Podstawy Systemów Wbudowanych Laboratorium

Laboratorium 6: Kompletny węzeł IoT — TFT, czujniki i sterowanie wentylacją

Autorzy: Angelika Tefelska, Dariusz Tefelski

Spis treści

1 Wprowadzenie **1**

2 Wyposażenie stanowiska **1**

3 Zadanie 1: Lokalny interfejs — wyświetlacz TFT **2**
3.1 Pierwszy ekran 2

4 Zadanie 2: Dane środowiskowe na TFT **4**

5 Zadanie 3: Sterowanie silnikiem DC przez DRV8835 **5**
5.1 Podłączenie sterownika 5
5.2 Prędkość obrotowa i PWM 6

6 Zadanie 4: Sterowanie wentylatorem z Adafruit IO **6**
6.1 Feedy sterujące 7
6.2 Obsługa dwóch subskrypcji 7

7 Zadanie końcowe: Inteligentna wentylacja IoT **9**

8 Podsumowanie **10**

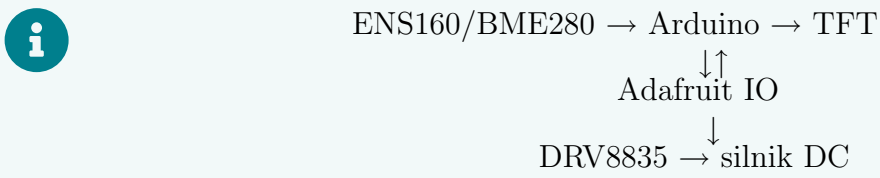
1 Wprowadzenie

Cel ćwiczenia

Celem ćwiczenia jest zbudowanie kompletnego węzła IoT łączącego pomiar parametrów środowiskowych, lokalny interfejs użytkownika, sterowanie elementem wykonawczym oraz komunikację z usługą chmurową. Arduino będzie odczytywało dane z modułu ENS160/BME280, prezentowało je na kolorowym ekranie TFT, sterowało silnikiem DC pełniącym rolę wentylatora oraz wymieniało dane z Adafruit IO.

W poprzednich laboratoriach poszczególne elementy systemu były uruchamiane osobno: poznaliśmy I^2C i SPI, czujniki środowiskowe, PWM oraz komunikację Wi-Fi i MQTT. W tym ćwiczeniu najważniejsza jest **integracja** tych elementów w jeden program.

Architektura systemu



Czujnik dostarcza dane, TFT prezentuje je lokalnie, Adafruit IO umożliwia podgląd i sterowanie zdalne, a silnik DC symuluje wentylację pomieszczenia.

2 Wyposażenie stanowiska

Do wykonania ćwiczenia potrzebne są:

- Arduino UNO R4 WiFi,
- przewód USB,
- wyświetlacz Waveshare 2.8” TFT Touch Shield Rev. 2.1,
- moduł SparkFun ENS160/BME280 oraz przewód Qwiic,
- sterownik silnika DC DRV8835,
- miniaturowy silnik DC,
- płytka stykowa i przewody połączeniowe,
- źródło zasilania silnika wskazane przez prowadzącego,
- komputer z Arduino IDE i dostępem do Adafruit IO.

Co należy zapamiętać?



Silnika DC nie wolno zasilać bezpośrednio z pinu GPIO. Silnik podłączamy wyłącznie przez sterownik DRV8835. Masa zasilania silnika, sterownika i Arduino musi być wspólna.

3 Zadanie 1: Lokalny interfejs — wyświetlacz TFT

W laboratorium wykorzystamy wyświetlacz Waveshare 2.8" TFT Touch Shield Rev. 2.1 o rozdzielczości 320×240 pikseli. Ekran wykorzystuje kontroler ST7789 i komunikuje się z Arduino przez SPI.

Shield wykorzystuje następujące piny Arduino:

Pin	Sygnał	Funkcja
`D10`	LCD_CS	wybór wyświetlacza
`D7`	LCD_DC	dane / komenda
`D9`	LCD_BL	podświetlenie
`D11`	MOSI	dane SPI do wyświetlacza
`D12`	MISO	dane SPI z urządzeń shielda
`D13`	SCK	zegar SPI
`D4`	TP_CS	wybór kontrolera dotyku
`D3`	TP_IRQ	przerwanie panelu dotykowego
`D5`	SD_CS	wybór karty microSD

W tym laboratorium wykorzystujemy tylko funkcję wyświetlania. Panel dotykowy oraz kartę microSD pozostawiamy niewykorzystane.

Planowanie pinów



Shield zajmuje większość cyfrowych pinów Arduino. Dlatego sterowanie silnikiem zostało zaplanowane na wolnych pinach `D2` oraz `D6`. W systemach wbudowanych dobór pinów należy planować przed rozpoczęciem łączenia całego układu.

3.1 Pierwszy ekran

Do obsługi wyświetlacza wykorzystamy biblioteki `Adafruit GFX Library` oraz `Adafruit ST7735 and ST7789 Library`, dostępne w *Library Manager* Arduino IDE.

Zadanie 1.1 — Uruchomienie wyświetlacza



Zainstaluj wymagane biblioteki. Zamontuj shield na Arduino przy odłączonym zasilaniu, a następnie uruchom poniższy program:

```ino
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ST7789.h>

const int TFT_CS = 10;
const int TFT_DC = 7;
const int TFT_BL = 9;
const int TFT_RST = -1;

Adafruit_ST7789 tft(TFT_CS, TFT_DC, TFT_RST);

void setup() {
    pinMode(TFT_BL, OUTPUT);
    digitalWrite(TFT_BL, HIGH);

    tft.init(240, 320);
    tft.setRotation(1);
    tft.fillScreen(ST77XX_BLACK);

    tft.setTextColor(ST77XX_WHITE);
    tft.setTextSize(2);
    tft.setCursor(10, 15);
    tft.print("PSW - Lab 6");
}

void loop() {
}
```

Zmodyfikuj kod tak, aby wyświetlał co najmniej trzy własne napisy w różnych pozycjach. Następnie narysuj prostokąt lub linię oddzielającą nagłówek od części przeznaczonej na dane pomiarowe.

Rysowanie prostych elementów graficznych

Biblioteka `Adafruit_GFX` udostępnia gotowe funkcje do rysowania podstawowych figur. Przykładowo:

- `tft.drawLine(x1, y1, x2, y2, kolor)` — rysuje linię od punktu (x_1, y_1) do (x_2, y_2),
- `tft.drawRect(x, y, szerokosc, wysokosc, kolor)` — rysuje obrys prostokąta,
- `tft.fillRect(x, y, szerokosc, wysokosc, kolor)` — rysuje wypełniony prostokąt.

Przykład linii oddzielającej nagłówek:

```ino
tft.drawLine(0, 45, 319, 45, ST77XX_WHITE);
```

Przykład ramki:

```
tft.drawRect(5, 55, 310, 160, ST77XX_CYAN);
```

Pamiętaj, że po ustawieniu `setRotation(1)` szerokość ekranu wynosi 320 pikseli, a wysokość 240 pikseli.

Nie czyść całego ekranu bez potrzeby

W dalszej części programu nie wywołuj `fillScreen()` przy każdym odświeżeniu pomiarów. Pełne czyszczenie ekranu może powodować widoczne migotanie. Lepiej wyczyścić tylko fragment, w którym zmienia się wartość, np. funkcją `fillRect()`.

4 Zadanie 2: Dane środowiskowe na TFT

W Laboratorium 3 korzystaliśmy już z modułu ENS160/BME280. W tym ćwiczeniu nie uczymy się jego obsługi od początku — czujnik jest źródłem danych dla większego systemu.

Jeżeli złącze Qwiic staje się trudno dostępne po zamontowaniu shielda TFT, podłącz przewód Qwiic przed nałożeniem shielda.

Zadanie 2.1 — Lokalny panel parametrów środowiskowych

Wykorzystaj kod obsługujący ENS160/BME280 przygotowany w Laboratorium 3. Napisz funkcję aktualizującą ekran TFT tak, aby prezentował co najmniej:

- temperaturę w °C,
- wilgotność względną w %,
- ciśnienie w hPa,
- AQI.

Przykładowy układ ekranu:

```
SYSTEM WENTYLACJI

T: 23.4 C
RH: 42 %
p: 1008 hPa
AQI: 2

WENTYLATOR: OFF
```

Odświeżaj dane pomiarowe co kilka sekund. Program nie powinien używać długich opóźnień blokujących dalsze działanie systemu.

5 Zadanie 3: Sterowanie silnikiem DC przez DRV8835

Mikrokontroler nie może bezpośrednio zasilać silnika, ponieważ silnik pobiera znacznie większy prąd i generuje zakłócenia. Do sterowania wykorzystamy mostek H DRV8835.

W trybie PHASE/ENABLE kanał A sterownika wykorzystuje dwa sygnały:

- `APHASE` (AIN1) — kierunek obrotów,
- `AENABLE` (AIN2) — włączenie kanału i sterowanie prędkością za pomocą PWM.

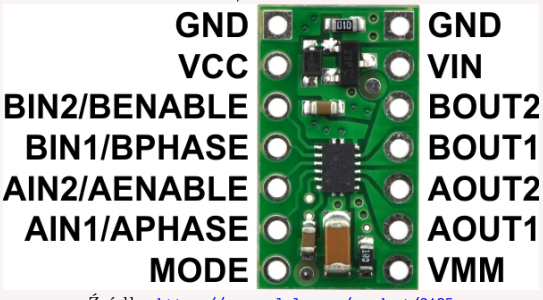
Aby wybrać ten tryb, pin `MODE` musi mieć stan wysoki. W naszym układzie kierunek pozostanie stały, ponieważ silnik symuluje wentylator. Regulować będziemy jedynie jego prędkość.

5.1 Podłączenie sterownika

Zadanie 3.1 — Podłączenie wentylatora

Przy odłączonym zasilaniu połącz sterownik DRV8835 w trybie PHASE/ENABLE zgodnie z tabelą:

DRV8835	Połączenie
GND	GND Arduino
VIN	5V Arduino
VCC	5V Arduino
A O1	jednak nóżka silnika
A O2	druga nóżka silnika
B O1 i B O2	nie podłączaj
VM	nie podłączaj
MD	5V
A INT1	`D2`
A INT2	`D6` – PWN
B INT1 i INT2	nie podłączaj



Źródło: https://www.pololu.com/product/2135

Kanał B sterownika pozostaw niewykorzystany. Przed włączeniem zasilania upewnij się, że Arduino i zasilanie silnika mają wspólną masę.

Co należy zapamiętać?

Przed włączeniem zasilania poproś prowadzącego o sprawdzenie połączeń części mocy. Nie zmieniaj połączeń silnika przy włączonym zasilaniu.

5.2 Prędkość obrotowa i PWM

Do ustawiania prędkości wykorzystamy wartość w procentach. Funkcja może przeliczać ją na zakres PWM 0–255:

```
const int MOTOR_PWM = 6;
const int MOTOR_DIR = 2;

void setFanSpeed(int percent) {
    percent = constrain(percent, 0, 100);
    int pwm = map(percent, 0, 100, 0, 255);

    digitalWrite(MOTOR_DIR, LOW);
    analogWrite(MOTOR_PWM, pwm);
}
```

Zadanie 3.2 — Charakterystyka wentylatora

W funkcji `setup()` skonfiguruj `D2` i `D6` jako wyjścia. Następnie sprawdź działanie silnika dla zadanych wartości prędkości, np. 0%, 30%, 60% i 100%.

PWM nie ustala dokładnej prędkości w obr/min

Wartość 50% oznacza tutaj wypełnienie sterowania PWM, a nie gwarancję, że silnik obraca się z dokładnie połową maksymalnej prędkości. Rzeczywista prędkość zależy m.in. od obciążenia, napięcia zasilania i właściwości silnika.

6 Zadanie 4: Sterowanie wentylatorem z Adafruit IO

W Laboratorium 5 Arduino publikowało temperaturę i odbierało polecenie włączenia diody. Teraz wykorzystamy ten sam mechanizm do sterowania rzeczywistym elementem wykonawczym.

6.1 Feedy sterujące

Zadanie 4.1 — Panel sterowania wentylatorem

W Adafruit IO utwórz dwa nowe feedy:

- `fan_on` — stan wentylatora: 0 lub 1,
- `fan_speed` — zadana prędkość: 0–100.

Na dashboardzie dodaj:

- blok *Toggle* powiązany z `fan_on`,
- blok *Slider* powiązany z `fan_speed`; ustaw zakres 0–100.

Sprawdź na stronach obu feedów, czy zmiany kontrolek powodują zapis nowych wartości.

6.2 Obsługa dwóch subskrypcji

W zadaniu wykorzystamy dwa feedy jednocześnie. Feed `fan_on` będzie odpowiadał za włączenie i wyłączenie wentylatora, natomiast `fan_speed` za wartość prędkości w zakresie 0–100%.

Zadanie 4.2 — Zdalne ON/OFF i regulacja prędkości

Wartości przesyłane przez Adafruit IO są odbierane w funkcjach callback jako obiekt `AdafruitIO_Data`. Aby odczytać z niego wartość liczbową, wykorzystamy funkcję `toInt()`.

Poniższy program zawiera kompletną obsługę obu feedów oraz sterowanie wentylatorem. Zakładamy, że w katalogu szkicu znajduje się plik `config.h` przygotowany w Laboratorium 5, zawierający dane dostępowe do sieci Wi-Fi i Adafruit IO.

```
#include "config.h"

const int MOTOR_PWM = 6;
const int MOTOR_DIR = 2;

AdafruitIO_Feed *fanOn = io.feed("fan_on");
AdafruitIO_Feed *fanSpeedFeed = io.feed("fan_speed");

bool fanEnabled = false;
int fanSpeed = 0;

void setFanSpeed(int percent) {
    percent = constrain(percent, 0, 100);
    int pwm = map(percent, 0, 100, 0, 255);

    digitalWrite(MOTOR_DIR, LOW);
    analogWrite(MOTOR_PWM, pwm);
```

```
}

void updateFan() {
    if (fanEnabled) {
        setFanSpeed(fanSpeed);
    }
    else {
        setFanSpeed(0);
    }
}

void handleFanOn(AdafruitIO_Data *data) {
    int value = data->toInt();

    if (value == 1) {
        fanEnabled = true;
    }
    else {
        fanEnabled = false;
    }

    updateFan();
}

void handleFanSpeed(AdafruitIO_Data *data) {
    int value = data->toInt();

    fanSpeed = constrain(value, 0, 100);
    updateFan();
}

void setup() {
    Serial.begin(115200);

    pinMode(MOTOR_DIR, OUTPUT);
    pinMode(MOTOR_PWM, OUTPUT);
    setFanSpeed(0);

    fanOn->onMessage(handleFanOn);
    fanSpeedFeed->onMessage(handleFanSpeed);

    io.connect();

    Serial.print("Laczenie z Adafruit IO");
    while (io.status() < AIO_CONNECTED) {
```

```
        Serial.print(".");
        delay(500);
    }

    Serial.println();
    Serial.println(io.statusText());

    // Pobranie ostatnich wartosci zapisanych w feedach
    fanOn->get();
    fanSpeedFeed->get();
}

void loop() {
    io.run();
}
```

Zwróć uwagę na dwie funkcje callback:

- `handleFanOn()` odczytuje wartość 0 lub 1 i zmienia zmienną `fanEnabled`,
- `handleFanSpeed()` odczytuje wartość liczbową ze slidera i ogranicza ją do zakresu 0–100 za pomocą `constrain()`.

Funkcja `updateFan()` jest wywoływana po każdej zmianie. Dzięki temu wyłączenie wentylatora nie zmienia zapamiętanej wartości `fanSpeed`; po ponownym włączeniu silnik wraca do ostatniej zadanej prędkości.

Sprawdź działanie układu:

1. ustaw suwak na 40% i włącz wentylator,
2. zwiększ prędkość do 80%,
3. wyłącz wentylator bez zmiany położenia suwaka,
4. ponownie go włącz — powinien wrócić do ostatniej zadanej prędkości.

7 Zadanie końcowe: Inteligentna wentylacja IoT

W ostatnim zadaniu połączymy wszystkie elementy w jeden program.

Zadanie 5.1 — Kompletny system

Przygotuj program spełniający następującą specyfikację:

1. Arduino łączy się z Wi-Fi i Adafruit IO,
2. moduł ENS160/BME280 jest odczytywany cyklicznie, np. co 20 s,
3. na TFT wyświetlane są temperatura, wilgotność, ciśnienie i AQI,
4. do Adafruit IO publikowane są co najmniej temperatura, wilgotność i AQI,
5. feed `fan_on` włącza lub wyłącza wentylator,
6. feed `fan_speed` ustala jego prędkość w zakresie 0–100%,
7. na TFT wyświetlany jest również stan wentylatora oraz zadana prędkość,

8. `io.run()` jest wykonywane regularnie w głównej pętli,
9. odmierzanie czasu realizowane jest przez `millis()`, bez długich `delay()` w normalnej pracy programu.

Na dashboardzie Adafruit IO przygotuj co najmniej:

- wykres lub wskaźnik temperatury,
- wskaźnik AQI,
- przełącznik ON/OFF wentylatora,
- suwak prędkości 0–100%.

Sprawdź, czy zmiana prędkości z dashboardu nie zatrzymuje aktualizacji danych pomiarowych na TFT i w chmurze.

Co integruje ten system?

W jednym programie pracują równocześnie:

- I^2C — odczyt ENS160/BME280,
- SPI — obsługa TFT,
- PWM — regulacja mocy silnika,
- Wi-Fi/MQTT — komunikacja z Adafruit IO,
- lokalny interfejs użytkownika — prezentacja danych na ekranie.

To przykład architektury typowej dla praktycznego systemu wbudowanego: urządzenie mierzy, prezentuje dane, komunikuje się z siecią i steruje elementem wykonawczym.

8 Podsumowanie

Po tych zajęciach powinieneś umieć

- uruchomić kolorowy wyświetlacz TFT korzystający z SPI,
- zaplanować wykorzystanie pinów przy jednoczesnej pracy kilku modułów,
- wyświetlać lokalnie dane z czujników środowiskowych,
- wyjaśnić, dlaczego silnik DC wymaga osobnego sterownika mocy,
- sterować kierunkiem i prędkością silnika za pomocą DRV8835 i PWM,
- sterować elementem wykonawczym z dashboardu Adafruit IO,
- obsłużyć kilka feedów MQTT w jednym programie,
- połączyć czujnik, wyświetlacz, aktuator i komunikację sieciową w jeden nieblokujący system.

Co należy zapamiętać?

Przed opuszczeniem stanowiska wyłącz zasilanie silnika, odłącz Arduino i uporządkuj wszystkie moduły oraz przewody. Nie pozostawiaj danych dostępowych Adafruit IO w plikach udostępnianych publicznie.