

CANduino i moduł MMcc1000.

Marcin Stolarski

Ver 2.0 22.05.2016

Wstęp

Moduł radiowy MMcc1000 to uniwersalny transceiver (nadajnik i odbiornik) do transmisji danych z wykorzystaniem komputera sterującego takiego jak Arduino.

Parametry układu:

Zasilanie 3.3V

Częstotliwość pracy: 430-440 MHz

Moc nadawcza 10 mW

Bardzo ważną sprawą jest to, że układ zasilany jest z 3.3V i nie jest odporny na napięcie 5V. Dlatego należy sprawdzić napięcie zasilania procesora. Płytką Arduino DUE zasilana jest z 3.3V, więc może być podłączona bezpośrednio do układu MMcc1000, ale płytki Arduino UNO oraz MEGA zasilane są z 5V i pomiędzy płytką a modułem MMcc1000 należy zainstalować dodatkowo konwerter poziomów 3.3V<->5V.

Konfiguracja z płytą Arduino DUE

Moduł MMcc1000 można podłączyć bezpośrednio do płyty Arduino DUE. Przykładowe podłączenie pokazane jest na rysunku oraz w tabeli poniżej.

MMcc1000	Arduino DUE
PALE	D54 (A0)
PDATA	D55 (A1)
PCLK	D56 (A2)
DCLK	D57 (A3)
DIO	D58 (A4)
CHP	D59 (A5)
RSSI	D60 (A6)
VCC	3,3V
GND	GND

Uruchomienie

Przykładowy projekt można znaleźć w katalogu:

arduino-1.6.9-windows\arduino-1.6.9\libraries\canduino_cc1000\examples\cc1000_rtty_data_send\cc1000_rtty_data_send.ino

Zawartość pliku można zobaczyć poniżej.

```
/*
  cc1000 RTTY data send

  modified 22.05.2016
  by Marcin Stolarski
  */

#include "cc1000.h"

//create cc1000 trx object
Cc1000 trx;

// the setup function runs once when you press reset or power the board
void setup() {
  //inicjalise serial monitor
  Serial.begin(9600);
  // initialize digital pin 13 (LED diode) as an output.
  pinMode(13, OUTPUT);

  //inicjalization cc1000 module
  trx.init();
  trx.set_modem_mode(RTTY_MODE);
  trx.set_power(PA_VALUE_0DBM);
  trx.set_deviation(600);
  trx.set_bitrate(300);
  trx.set_frequency(432920000);
  //trx.set_frequency(432920000, VCO_AB, true);
  trx.set_trx_mode(TX_MODE);
  Serial.write("CANduino CC1000 test.\n");
  Serial.write("modem mode:");
  Serial.print(trx.get_modem_mode(), DEC);
  Serial.write(" power: ");
  Serial.print(trx.get_power(), DEC);
  Serial.write(" deviation: ");
  Serial.print(trx.get_deviation(), DEC);
  Serial.write(" bitrate: ");
  Serial.print(trx.get_bitrate(), DEC);
  Serial.write(" frequency: ");
  Serial.print(trx.get_frequency(), DEC);
  Serial.write("\n");
  //trx.set_trx_mode(PD_MODE);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);    // turn the LED on (HIGH is the voltage level)

  String str="";
  str = str + "$$$$$$$$ @@@@@@@@ ##### $$$$$$$$ "
  + "CANduino CC1000 test."
  + " Modem mode: " + trx.get_modem_mode()
  + " Power: " + trx.get_power()
  + " Deviation: " + trx.get_deviation() + " Hz"
  + " Bitrate: " + trx.get_bitrate() + " bps"
  + " Frequency: " + trx.get_frequency() + " Hz"
  + " $$$$$$$$ \n";
  Serial.write((char*)str.c_str());
  trx.send_data(str);
  trx.set_trx_mode(TX_MODE);

  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(3000);
}
```

Na początku dołączana jest biblioteka cc1000 oraz tworzony jest obiekt który będzie reprezentował modułu radiowy (transceiver).

```
#include "cc1000.h"

//create cc1000 trx object
Cc1000 trx;
```

Moduł domyślnie korzysta z pinów modułu Arduino DUE jak na Rysunek 1. Jeśli potrzebna była by inna konfiguracja pinów należy skorzystać z rozszerzonej definicji:

```
Cc1000 trx(pale, pdata, pclk, dclk, dio, chpo, rfadc);
```

Gdzie kolejne zmienne odpowiadają numerom pinów na płycie Arduino.

Następnie w sekcji setup() inicjujemy:

Port szeregowy

```
//inicjalise serial monitor
Serial.begin(9600);
```

Diode migającą

```
// initialize digital pin 13 (LED diode) as an output.
pinMode(13, OUTPUT);
```

Oraz moduł radiowy cc1000

```
//inicjalization cc1000 module
trx.init();
trx.set_modem_mode(RTTY_MODE);
trx.set_power(PA_VALUE_0DBM);
trx.set_deviation(600);
trx.set_bitrate(300);
trx.set_frequency(432920000);
//trx.set_frequency(432920000, VCO_AB, true);
trx.set_trx_mode(TX_MODE);
Serial.write("CANduino CC1000 test.\n");
Serial.write("modem mode:");
Serial.print(trx.get_modem_mode(), DEC);
Serial.write(" power: ");
Serial.print(trx.get_power(), DEC);
Serial.write(" deviation: ");
Serial.print(trx.get_deviation(), DEC);
Serial.write(" bitrate: ");
Serial.print(trx.get_bitrate(), DEC);
Serial.write(" frequency: ");
Serial.print(trx.get_frequency(), DEC);
Serial.write("\n");
//trx.set_trx_mode(PD_MODE);
```

gdzie:

```
trx.init();
```

Inicjalizuje wstępnie moduł radiowy.

```
trx.set_modem_mode(RTTY_MODE);
```

Ustawia trym pracy modemu na RTTY. W przyszłości będą dostępne inne tryby.

```
trx.set_power(PA_VALUE_0DBM);
```

Ustawia moc nadawczą modułu radiowego. Parametr przyjmuje wartości od 0 do 255. Dostępne są predefiniowane wartości:

- PA_VALUE_M30DBM - -30dBm (1uW)
- PA_VALUE_M20DBM - -20dBm (10uW)
- PA_VALUE_M10DBM - -10dBm (100uW)
- PA_VALUE_0DBM - 0dBm (1mW)
- PA_VALUE_10DBM – 10 dBm (10mW, w praktyce około 7.7dBm)

```
trx.set_deviation(600);  
trx.set_bitrate(300);
```

Ustawia szerokości pasma oraz prędkości transmisji. Dla transmisji RTTY zalecane są ustawienia:

- bitrate = 50 bps, deviation 100 Hz
- bitrate = 300 bps, deviation 600 Hz

```
trx.set_frequency(432920000);  
//trx.set_frequency(432920000, VCO_AB, true);
```

Ustawienie częstotliwości pracy modułu w Hz. Do celów eksperymentalnych zalecane jest pasmo ISM (433,050 MHz - 434,790 MHz). Ustawienie częstotliwości nie zawsze jest dokładne ze względu na ograniczenia konfiguracyjne modułu CC1000, dlatego docelowo należy spodziewać się różnicy pomiędzy ustawioną częstotliwością a rzeczywistą częstotliwością.

```
trx.set_trx_mode(TX_MODE);
```

Włączając nadajnik (transmit).

```
Serial.write("CANduino CC1000 test.\n");  
Serial.write("modem mode:");  
Serial.print(trx.get_modem_mode(), DEC);  
Serial.write(" power: ");  
Serial.print(trx.get_power(), DEC);  
Serial.write(" deviation: ");  
Serial.print(trx.get_deviation(), DEC);  
Serial.write(" bitrate: ");  
Serial.print(trx.get_bitrate(), DEC);  
Serial.write(" frequency: ");  
Serial.print(trx.get_frequency(), DEC);  
Serial.write("\n");
```

Odczytanie ustawionych parametrów I wysłanie ich na terminal szeregowy. UWAGA, odczytanie mocy nadawczej „get_power()” możliwe jest tylko, kiedy nadajnik jest włączony „set_trx_mode(TX_MODE)”.

```
trx.set_trx_mode(PD_MODE);
```

Wyłączenie nadajnika I przejście modułu w stan uśpienia (power down).

Następna sekcja loop() pokazuje przykładowe przygotowanie ramki.

```
String str="";  
str = str + "$$$$$$ @@@@@@ ##### $$$$$$ "  
+ "CANduino CC1000 test."  
+ " Modem mode: " + trx.get_modem_mode()  
+ " Power: " + trx.get_power()  
+ " Deviation: " + trx.get_deviation() + " Hz"  
+ " Bitrate: " + trx.get_bitrate() + " bps"  
+ " Frequency: " + trx.get_frequency() + " Hz"  
+ " $$$$$$\n";
```

Ramka została przygotowana w zmiennej typu String, aby ułatwić mieszanie tekstu z danymi liczbowymi.

Na początku znajduje się tak zwana “rozbiegówka” bądź czyli dane które posłużą odbiornikowi do złapania synchronizacji.

```
"$$$$$$$ @@@@@@@@ ##### $$$$$$ "
```

Kolejna sekcja to właściwa ramka danych.

```
+ "CANduino CC1000 test."
+ " Modem mode: " + trx.get_modem_mode()
+ " Power: " + trx.get_power()
+ " Deviation: " + trx.get_deviation() + " Hz"
+ " Bitrate: " + trx.get_bitrate() + " bps"
+ " Frequency: " + trx.get_frequency() + " Hz"
```

Na koniec umieszczono znane zakończenie ramki, aby ułatwić jej znalezienie w innych odebranych danych.

```
" $$$$$$$\n"
```

Wysłanie przygotowanej ramki na terminal szeregowy.

```
Serial.write((char*)str.c_str());
```

Wysłanie ramki przez modem radiowy.

```
trx.send_data(str);
```

Utrzymanie modemu w trybie nadawczym.

```
trx.set_trx_mode(TX_MODE);
```

Normalnie instrukcja „send_data(str)” wyłącza nadajnik co powoduje że po jego włączeniu musi upłynąć trochę czasu zanim jego częstotliwość pracy się ustabilizuje. Zastosowanie komendy `trx.set_trx_mode(TX_MODE)` zaraz po wyłączeniu nadajnika spowoduje jego natychmiastowe włączenie. W rezultacie uzyskamy stabilniejszą pracę nadajnika kosztem większego zużycia prądu elektrycznego.

```
digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
```

Powyższa sekcja zapali diode LED na czas nadawania ramki radiowej.

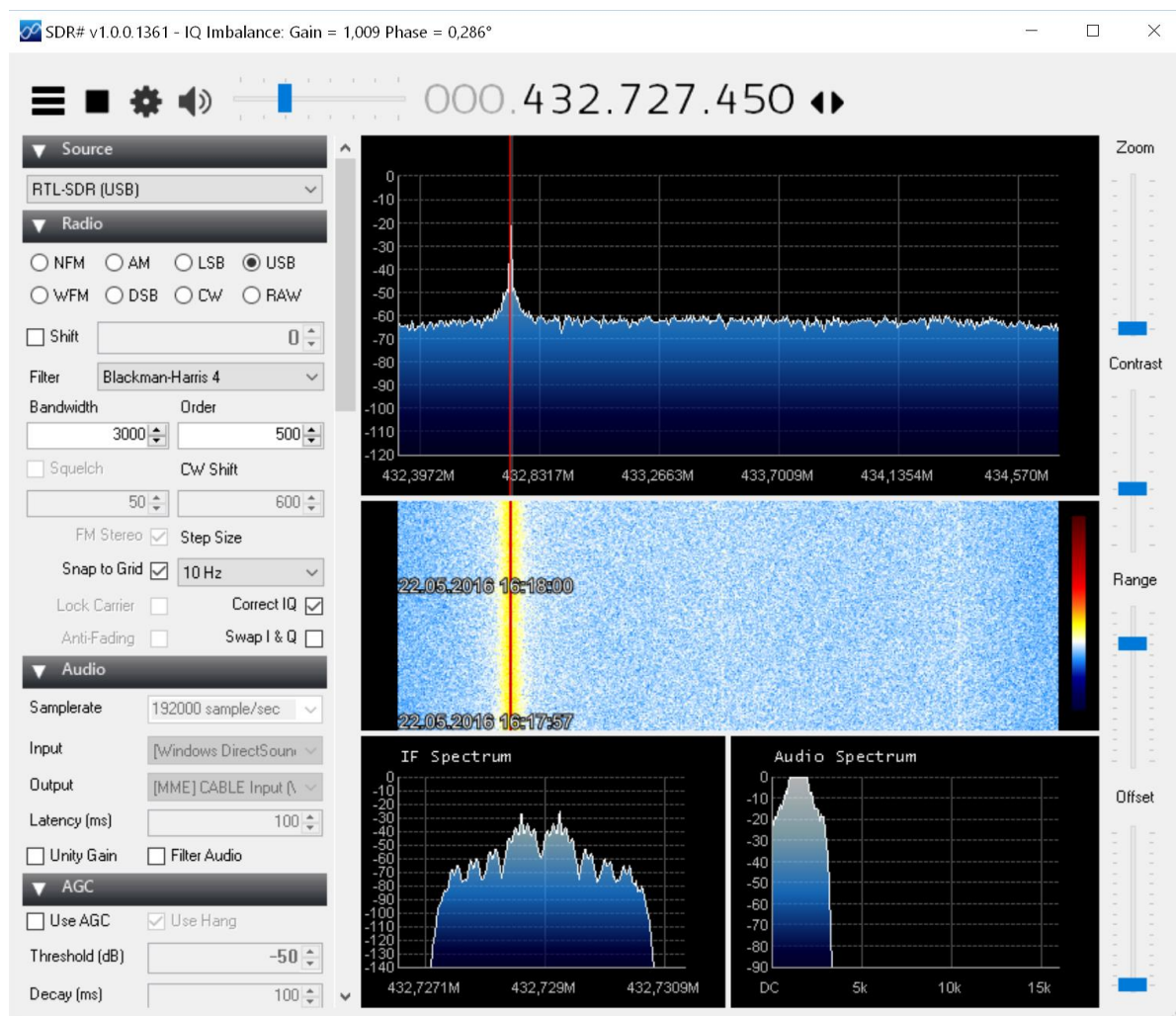
```
delay(3000);
```

Instrukcja `delay(3000)` wstrzyma działanie pętli na 3 sekundy.

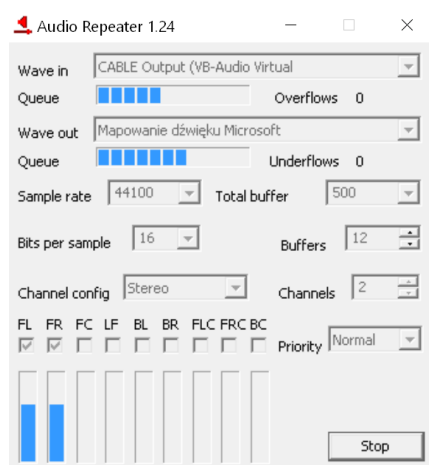
Odbiór danych

Szczegółowa instrukcja odbioru danych znajduje się w pakiecie stacji naziemnej (canduino_gs.zip.). Poniżej znajduje się skrócona instrukcja jej konfiguracji.

Należy uruchomić aplikację SDR# ustawić modulację USB. Jeśli korzystamy z modułu TV-USB (RTL) zalecane jest ustawienie szybkości próbkowania (Sampling rate) na 2,4 MSPS. Moduł RTL nie posiada bardzo stabilnego wzorca częstotliwości. Zwykle po włączeniu potrzebuje on około 10 minut aby jego częstotliwość się ustabilizowała, co związane jest ze stabilizacją jego temperatury. Odebrany sygnał należy przestać za pomocą wirtualnego kabla audio do aplikacji Fldigi. Dodatkowo sygnał audio można odsłuchiwać za pomocą aplikacji Audio Repeater.

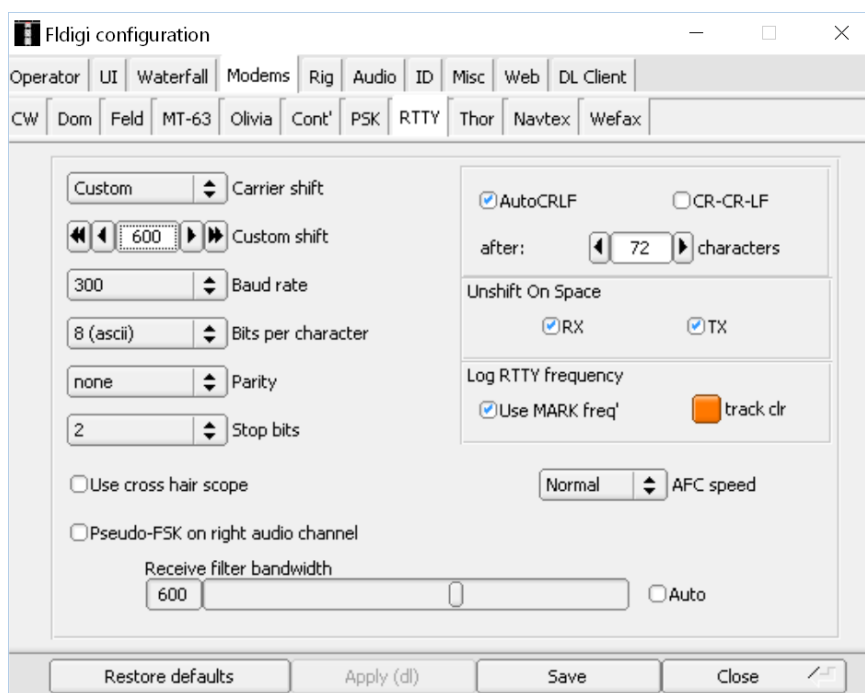


Rysunek 2. Oprogramowanie SDR# podczas odbioru danych z modułu MMcc1000.



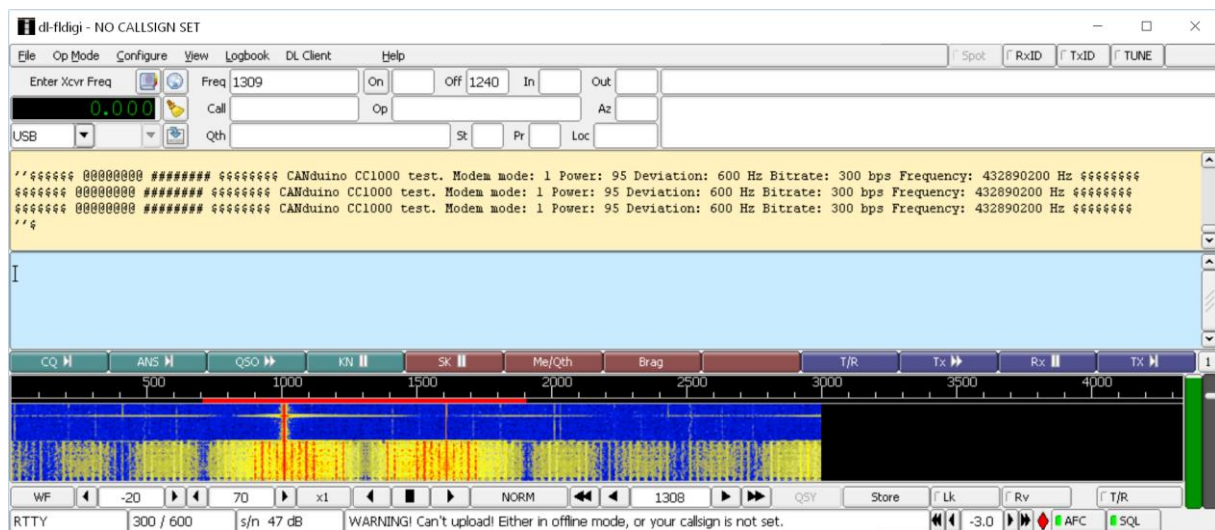
Rysunek 3. Aplikacja Audio Repeater.

Domyślnie załączony przykład nadaje modulacją RTTY (wybrać RTTY custom), bitrate = 300 bps, deviation (Custom shift) 600 Hz. System nadaje dane w standardzie ASCII 8N2 (8 bitów na znak, brak bitu parzystości i 2 bity stopu). Dodatkowo należy ustawić filtr (Receive filter bandwidth) na 600 Hz.



Rysunek 4. Konfiguracja aplikacji Fldigi.

W programie Fldigi na diagramie częstotliwości (waterfall) należy za pomocą myszki wskazać częstotliwość nadawanych danych i w polu odbiorczym powinny zacząć się pokazywać odbierane dane.



Rysunek 5. Aplikacja Fldigi.