Systemy wbudowane

Oprogramowanie MQTT bazującego na ESP8266 oraz STM32 F429ZI w środowisku Arduino

Autorzy
Kacper Korban
Andrzej Ratajczak
https://github.com/BarkingBad/EmbeddedSystems

1. Wstęp

1. Cel projektu

Celem projektu było uzyskanie komunikacji pomiędzy dwoma płytkami za pomocą protokołu MQTT. Projekt został jednak rozszerzony o element badawczy, który polegał na oprogramowaniu płytki ewaluacyjnej z mikrokontrolerem STM32za pomocą Arduino. Jako przykładowe praktyczne użycie komunikacji za pomocą MQTT stworzono zdalny termometr, który wysyła co 5 sekund informację o aktualnym odczycie temperatury. Drugie urządzenie odbiera wartość i za pomocą wbudowanej diody LED informuje o zmianie zmierzonej temperatury - jedno mignięcie oznacza brak zmiany temperatury względem poprzedniego odczytu, dwa mignięcia oznaczają wzrost temperatury, trzy mignięcie odpowiednio oznaczają spadek temperatury.

2. Układ ESP8266

Płytki z modułem ESP8266 wspierane są przez środowisko Arduino. Dzięki interfejsowi WiFi układu ESP8266 możliwa jest implementacja usług stosu IP, które umożliwiają użycie protokołu MQTT. W naszym projekcie płytka z tym modułem jest klientem MQTT oraz jest producentem. To odczytu temperatury wyposażyliśmy płytkę w termometr marki Dallas DS18B20.

3. Układ STM32 F429ZI

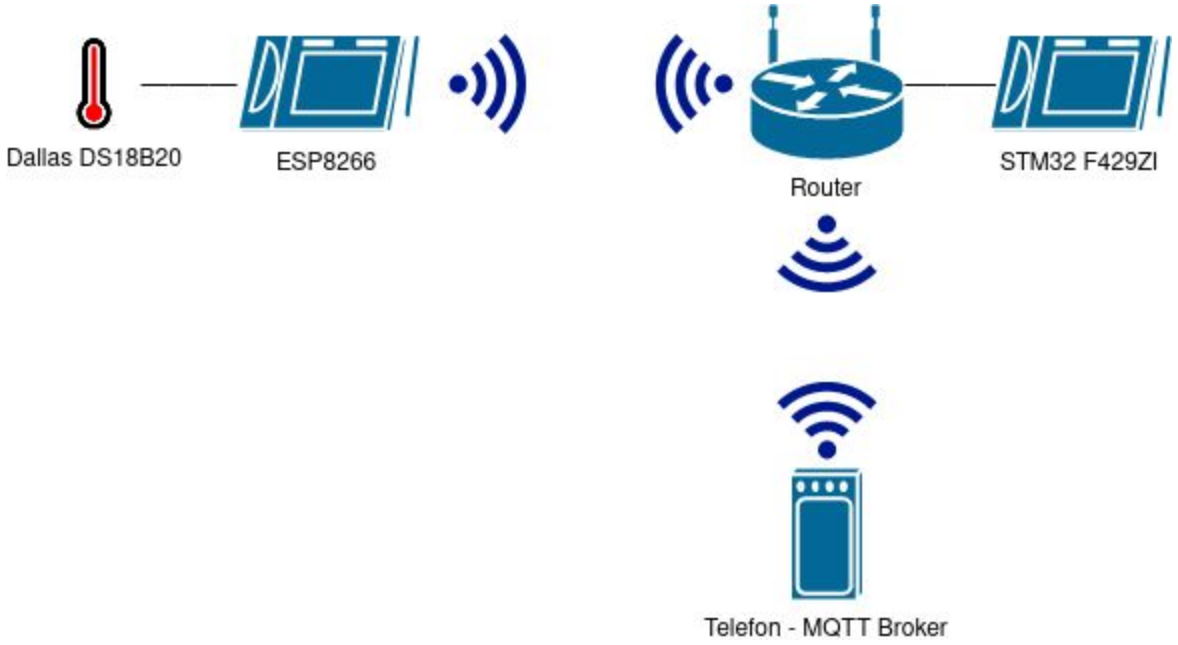
Mikrokontroler STM32 F429ZI należy do rodziny 32 bitowych mikrokontrolerów firmy STMicroelectronics. Oprogramowanie takiej płytki zazwyczaj wymaga odpowiednich sterowników oraz programatora. W naszym projekcie zdecydowaliśmy się skorzystać z STM32duino - projektu który pozwala pisać oprogramowanie w środowisku Arduino na płytki z mikrokontrolerami z rodziny STM32. Płytka wyposażona jest w interfejs Ethernet, przez co do połączenia się z inną płytką potrzebny jest Router, który stworzy nam sieć lokalną do przesyłu danych. Płytka ta również pełni rolę klienta MQTT i jest konsumentem. Za pomocą wbudowanej diody pozwala użytkownikowy kontrolować aktualny stan temperatury odczytywany przez pierwszą płytkę.

4. Broker MQTT

Broker został postawiony na telefonie podłączonym do wspólnej sieci lokalnej z układami wysyłającymi i odbierającymi. Skorzystaliśmy z gotowego rozwiązania, jako że nie było to częścią projektu.

5. Diagram

Poniższy diagram przedstawia architektoniczny schemat współpracy poszczególnych urządzeń

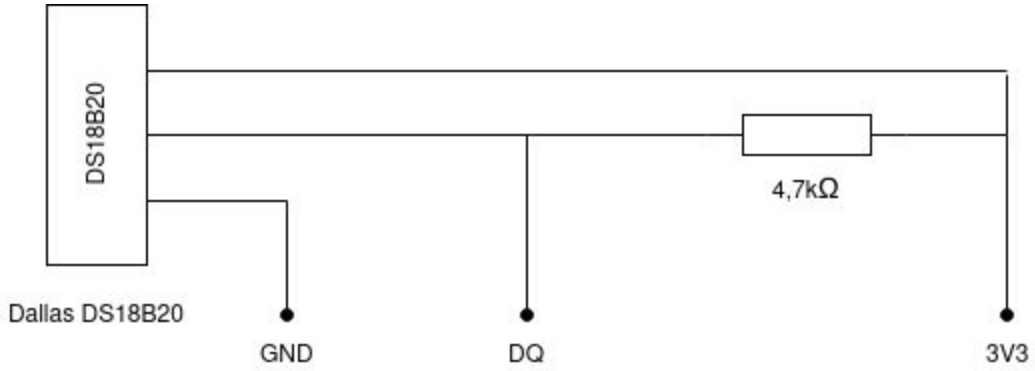


Rysunek 1. Diagram architektury systemu

2. Przygotowanie środowiska dla ESP8266

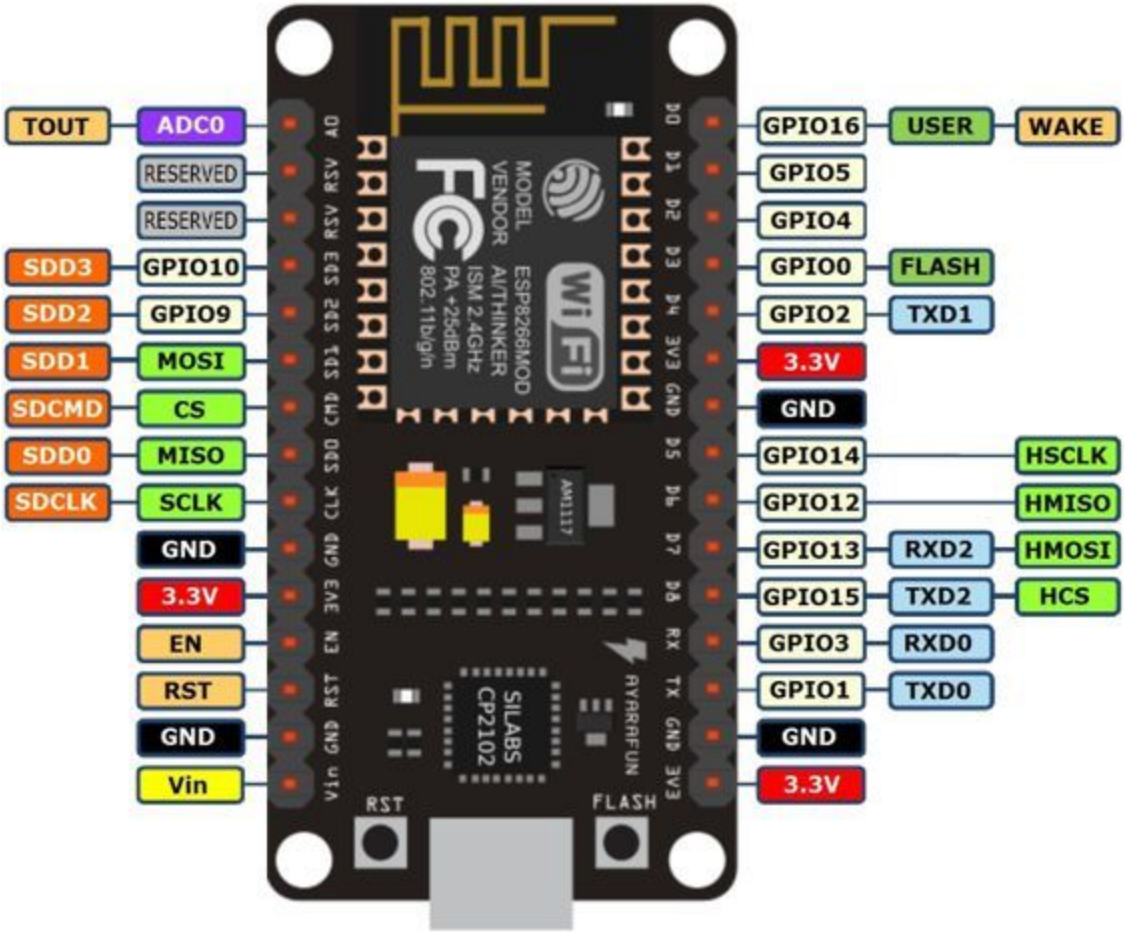
1. Przygotowanie sprzętu

Oprócz samej płytki, potrzebny nam będzie również termometr DS18B20 firmy Dallas oraz rezystor o oporności 4,7 kΩ. Termometr montujemy zgodnie z poniższym schematem.



Rysunek 2. Diagram przedstawiający poprawne podłączenie termometru.

Na diagramie zakładamy że termometr leży częścią wypukłą i możemy odczytać z płaskiej części napis “DS18B20”. Jest to ważne, gdyż w przeciwnym wypadku może dojść do spalenia urządzenia. Masę i napięcie łączymy z płytką odpowiednio węzeł 3V3 z wyjściem na płytce oznaczonym 3V3, a uziemienie z wyjściem GND. Łączenie portu DQ należy skonsultować z poniższym diagramem.



Rysunek 3. Pinout na płytce ESP8266 NodeMCU

Do dyspozycji mamy wszystkie porty oznaczone jako GPIO. Jest to ważne gdyż w Arduino będziemy odwoływać się do numeru po GPIO, nie D. My skorzystaliśmy z portu D2, czyli GPIO4.

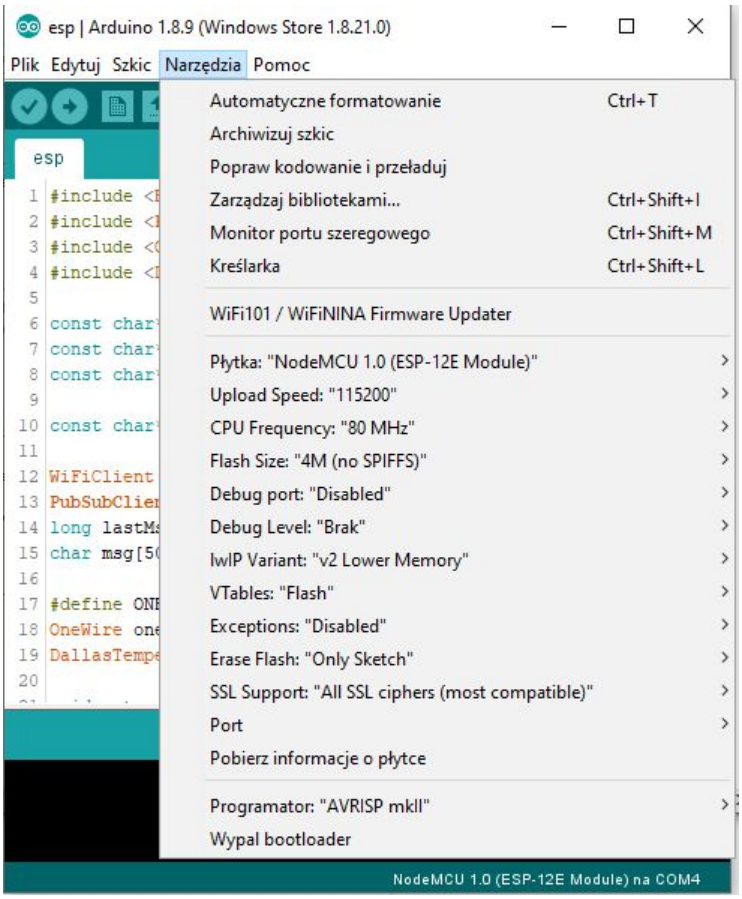
2. Przygotowanie oprogramowania

Należy dołączyć bibliotekę płytki esp8266 w wersji 2.5.2 . Pobieramy ją z menedżera płytek (Narzędzia->Płytka->Menedżer płytek).
Do działania potrzebne będą 3 biblioteki, które można pobrać z oficjalnego repozytorium. Są to

- DallasTemperature v. 3.8.0
- OneWire v. 2.3.5
- PubSubClient v. 2.7.0

Rysunek 4. Zrzut ekranu prezentujący okno menedżera bibliotek

W zakładce narzędzia przed wgraniem oprogramowania na płytkę należy ustawić odpowiednie parametry zgodnie z poniższym zrzutem ekranu.



Rysunek 5. Zrzut ekranu prezentujący konfigurację środowiska pod płytkę ESP8266

3. Oprogramowanie ESP8266

Protokół MQTT opiera się na idei producentów / konsumentów i kanałów (ang. Topics), do których producenci mogą publikować dane a konsumenci mogą subskrybować, w efekcie otrzymując te dane asynchronicznie, jak tylko pojawią się.
ESP8266 w przedstawionej konfiguracji będzie producentem. Poniżej znajduje się omówienie kluczowych kwestii ustawienia połączenia, pełen kod źródłowy znajduje się w załączniku. Jako, że obie płytki zostają zaprogramowane w Arduino, większość kodu jest przenoszalna, więc zamiana roli producenta/konsumenta z STM32 wymaga niewielu poprawek.

Na początku definiujemy kilka zmiennych i stałych globalnych
Zaczynamy od stworzenia klienta WiFi oraz clienta MQTT (PubSub od Publish/Subscribe).

```
WiFiClient espClient;
PubSubClient client(espClient);
```

Następnie inicjalizujemy odczyt z termometru. Korzystamy tutaj, jak już wcześniej wspomniano, z numer GPIO, czyli 4.

```
#define ONE_WIRE_BUS 4 //D2 ON BOARD
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Definiujemy stałe odpowiedzialne za ustanowienie połączenia z siecią. SSID i password to dane dostępowe do access pointu sieci lokalnej, natomiast mqtt_server to adres IP urządzenia, które jest brokerem MQTT.

```
const char* ssid = "SSID";
const char* password = "password";
const char* mqtt_server = "A.B.C.D";
```

Do inicjalizacji połączenia i odczytu z sensorów skorzystamy z poniższych metod. Obsługę błędów można znaleźć w kodzie źródłowym.

```
WiFi.begin(ssid, password);
client.setServer(mqtt_server, 1883); // Korzystamy z portu 1883
sensors.begin();
```

Połączenie z brokerem uzyskujemy za pomocą metody. Ponownie, należy obsłużyć wyjątki.
client.connect(“client_name”)

Na koniec pozostaje nam przygotować główną pętlę do publikowania wiadomości.

```
const char* topic = "outTopic";
long lastMsg = 0;
char msg[50];

void loop() {
  client.loop(); // Funkcja, która odnawia połączenie, sprawdza subskrypcje, publikuje dane

  long now = millis();
  if (now - lastMsg > 5000) {
    lastMsg = now;
    sensors.requestTemperatures();
    double tmp = sensors.getTempCByIndex(0);
    snprintf (msg, 75, "%lf", tmp); // Funkcja publish przyjmuje tylko tekst, więc trzeba
                                    // przekonwertować dane
    Serial.println(msg);
    client.publish(topic, msg);
  }
}
```

4. Przygotowanie środowiska dla STM32 F429ZI

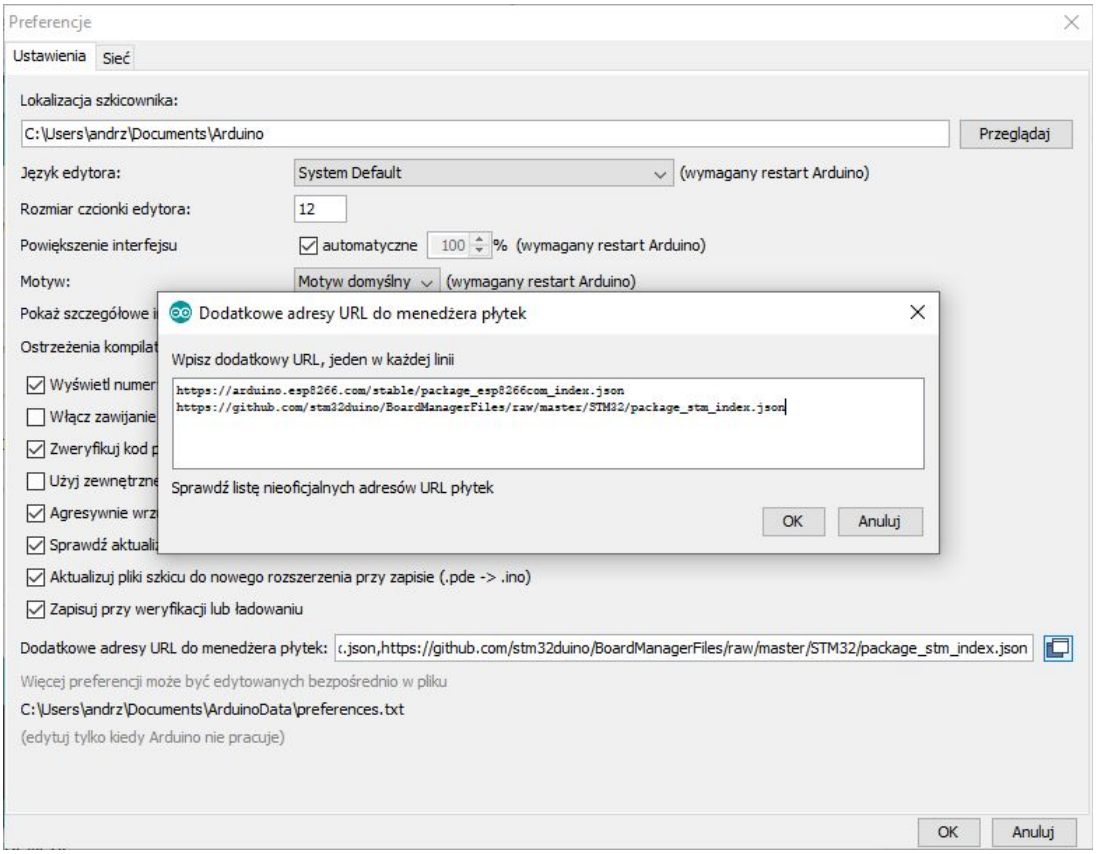
Korzystamy z STM32duino, które jest otwartoźródłowym oprogramowaniem. Oficjalne repozytorium można znaleźć w serwisie github
https://github.com/stm32duino/Arduino_Core_STM32

Najłatwiejszy sposób na korzystanie z bibliotek w ArduinoIDE to pobranie ich przez wbudowane narzędzie do zarządzania paczkami. Najpierw jednak trzeba dodać repozytorium STM32duino w zakładce Plik->Preferencje
https://github.com/stm32duino/BoardManagerFiles/raw/master/STM32/package_stm_index.json

Przechodzimy ponownie do menedżera płytek (Narzędzia->Płytka->Menedżer płytek) i instalujemy bibliotekę STM32 Cores w wersji 1.7.0

Następnie instalujemy paczki przy użycia managera. Potrzebne będą nam

- STM32duino Ethernet v. 1.0.5
- STM32duino LwIP v. 2.1.2
- PubSubClient v. 2.7.0



Rysunek 6. Dodanie repozytorium STM32duino

Jeżeli chcielibyśmy podłączyć termometr do naszej płytki to należy dołączyć dwie paczki, takie same jak dla ESp8266, czyli

- OneWire v. 2.3.5
- Dallas Temperature v. 3.8.0

Schemat podłączenia termometru jest identyczny, jak w ESP8266, węzeł DQ należy skonsultować ze schematem pinout od oficjalnego wydawcy płytki.

Twórcy STM32duino Ethernet wymagają zależności do biblioteki STM32duino LwIP, jednak menager nam tego automatycznie nie pobierze. Po pobranie oddzielnie STM32duino Ethernet oraz STM32duino LwIP należy przenawigować do katalogu bibliotek oraz skopiować zawartość katalogu LwIP do katalogu Ethernet.

5. Oprogramowanie STM32 F429ZI

W naszej architekturze SMT32 jest konsumentem kanału protokołu MQTT. Jeżeli chcielibyśmy, aby był producentem wystarczy kilka prostych zmian które można skonsultować z kodem producenta ESP8266. Poniżej znajdują się najważniejsze linie kodu wraz z krótkim omówieniem. Cały kod źródłowy można znaleźć w załączniku.

Zmienne globalne

```
const char topic[] = "outTopic";
const char server[] = "A.B.C.D";
```

Adres IP i kanał MQTT

```
const float epsilon = 0.01;
float lastMsg = 0;
```

Stała i zmienna pomocna w obsłudze

W podobny sposób jak na ESP8266 deklarujemy klienta MQTT

```
EthernetClient ethClient;
PubSubClient client(ethClient);
```

W bloku setup muszą znaleźć się poniższe instrukcje. Oczywiście należy obsłużyć sytuacje wyjątkowe

```
pinMode(LED_BUILTIN, OUTPUT); // Inicjalizacja diody wbudowanej w płytkę
client.setServer(server, 1883); // Ustawienie adresu i portu klienta MQTT
Ethernet.begin(); // Otwarcie połączenia ethernetowego
client.setCallback(callback); // Ustawienie wskaźnika na funkcję callback obsługującą otrzymane komunikaty
```

Blok loop odpowiedzialny jest za utrzymanie połączenia. Znajdą się tam poniższe instrukcje

```
while (!client.connected()) {
  if (!client.connect("Nucleo-144"))
    delay(1000);
  }
}
client.subscribe(topic); // Subskrypcja MQTT kanału topic
client.loop(); // Funkcja odpowiedzialna za odpytanie servera MQTT czy są oczekujące wiadomości
```

Funkcja callback przyjmuje następującą sygnaturę (**char***, byte*, **unsigned int**), gdzie są to kolejno tablica znaków z nazwą kanału MQTT, z którego odebrano komunikat. Następnie tablica bajtów otrzymanej wiadomości oraz długość tablicy.

Nasza funkcja callback

```
void callback(char* topic, byte* payload, unsigned int length) {

  float newMsg = atof((char*) payload); // Konwersja na wartość typu float
  if(newMsg != 0.0) {
    float diff = lastMsg - newMsg;

    if (diff < epsilon) {
      blink(1);
    } else if(diff < 0) {
      blink(2);
    } else {
      blink(3);
    }
    lastMsg = newMsg;
  }
}
```

Funkcja pomocnicza blink umożliwiająca komunikację płytki z użytkownikiem

```
void blink(int times) {

  for(int i = 0; i < times; i++) {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
  }
}
```

6. Działanie

Jak już wcześniej wspomniano, powstały układ ma działać w formie zdalnego termometru. Esp8266 odpowiada za odczyt temperatury i wysłanie jej na kanał protokołu MQTT. Z kolei STM32 jest odbiornikiem i sygnalizuje użytkownikowi o zmianach temperatury w otoczeniu Esp 8266 za pomocą diody - jedno mrugnięcie przy nie wykryciu zmiany temperatury od ostatniego pomiaru, dwa mrugnięcia dla rosnącej temperatury, i trzy mrugnięcia dla spadającej temperatury. Wartość minimalnej wykrywalnej różnicy temperatury zależy od rzeczywistej implementacji. Działanie można obejrzeć w przygotowanym materiale wideo, który załączono w sprawozdaniu.

7. Pomiary opóźnienia przesyłu danych

Po ukończeniu projektu pojawił się pomysł zmierzenia czasu przesyłu danych.Skorzystaliśmy z będącego na wyposażeniu laboratorium urządzenia Analog Discovery produkcji Digilent, aby dokonać odpowiednich pomiarów.

Zaczynamy od zmian w kodzie. Musimy skonfigurować na każdej płytce jeden z pinów GPIO, aby móc go wykorzystać do odczytu przy użyciu oscyloskopu. Robimy to ponownie przy użyciu instrukcji.

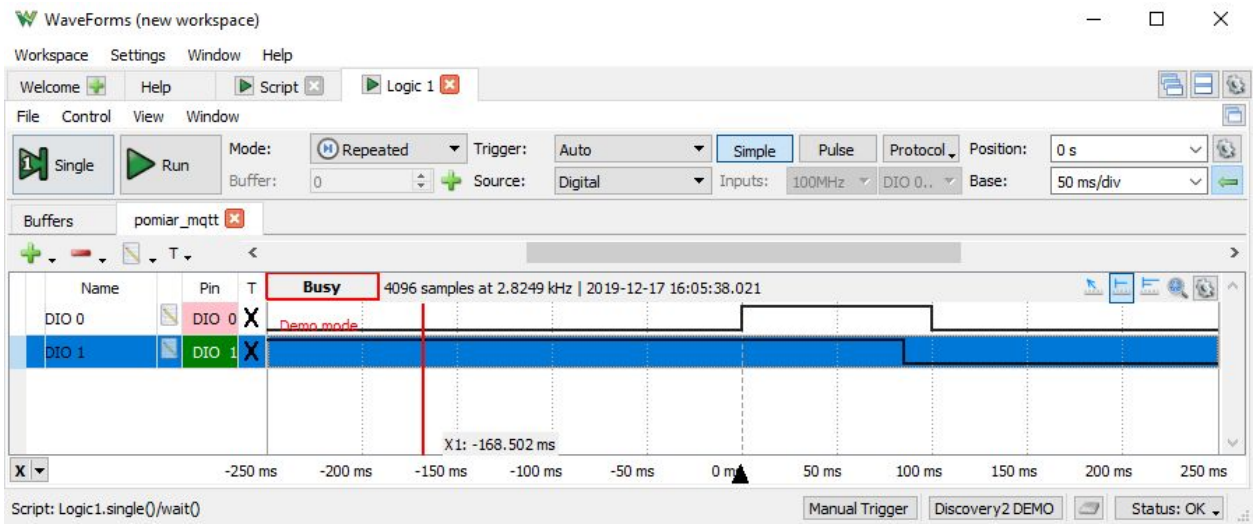
pinMode(pin, OUTPUT); // Gdzie pin to numer odpowiedniego pinu na płytce

Następnie dla danego pinu możemy zmieniać stan

digitalWrite(pin, value) // Gdzie pin to numer odpowiedniego pinu na płytce, a value przyjmuje HIGH lub LOW

Używając tych instrukcji producent wysyłając wiadomość na 100ms zmieniał wyjście pinu w stan wysoki, natomiast konsument za każdym razem kiedy otrzymuje wiadomość zmienia stan na przeciwny.

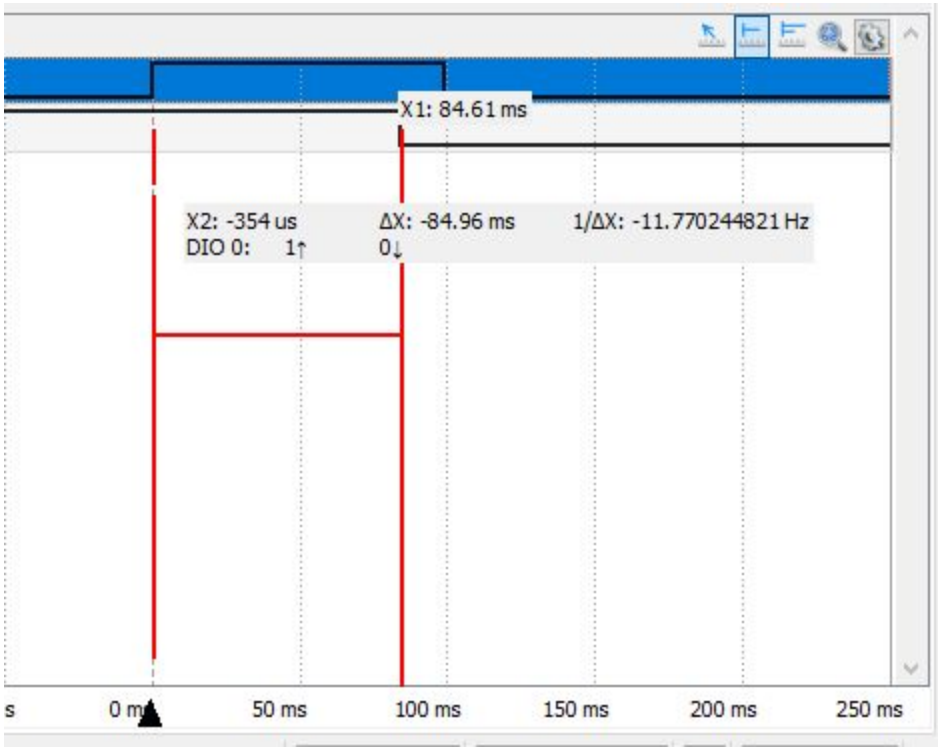
Podłączyliśmy urządzenie do odpowiednich pinów i uruchomiliśmy płytki. Urządzenia kilka razy skomunikowały się ze sobą a my uzyskaliśmy odczyt. Do analizy odczytu można wykorzystać dedykowanego oprogramowania firmy Digilent WaveForms. Po uruchomieniu i zaimportowaniu pliku, należy wybrać ścieżki sygnałowe, którymi odczytywano stany logiczne na pinach. W naszym wypadku jest to DIO 0 i DIO 1.



Rysunek 7. Widok interfejsu programu WaveForms z zaimportowanym odczytem

Widzimy że na ścieżce DIO 0 mamy stan wysoki przez około 100 ms - jest to odczyt z producenta. Z kolei na dolnej ścieżce około 80ms później od zbocza narastającego DIO 0 mamy zmianę stanu - jest to nasz konsument.

Przy użyciu wbudowanych narzędzi możemy sprawdzić dokładny czas który mija od nadania wiadomości do jej odebrania.



Rysunek 8. Pomiar czasu opóźnienia

Jest to około 85 ms. W prezentowanym scenariuszu użycia taki czas można uznać za akceptowalny.