zewnętrznymi. Może być on przygotowany do pomiaru, na przykład, w środowisku, gdzie panuje duże zawilgocenie lub do pomiaru temperatury wody bezpośrednio. Wtedy czujnik zabudowany jest w postaci sondy, jak to przedstawiono na fotografii 2. W taką sondę wyposażony jest zestaw laboratoryjny.



Fotografia 2: Sonda wodoodporna z czujnikiem temperatury DS18B20.

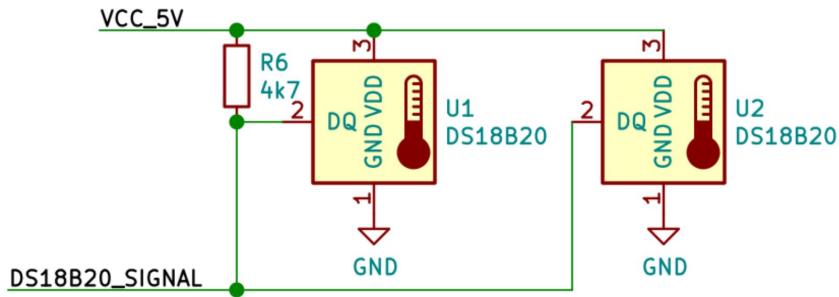
1.2 Magistrala 1-Wire i podłączenie czujników w zestawie laboratoryjnym

Komunikacja z układem następuje za pomocą magistrali 1-Wire. Jest to asynchroniczna magistrala szeregową. Ten typ magistrali może wykorzystywać połączenie dwu- lub trzy-przewodowe. Na laboratorium korzystamy z tego drugiego. Karta katalogowa podaje, że podłączenie powinno wyglądać jak na schemacie 1. Szersze omówienie tej magistrali na wykładzie.



Schemat 1: 1-Wire w wersji trzy-przewodowej. Schemat z karty katalogowej DS18B20 producenta, firmy Maxim Integrated.

Dokładnie w taki sposób, połączone są oba czujniki DS18B20 w zestawie laboratoryjnym, czyli układ scalony i sonda z układem scalonym wewnątrz. Ilustruje to schemat 2.



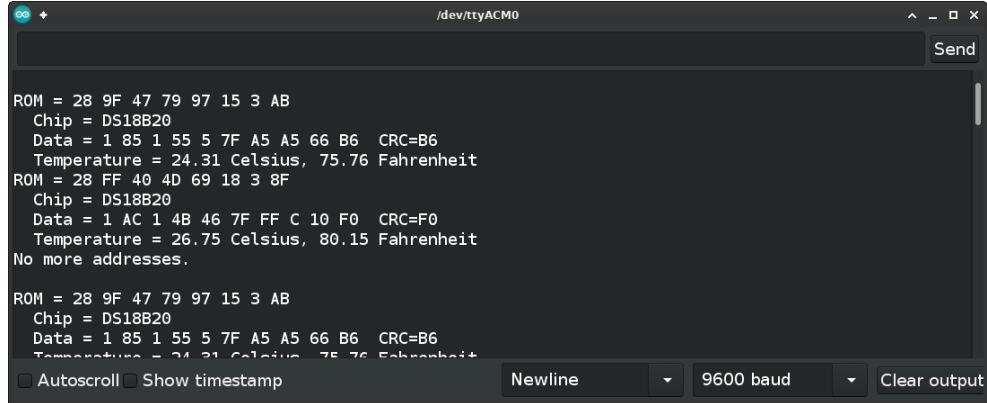
Schemat 2: Podłączenie czujników DS18B20 w zestawie laboratoryjnym. Linia sygnałowa DS18B20_Signal podłączona jest do pinu A1 płytki Arduino UNO.

1.3 Obsługa programowa czujnika DS18B20

Obsługa czujników wymaga bibliotek *DallasTemperature* i *OneWire*. W ramach przykładów dostarczonych z biblioteką *OneWire* znajduje się program dostępny w Arduino IDE na Menu: *File > Examples > OneWire > DS18x20_Temperature*. Program ten cyklicznie wyszukuje i wyświetla informacje o naszych czujnikach na magistrali 1-Wire. Aby program działał w naszym zestawie, należy zmienić jedynie deklarację obiektu magistrali w linii 10 do następującej postaci:

```
OneWire ds(A1);
```

ponieważ, do pinu A1 płytki Arduino przyłączone są czujniki temperatury. Wynikiem działania programu jest lista drukowana przez łącze szeregowe. Przykładowe wyniki prezentuje rysunek 1.



Rysunek 1: Wynik działania programu przykładowego *DS18x20_Temperature* z biblioteki *OneWire* na jednym z zestawów laboratoryjnych.

Do obsługi czujników wykorzystamy biblioteki *DallasTemperature* i *OneWire*. Ta druga to obsługa magistrali 1-Wire. Biblioteka *DallasTemperature* umożliwia dostęp do odczytów temperatury z czujników DS18B20. Przykładowy program, umożliwiający odczyt temperatury z czujników przedstawiony jest jako kod 1. Program odczytuje dane z czujników i prezentuje je na wyświetlaczu LCD.

Kod 1: Program odczytujący temperaturę z obu czujników zestawu laboratoryjnego.

```

1 #include <Wire.h>
2 #include <LiquidCrystal_I2C.h>
3 #include <OneWire.h>
4 #include <DallasTemperature.h>
5
6 LiquidCrystal_I2C lcd(0x27, 16, 2);
7
8 OneWire oneWire(A1);
9 DallasTemperature tempSensors(&oneWire);
10
11 void setup()
12 {
13     tempSensors.begin();
14
15     lcd.init();
16     lcd.backlight();
17     lcd.clear();
18 }
19
20 void loop()
21 {

```

```

22 tempSensors.requestTemperatures();
23 float tempIn = tempSensors.getTempCByIndex(1);
24 float tempOut = tempSensors.getTempCByIndex(0);
25
26 char buffer[40];
27 sprintf(buffer, "Temp IN%8s", String(tempIn, 4).c_str());
28 lcd.setCursor(0, 0);
29 lcd.print(buffer);
30 sprintf(buffer, "Temp OUT%8s", String(tempOut, 4).c_str());
31 lcd.setCursor(0, 1);
32 lcd.print(buffer);
33 }

```

Program jest stosunkowo prosty i nie wymaga rozległych komentarzy. Pewnym problemem jest tu identyfikacja czujników. Wyniki działania programu *DS18x20_Temperature* zaprezentowane na rysunku 1 pozwalają nam poznać dokładnie adresy urządzeń. Są one w liniach rozpoczynających się słowem **ROM**. Można się do czujników odwoływać korzystając z tego adresu. Unikniemy wtedy niejednoznaczności i będziemy mieli pewność, że rozmawiamy z tym urządzeniem, z którym chcemy. Niemniej jednak, nie bardzo wiadomo jak adres powiązać z fizycznym czujnikiem czy sondą, bo nie są one opisane tymi adresami na obudowie. Nasuwają się dwie metody identyfikacji. Pierwsza, to włączać je pojedynczo, co w przypadku zestawu laboratoryjnego w czasie zajęć jest zabronione, bo zabronione jest manipulowanie w układzie elektronicznym zestawów przez Studentów. Druga metoda jest „nieinwazyjna” i prosta. Włączamy odczyty z obu czujników, ogrzewamy sondę na kablu dlonią. Jej temperatura powinna wzrosnąć. Pozwala to zidentyfikować czujniki. Proszę to zrobić pisząc własne programy, bo nie ma gwarancji, że dostęp przez indeks nie zamienił kolejności czujników w zestawie, w stosunku do programu przykładowego.

Wyjaśnienia wymaga też dlaczego w kodzie 1 przyjęto precyzję aż czterech miejsc po przecinku. Uczyniono to nie ze względów praktycznych, ale poznawczych. Karta katologowa czujnika DS18B20 stwierdza co następuje:

"The resolution of the temperature sensor is user-configurable to 9, 10, 11, or 12 bits, corresponding to increments of 0.5°C, 0.25°C, 0.125°C, and 0.0625°C, respectively. The default resolution at power-up is 12-bit."

Precyzja do czwartego miejsca po przecinku pozwala zobaczyć, że jest to rzeczywiście odczyt, który nie przyjmuje dowolnych wartości. Jeśli poświęcicie Państwo nieco czasu na obserwację, to dojdziecie do wniosku, że przeciętnemu człowiekowi patrzącemu na ekran LCD należy zaprezentować wartości do pierwszego miejsca po przecinku.Więcej tylko wtedy, kiedy istnieją realne przesłanki, aby to uczynić.

Uwaga do kodu 1: Wyjaśnijmy, dlaczego w pętli `loop()`, wydruk do bufora jest następującej postaci:

```
sprintf(buffer, "Temp IN%8s", String(tempIn,4).c_str());
```

W przypadku Arduino, funkcja `sprintf()` nie wydrukuje liczby zmiennoprzecinkowej bez dodatkowych zabiegów. Dlatego taką liczbę konwertujemy najpierw do klasy `String` zdefiniowanej w ramach „języka Arduino”. Wykorzystujemy do tego konstruktor, który pozwala dodatkowo wyznaczyć precyzję konwersji. Ta klasa udostępnia metodę `c_str()` zwracającą typowy wskaźnik języka C na łańcuch tekstowy `const char*`, który to łańcuch znajduje się w buforze klasy `String`. A ten wskaźnik jest już akceptowany przez funkcję `sprintf()`. Zainteresowanych problemem zachęcam też do przeczytania artykułów „How to sprintf a float with Arduino,” ([link](#)) oraz „Do you know Arduino? – sprintf and floating point” ([link](#)).

Zadanie 1: Program wykorzystujący czujniki DS18B20

Przygotuj program, który uczyńi zestaw laboratoryjny małą „stacją pogodową”. Na wyświetlaczu LCD ma prezentować temperaturę wewnętrzną (czujnik wewnątrz zestawu) i zewnętrzną (sonda na przewodzie). Ponadto program powinien pamiętać wartość maksymalną i minimalną temperatury mierzonej przez sondę na przewodzie. Dodatkowo, dioda RGB, może sygnalizować czy jesteśmy w strefie komfortu temperaturowego, lub jest za gorąco, lub za zimno.

Jest to zadanie przykładowe. Prowadzący może zmodyfikować lub zmienić treść zadania.

2 Moduły i własne biblioteki programowe

2.1 Podział kodu na moduły

W dotychczasowych ćwiczeniach korzystaliśmy z bibliotek przygotowanych dla różnych urządzeń, na przykład, wyświetlacza LCD czy czujnika temperatury. Można też zadawać sobie pytanie, czy wszystko należy zawsze pisać „od zera.”? Czy muszę przepisywać do nowego szkicu (pliku z rozszerzeniem „.ino”) wielokrotnie wykorzystywany kod? Oczywiście nie.

Kod 2 prezentuje program, którego zadaniem jest miganie diodami. Obsługę migania pojedynczą diodą wydzieleno do osobnej klasy `LedBlinker` implementowanej w C++, której deklaracja znajduje się w pliku `LedBlinker.h` (plik prezentowany jako kod 3), a implementacja metod w pliku `LedBlinker.cpp` (plik prezentowany jako kod 4).

Kod 2: Program korzystający z klasy `LedBlinker`.

```

1 #include "LedBlinker.h"
2
3 #define LED_RED 6
4 #define LED_BLUE 3
5
6 LedBlinker redBlinker;

```

```

7 LedBlinker blueBlinker;
8
9 void setup()
10 {
11     redBlinker.init(LED_RED, 1000, 2000);
12     blueBlinker.init(LED_BLUE, 1000, 3000);
13 }
14
15 void loop()
16 {
17     redBlinker.runMeInLoop();
18     blueBlinker.runMeInLoop();
19
20     /*YOUR CODE*/
21 }

```

Kod 3: Plik *LedBlinker.h.*

```

1 #ifndef LEDBLINKER_H_
2 #define LEDBLINKER_H_
3
4 #include "Arduino.h"
5
6 typedef unsigned long Time;
7
8 class LedBlinker
9 {
10 public:
11     void init(byte led, Time onPeriod, Time offPeriod);
12     void runMeInLoop();
13
14 private:
15     byte led_;
16     Time onPeriod_;
17     Time offPeriod_;
18     Time lastChangeTime_;
19     byte ledState_;
20 };
21
22 #endif

```

Kod 4: Plik *LedBlinker.cpp.*

```

1 #include "LedBlinker.h"
2
3 void LedBlinker::init(byte led, Time onPeriod, Time offPeriod)
4 {
5     led_ = led;
6     onPeriod_ = onPeriod;
7     offPeriod_ = offPeriod;
8     lastChangeTime_ = 0;
9     ledState_ = LOW;
10    pinMode(led, OUTPUT);
11    digitalWrite(led, LOW);
12 }
13
14 void LedBlinker::runMeInLoop()
15 {
16     Time time = millis();
17     if (ledState_ == LOW)
18     {
19         if (time >= lastChangeTime_ + offPeriod_)
20         {
21             ledState_ = HIGH;
22             digitalWrite(led_, HIGH);
23             lastChangeTime_ = time;
24         }
25     }
26     else
27     {
28         if (time >= lastChangeTime_ + onPeriod_)
29         {
30             ledState_ = LOW;
31             digitalWrite(led_, LOW);
32             lastChangeTime_ = time;
33         }
34     }
35 }

```

Jeśli chodzi o kod, to zwróćmy uwagę, że w pliku głównym programu (kod 2), umieszczono dyrektywę `#include`, podobnie jak robi się to dla bibliotek, która włącza do programu deklarację klasy `LedBlinker`. Ponadto, w pliku `LedBlinker.h` (kod 3) włączono, poprzez dyrektywę `#include "Arduino.h"`, plik nagłówkowy z informacjami, pozwalającymi w kodzie biblioteki wykorzystywać elementy „języka Arduino” takie, jak identyfikatory (stałe) `HIGH`, `LOW`, funkcje `pinMode()`, `digitalWrite()`, `millis()`, itd.

Pewnego wyjaśnienia wymaga potrzeba wywoływania cyklicznego metody `runMeInLoop()`. Jak widać z implementacji klasy, przy wywołaniu metody `init()`, podajemy, jak długo dioda ma być zaświecona, a następnie jak długo zgaszcza. Nie wykorzystujemy tu funkcji zawieszających wykonanie programu typu `delay()`, tylko w metodzie `runMeInLoop()` wykonujemy sprawdzenie, czy upłynął wymagany czas do przełączenia diody, przełączamy diodę jeśli upłynął, i natychmiast oddajemy sterowanie. Niczego nie blokujemy i na nic nie czekamy w tej funkcji. Wymaga to jednak okresowego (cyklicznego) wywoływania tej funkcji. Takie rozwiązanie pozwala na miganie wieloma diodami w tym samym czasie, z różnymi okresami przełączeń, i jeszcze znajdzie się czas na kod realizujący inne funkcjonalności programu.

Oba pliki z klasą `LedBlinker` umieszczone są w tym samym katalogu z głównym plikiem programu. Środowisko programistyczne Arduino IDE, po otwarciu pliku głównego programu (szkicu), automatycznie otworzyło też pliki klasy `LedBlinker`.

```
#include "LedBlinker.h"

#define LED_RED 6
#define LED_BLUE 3

LedBlinker redBlinker;
LedBlinker blueBlinker;

void setup()
{
    redBlinker.init(LED_RED, 1000, 2000);
    blueBlinker.init(LED_BLUE, 1000, 3000);
}

void loop()
{
    redBlinker.runMeInLoop();
    blueBlinker.runMeInLoop();

    /*YOUR CODE*/
}
```

Rysunek 2: Okno Arduino IDE z wszystkimi plikami projektu w kolejnych zakładkach.

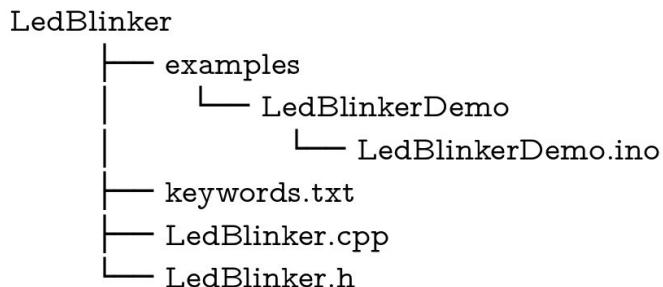
Kod kompliuje się i wgrywa poprawnie do zestawu laboratoryjnego.

2.2 Biblioteki

Przygotowaliśmy kod, który może być wykorzystywany wielokrotnie. Ale w rozwiązaniu z poprzedniego paragrafu musimy za każdym razem kopiować plik do nowego projektu. Kod ten można transformować do biblioteki, która zostanie zainstalowana wraz z innymi bibliotekami dostępnymi w Arduino IDE i można będzie z niej korzystać w dowolnym programie (szkicu) w Arduino IDE.

Tworzenie biblioteki opisane jest w dokumentacji Arduino „[Writing a Library for Arduino](#)” ([link](#)). Proszę zapoznać się z tym dokumentem. Właściwe tworzenie biblioteki znajduje się w drugiej części dokumentu.

Postępując zgodnie z powyższą instrukcją przygotowano strukturę plikową jak na rysunku 3.



Rysunek 3: Pliki w katalogu biblioteki `LedBlinker`.

Plik `LedBlinkerDemo.ino` to nasz program przedstawiony jako kod 2, tylko nazwę zmieniono na bardziej adekwatną do obecnej roli. Zawartość pliku `keywords.txt`, który definiuje jak kolorwać składnię naszej biblioteki w Arduino IDE, prezentuje kod 5.

Kod 5: Plik `keywords.txt`.

```

1| LedBlinker KEYWORD1
2| init KEYWORD2
3| runMeInLoop KEYWORD2

```

Pliki `LedBlinker.h` i `LedBlinker.cpp` zostały przeniesione bez zmian. Całość, włączając w to katalog `LedBlinker`, została zapakowana do pojedynczego pliku zip o nazwie `LedBlinker.zip`. To kończy przygotowanie biblioteki.

Wczytywanie bibliotek w różnych formatach opisuje dokument Arduino „[Installing Additional Arduino Libraries](#)” ([link](#)). Paragraf „*Importing a .zip Library*” tego dokumentu opisuje jak zainstalować bibliotekę dystrybuowaną w pliku „.zip”. Proszę zapoznać się z tym dokumentem.

Zgodnie z powyższym dokumentem, aby zainstalować naszą bibliotekę, trzeba wybrać Menu: *Sketch > Include Library > Add .ZIP Library* i wskazać plik `zip`. Biblioteka zostanie załadowana obok już istniejących. Powinniśmy ją odnaleźć na liście bibliotek po wybraniu jedynie *Sketch > Include Library* i na dysku w katalogu z bibliotekami. Ponadto, w Menu: *File > Examples* powinno pojawić się podmenu `LedBlinker`, a w nim program przykładowy `LedBlinkerDemo`. Po wybraniu tego programu, możemy go uruchomić. Nasza biblioteka, funkcjonuje teraz na równi z innymi.

Zadanie 2: Implementacja własnej biblioteki ze sterownikiem wybranego urządzenia

Przygotuj własną bibliotekę, która będzie służyła do sterowania diodą RGB. Samodzielnie zaprojektuj interfejs programistyczny tej biblioteki, tak aby pozwalał podać, do których pinów podłączona jest dioda, oraz aby pozwalał ustawiać kolory poprzez podanie wartości komponentów RGB oraz przez nazwy kolorów: RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, BLACK, WHITE. Zadbaj o kolorowanie składni w Arduino IDE.

Jest to zadanie przykładowe. Prowadzący może zmodyfikować lub zmienić treść zadania.