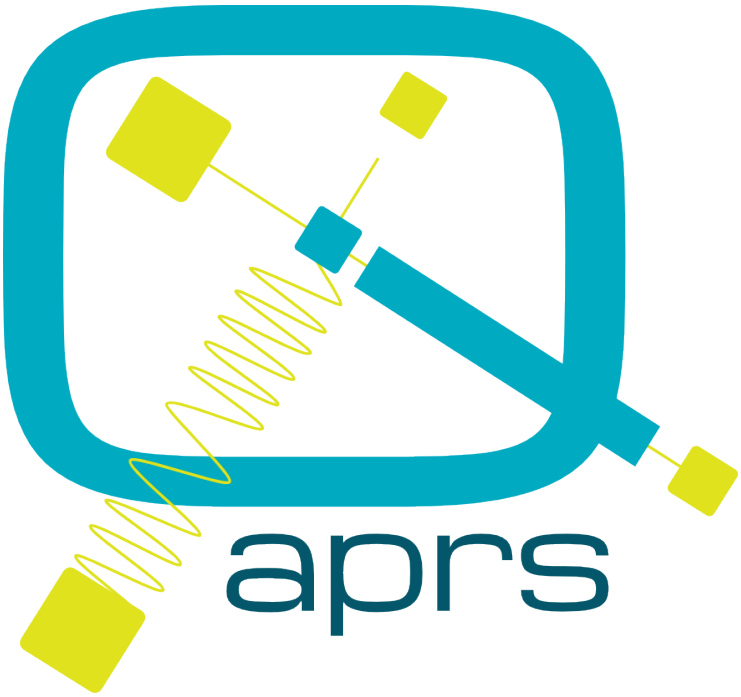
Dokumentacja biblioteki ArduinoQAPRS

Łukasz Nidecki SQ5RWU

01.09.2013



Spis treści

Rozdział 1

Informacje ogólne

ArduinoQAPRS jest biblioteką napisaną w języku C++ na platformę Arduino, mającą na celu uproszczenie implementacji APRS w projektach amatorskich. Człon *Arduino* w nazwie ArduinoQAPRS jest opcjonalny.

1.1 Licencja biblioteki

ArduinoQAPRS jest wolnym oprogramowaniem; możesz go rozprowadzać dalej i/lub modyfikować na warunkach Powszechnej Licencji Publicznej GNU, wydanej przez Fundację Wolnego Oprogramowania - według wersji 2 tej Licencji lub (według twojego wyboru) którejś z późniejszych wersji.

Niniejszy program rozpowszechniany jest z nadzieją, iż będzie on użyteczny - jednak BEZ JAKIEJKOLWIEK GWARANCJI, nawet domyślnej gwarancji PRZYDATNOŚCI HANDLOWEJ albo PRZYDATNOŚCI DO OKREŚLONYCH ZASTOSOWAŃ. W celu uzyskania bliższych informacji sięgnij do Powszechnej Licencji Publicznej GNU.

Z pewnością wraz z niniejszym programem otrzymałeś też egzemplarz Powszechnej Licencji Publicznej GNU (GNU General Public License); jeśli nie - napisz do Free Software Foundation, Inc., 59 Temple Place, Fifth Floor, Boston, MA 02110-1301 USA

1.2 Założenia projektu

Tworzeniu biblioteki ArduinoQAPRS przyświecały następujące cele:

- uwolnienie użytkownika od:
  - konieczności samodzielnego budowania pakietu ax.25
  - samodzielnego kodowania NRZI
  - dbania o zależności czasowe
- obsługa różnych sprzętowych rozwiązań generujących sygnał do nadawania (HAL)
- powstanie wolnego kodu źródłowego (wolnego od *wolność*, nie *powolność* ;)) na licencji GNU

1.3 Użyte słownictwo

string oznacza tablicę znaków zakończoną bajtem o wartości 0. Zobacz także: Null-terminated string

Rozdział 2

Podstawy użycia

2.1 Inicjalizacja

W celu zainicjowania obiektu QAPRS (co częściowo zależy od użytej implementacji sprzętowej i zostanie omówione dalej) należy wywołać metodę **init**:

Listing 2.1: Prototyp metody init

```
1 init(int8_t sensePin, int8_t txEnablePin, char* from_addr,
      uint8_t from_ssid, char* to_addr, uint8_t to_ssid, char*
      relays);
```

na przykład, dla wersji R2R będzie to:

Listing 2.2: Przykład użycia metody init

```
1 QAPRS.init(..., ..., "SQ5RWU", '0', "APZQAP", '0', "WIDE1-1");
```

Pomińmy chwilowo 2 pierwsze parametry i skupmy się na pozostałych.

from_addr string z adresem źródłowym ramek np. nasz znak (w przykładzie: SQ5RWU)

from_ssid SSID stacji źródłowej. Podawany jako pojedynczy, szesnastkowy, znak od '0' do 'f'. Numery te są opisane w specyfkacji APRS.

to_addr string z adresem docelowym ramek. W APRS służy on określeniu wersji trackera.

from_ssid SSID stacji docelowej.

relays string z danymi relayów APRS. Np. "WIDE2-2,WIDE1-1"

Metoda **init** jest metodą wielokrotnie przeciążaną.

2.2 Wysyłanie pakietów

Celem wysłania pakietu należy skorzystać z jednej z 2 metod.

Listing 2.3: Prototyp metody send (jedna z wielu przeładowanych wersji)

```
1 QAPRSReturnCode send(char* from_addr, uint8_t from_ssid, char*
      to_addr, uint8_t to_ssid, char* relays, char* packet_content
      )
```

lub

Listing 2.4: Prototyp metody sendData (jedna z przeładowanych wersji)

```
1 QAPRSReturnCode sendData(char* buffer)
```

QAPRSReturnCode wartość zwracana

QAPRSReturnOK = 0 → Powodzenie generowania pakietu

QAPRSReturnError = 1 → Nieokreślony błąd

QAPRSReturnErrorTimeout = 2 → Przekroczenie czasu oczekiwania na wolny kanał transmisji. ArduinoQAPRSco 100ms sprawdza, czy kanał jest wolny, jeśli upłynie 2000 ms i kanał nie zostanie zwolniony to zwracany jest ten kod.

Metoda 2.3 pozwala na ustawienie innego niż przy inicjalizacji metodą 2.2 znaku itd. przy wysyłaniu pakietu. Metoda 2.4 jest maksymalnie uproszczona i bazuje na domyślnych ustawieniach jakie zostały przekazane przy inicjalizacji.

Np.

Listing 2.5: Przykład użycia metody send

```
1    char from_addr[] = "SQ5RWU";
2    char dest_addr[] = "APZQAP";
3    char relays[] = "WIDE1 1";
4    char packet_buffer[] = "!5215.68N/02057.48ES#";
5  QAPRS.init(..., ..., "SQ5RWU", '0', "APZQAP", '0', "WIDE1-1");
6  QAPRS.send(from_addr, '0', dest_addr, '0', relays,
       packet_buffer);
```

relays string z relayami

UWAGA!
Metoda **send** przyjmuje jako parametr relays przetworzone dane o relayach. Np. WIDE2-2,WIDE1-1 przekazujemy jako "WIDE2 2WIDE1 1"

packet_buffer string z zawartością pakietu do wysłania.

Listing 2.6: Przykład użycia metody sendData

```
1  QAPRS.init(..., ..., "SQ5RWU", '0', "APZQAP", '0', "WIDE1-1");
2    char packet_buffer[] = "!5215.68N/02057.48ES#";
3  QAPRS.sendData(packet_buffer);
```

W przypadku użycia metody **sendData** podajemy jedynie zawartość pakietu APRS do wysłania, natomiast adresy, ssidy itd. są ustawiane bądź to w metodzie **init** bądź z użyciem metod które zostaną omówione dalej (2.3).

2.3 Konfiguracja

2.3.1 Adres źródłowy

Listing 2.7: Prototyp metody setFromAddress

```
1 void setFromAddress(char * from_addr, uint8_t from_ssid);
```

from_addr adres nadawcy. Patrz 2.1

from_ssid ssid nadawcy. Patrz 2.1

2.3.2 Relay'e

Listing 2.8: Prototyp metody setRelays

```
1 void setRelays(char * relays);
```

relays relay'e Patrz 2.1

2.3.3 Wariant (VHF/HF)

Listing 2.9: Prototyp metody setVariant

```
1 void setVariant(QAPRSVariant variant = QAPRSVHF);
```

variant przyjmuje następujące wartości:

QAPRSVHF wartość 1 - VHF - 1200bodów, 1200Hz/2200Hz

QAPRSHF wartość 2 - HF - 300bodów, 1600Hz/1800Hz

2.3.4 Opóźnienie nadawania

Listing 2.10: Prototyp metody setTxDelay

```
1 void setTxDelay(uint16_t txDelay);
```

txDelay opóźnienie pomiędzy podaniem sygnału włączenia nadajnika a rozpoczęciem generowania pakietu w milisekundach (ms). Jeśli nie zostanie wywołana ta metoda to domyślnie zostaje ustawione 300 ms

Rozdział 3

Dostępne implementacje

3.1 QAPRSR2R

Implementacja sprzętowa generująca sygnał AFSK za pomocą 4bitowej drabinki R2R.

Ze względów wydajnościowych nie jest możliwe podanie za pomocą konfiguracji pinów do których podpięta jest drabinka!

Obligatoryjnie wybrano piny od PB0 do PB3.

UWAGA!
Jest to domyślna implementacja sprzętowa biblioteki ArduinoQAPRS.

3.1.1 Inicjalizacja

Listing 3.1: Metody inicjalizacyjne dla QAPRS R2R

```
1 void init(int8_t sensePin, int8_t txEnablePin);
2 void init(int8_t sensePin, int8_t txEnablePin, char * from_addr,
      uint8_t from_ssid, char * to_addr, uint8_t to_ssid, char *
    relays);
```

sensePin pin Arduino służący do wykrywania czy kanał nadawczy jest wolny

- **wartość > 0** stan niski na podanym pinie oznacza, że kanał jest wolny
- **wartość < 0** stan wysoki na podanym pinie (a dokładniej na pinie bez znaku minus) oznacza, że kanał jest wolny
- **wartość = 0** biblioteka nie będzie próbowała sprawdzać czy może nadawać (czy kanał jest wolny) tylko to po prostu zrobi - wykrywanie zajętości kanału bierze na siebie użytkownik

txEnablePin pin Arduino służący do włączenia nadajnika

- **wartość > 0** w celu włączenia nadajnika na pinie zostanie ustawiony stan wysoki
- **wartość < 0** w celu włączenia nadajnika na pinie (a dokładniej na pinie bez znaku minus) zostanie ustawiony stan niski

from_addr 2.1

from_ssid 2.1

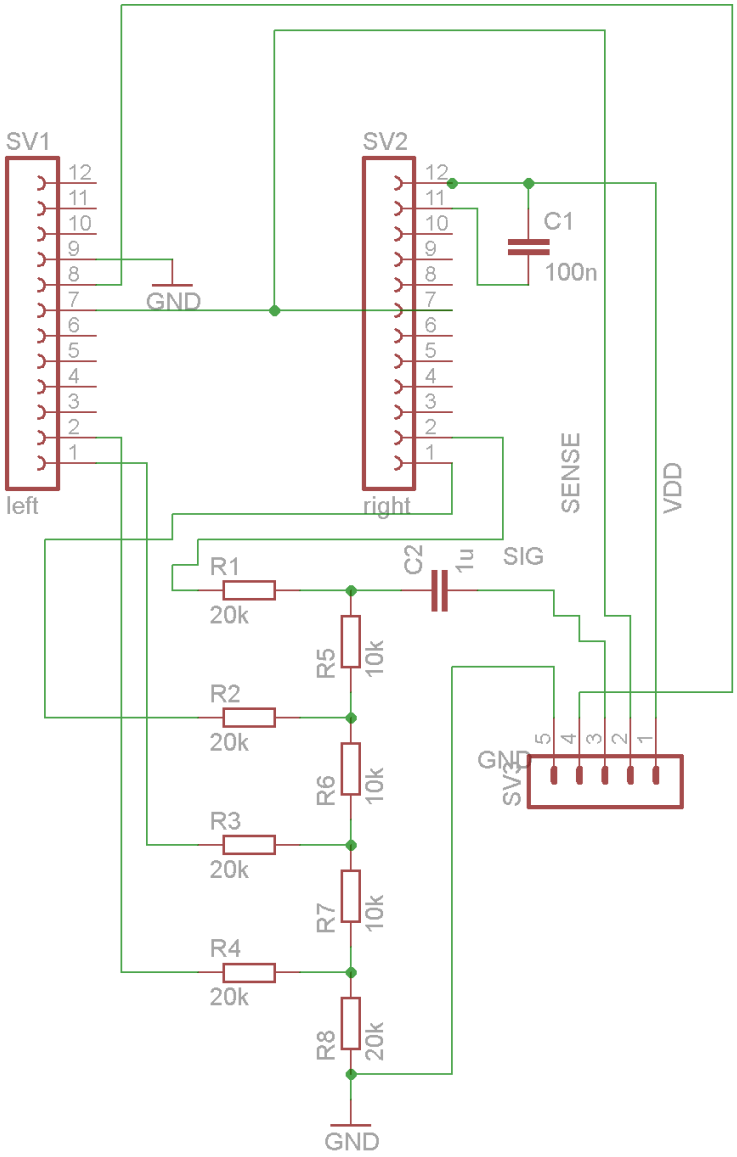
to_addr 2.1

from_ssid 2.1

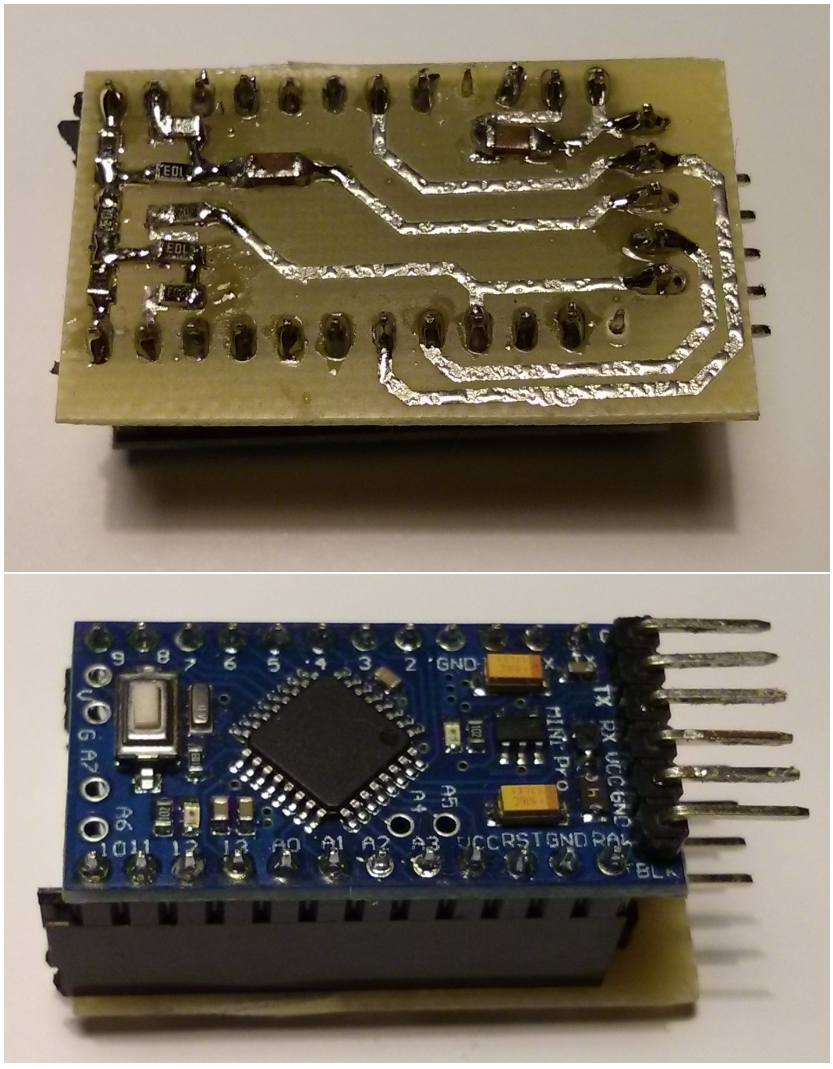
relays 2.1

W razie użycia 1 wariantu metody należy dokonać konfiguracji 2.3

3.1.2 Przykładowa implementacja z użyciem Arduino Pro Mini



Rysunek 3.1: Przykładowa implementacja sprzętowa



Rysunek 3.2: Prototyp

3.1.3 Przykłady użycia

Listing 3.2: Przykład użycia QAPRSR2R

```
#include <Arduino.h>
#include <SPI.h>
#include <ArduinoQAPRS.h>

char * packet_buffer  = "
    \n ";
char from_addr[] = "SQ5RWU";
char dest_addr[] = "APZQAP";
char relays[] = "WIDE2 1";

void setup(){
      // inicjalizacja
      // pin 3 to sensePin [wejscie] - 1 oznacza brak mozliwosci
           nadawania
      // pin 2 to txPin [wyjscie] - stan wyski w momencie
          rozpoczecia nadawania
      QAPRS.init(3,2);
}
void loop() {
      // nadanie paketu typu komentarz
      packet_buffer = ":TEST TEST TEST de SQ5RWU";
      QAPRS.send(from_addr, '0', dest_addr, '0', relays,
          packet_buffer);
      // nadanie pakietu z pozycja i symbolem wahadlowca
      packet_buffer = "!5215.68N/02057.48ES#";
      QAPRS.send(from_addr, '0', dest_addr, '0', relays,
          packet_buffer);
      // nadanie danych pogodowych bez pozycji
      packet_buffer = "
          _07071805c025s009g008t030r000p000P000h00b10218";
      QAPRS.send(from_addr, '0', dest_addr, '0', relays,
          packet_buffer);
      delay(5000);
}
```

Listing 3.3: Przykład użycia QAPRSR2R uproszczone API

```
#include <Arduino.h>
#include <SPI.h>
#include <ArduinoQAPRS.h>

char * packet_buffer  = "

    \n ";


void setup(){
      // inicjalizacja
      // pin 3 to sensePin [wejscie] - 1 oznacza brak mozliwosci
           nadawania
      // pin 2 to txPin [wyjscie] - stan wyski w momencie
          rozpoczecia nadawania
      QAPRS.init(3,2,"SQ5RWU", '0', "APZQAP", '0', "WIDE1-1");
}

void loop() {
      // nadanie paketu typu komentarz
      packet_buffer = ":TEST TEST TEST de SQ5RWU";
      // zmiana adresu źródłowego i ssida
      QAPRS.setFromAddress("SQ5R", '1');
      QAPRS.sendData(packet_buffer);
      // nadanie pakietu z pozycja i symbolem wahadlowca
      packet_buffer = "!5215.68N/02057.48ES#";
      // zmiana adresu źródłowego, ssida i ścieżki
      QAPRS.setFromAddress("SQ5RWU", '2');
      QAPRS.setRelays("WIDE2-2");
      QAPRS.sendData(packet_buffer);
      // nadanie danych pogodowych bez pozycji
      packet_buffer = "
          _07071805c025s009g008t030r000p000P000h00b10218";
      // zmiana ścieżki
      QAPRS.setRelays("WIDE1-1");
      QAPRS.sendData(packet_buffer);
      delay(5000);
}
```

3.2 QAPRSSi4463

UWAGA!
Implementacja wysoce eksperymentalna!
Włączenie wymaga ingerencji w plik *ArduinoQAPRS.h*

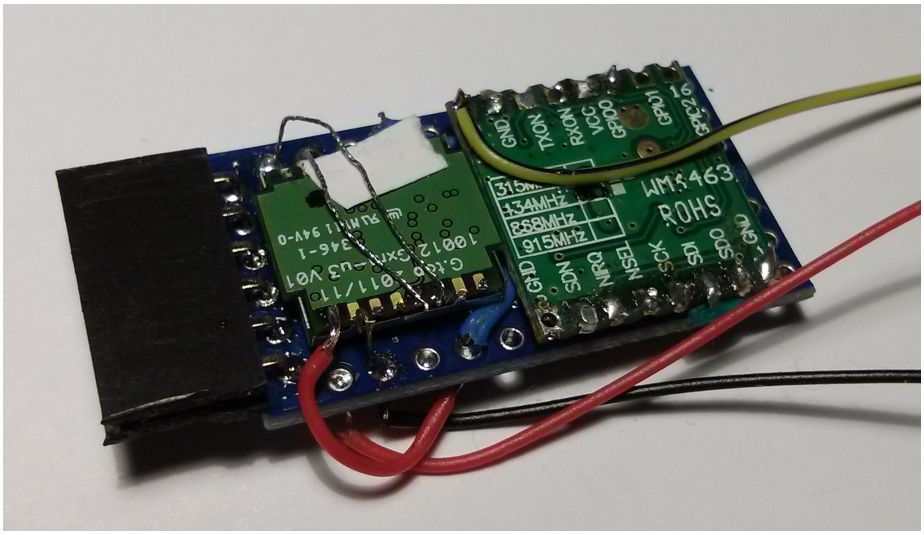
Moduł Si4463 nie umożliwia generowania sygnału AFSK. Można go jednak przełączyć w tryb *Direct*, który umożliwia sterowania binarne dewiacją nadajnika za pomocą jednego z pinów GPIO modułu.

Pierwsze eksperymenty wykazują, że binarne kluczowanie dewiacją daje wstępnie zadowalające rezultaty jeśli chodzi o odbiór przez igate.

3.2.1 Inicjalizacja

QAPRSSi4463 inicjalizuje się w identyczny sposób, jak QAPRSR2R3.1.1, przy czym parametry dotyczące *txEnablePin* i *sensePin* są ignorowane. W kolejnych wersjach zostanie zmieniony sposób inicjalizacji tej implementacji sprzętowej. QAPRSSi4463 używa domyślnie pierwszego dostępnego portu SPI pinu PD.7 do kluczowania nadajnika. W kolejnych wersjach zostanie zaimplementowane konifurowanie tego przy inicjalizacji.

3.2.2 Przykładowa implementacja z użyciem Arduino Pro Mini i "chińskiego" modułu z Si4463 + GPS



Rysunek 3.3: Prototyp

3.3 QAPRSAD9850

UWAGA!
Implementacja wysoce eksperymentalna!
Włączenie wymaga ingerencji w plik *ArduinoQAPRS.h*

Układ scalony AD9850 jest kompletnym scalonym syntezerem DDS. Stosunkowo dużą popularnością wśród krótkofalowców cieszy się tani chińskiej produkcji moduł zawierający układ AD9850, oscylator 125MHz i garść elementów dyskretnych. QAPRSAD9850 powstał jako eksperymentalny układ generowania pakietów APRS poprzez bezpośredią cyfrową syntezę sygnału radiowego dla pasma HF. APRS przewiduje odpowiednie częstotliwości i modulację FSK dla pasma krótkofalowego.

3.3.1 Inicjalizacja

Listing 3.4: Metody inicjalizacyjne dla QAPRSAD9850

```
1 void init(int8_t pinData, uint8_t pinClk, uint8_t pinFq,
      uint32_t outputFrequency, char * from_addr, uint8_t
      from_ssid, char * to_addr, uint8_t to_ssid, char * relays);
```

pinData pin Arduino podłączony do pinu Data w module

pinClk pin Arduino podłączony do pinu Clk w module

pinFq pin Arduino podłączony do pinu Fq w module

outputFrequency częstotliwość w Hz (USB)

from_addr 2.1

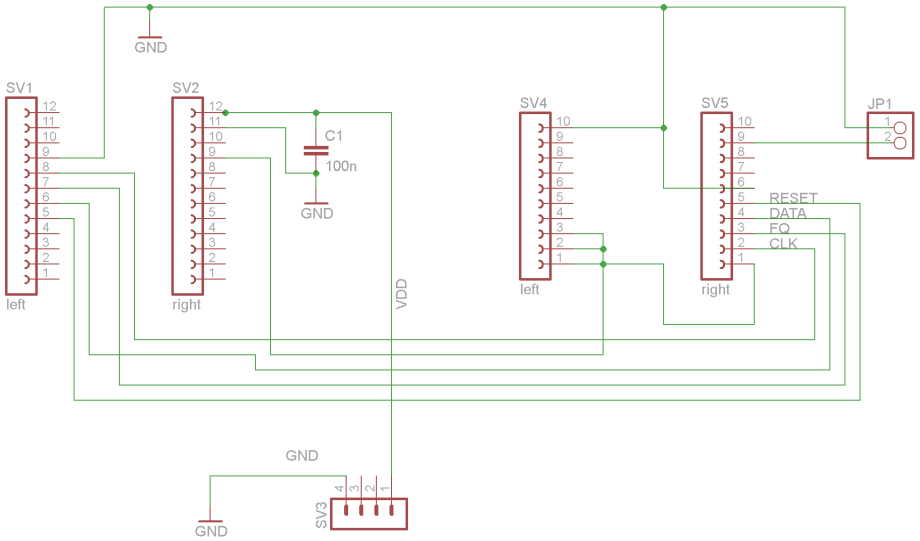
from_ssid 2.1

to_addr 2.1

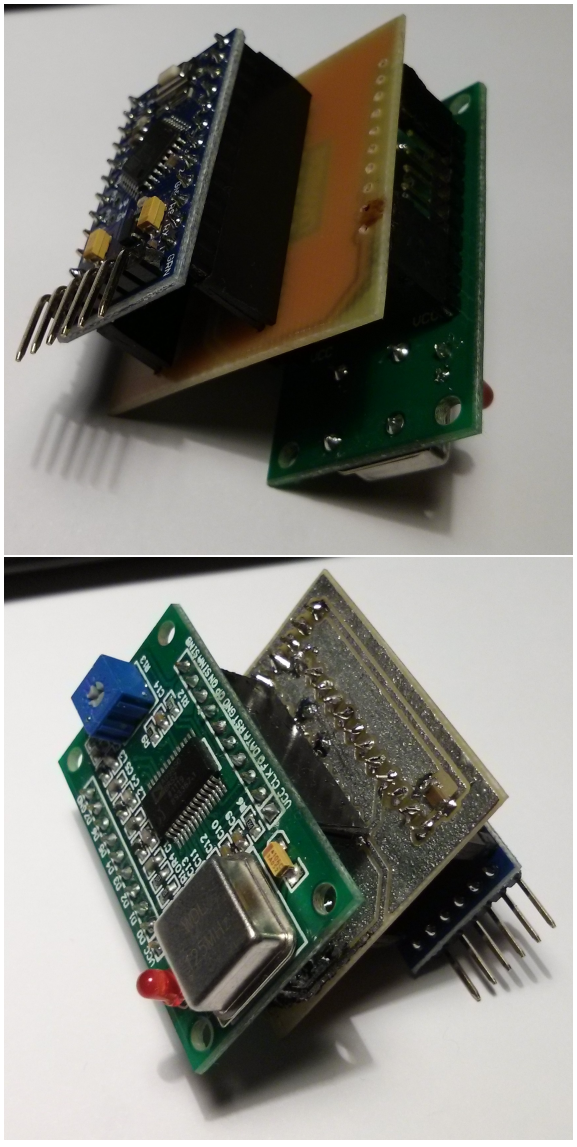
from_ssid 2.1

relays 2.1

3.3.2 Przykładowa implementacja z użyciem Arduino Pro Mini i "chińskiego" modułu z AD9850



Rysunek 3.4: Przykładowa implementacja sprzętowa



Rysunek 3.5: Prototyp

3.3.3 Przykłady użycia

Listing 3.5: Przykład użycia QAPRSAD9850

```
#include <Arduino.h>
#include <SPI.h>
#include <ArduinoQAPRS.h>

char * packet_buffer  = "

    \n ";


void setup(){
   // inicjalizacja
   // data - pin 4
   // clk - pin 2
   // fq - pin 3
   // 10147600 - czestotliwosc bazowa
   QAPRS.init(4,2,3,10147600,"SQ5RWU", '0', "APZQAP", '0', "
       WIDE1-1");

   // przełączenie się na tryb odpowiedni dla HF/KF
   QAPRS.setVariant(QAPRSHF);
}

void loop() {
      // nadanie paketu typu komentarz
      packet_buffer = ":TEST TEST TEST de SQ5RWU";
      // zmiana adresu źródłowego i ssida
      QAPRS.setFromAddress("SQ5R", '1');
      QAPRS.sendData(packet_buffer);
      // nadanie pakietu z pozycja i symbolem wahadlowca
      packet_buffer = "!5215.68N/02057.48ES#";
      // zmiana adresu źródłowego, ssida i ścieżki
      QAPRS.setFromAddress("SQ5RWU", '2');
      QAPRS.setRelays("WIDE2-2");
      QAPRS.sendData(packet_buffer);
      // nadanie danych pogodowych bez pozycji
      packet_buffer = "
          _07071805c025s009g008t030r000p000P000h00b10218";
      // zmiana ścieżki
      QAPRS.setRelays("WIDE1-1");
      QAPRS.sendData(packet_buffer);
      delay(5000);
}
```

Rozdział 4

Adresy, nazwiska, kontakty

Strona z kodem projektu https://bitbucket.org/Qyon/arduinoqaprs/

Strona z dokumentacją http://niezle.info/qaprs

Email do autora qyonek+qaprs@gmail.com

Złożono w systemie LaTeX