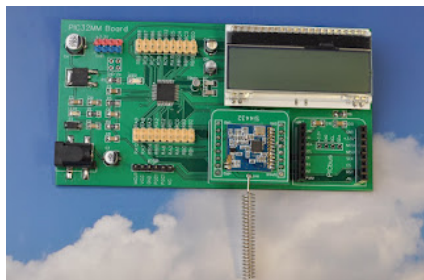


# Mikrokontrolery PIC

Blog jest formą pamiętnika technicznego, upamiętniający moją przygodę z mikrokontrolerami PIC firmy Microchip. "Per aspera ad astra..."

PONIEDZIAŁEK, 11 CZERWCA 2018

## SI4432 moduł RF firmy Silicon Labs - zajęcia praktyczne.



Wreszcie znalazłem chwilę czasu aby przetestować moduły radiowe **RF** oparte na chipsecie **SI4432** firmy **Silicon Labs**. Pomimo , że chipset ten nie jest już rekomendowany przez producenta i zaleca się go zastąpić nowszymi wersjami takimi jak np **SI446x** to jest on nadal bardzo łatwo dostępny i tani. Kompletnie moduły można nabyć poniżej **2 USD**. Warto zauważyć , że inny popularny u nas moduł radiowy **RFM22** firmy **Hoperf** został zbudowany na bazie **SI4432**. Opinie o modułach **SI4432** w necie są bardzo pozytywne, jako podstawowe zalety wymienia się niezawodność transmisji i duże zasięgi. No ale najlepszą metodą zweryfikowania opinii innych osób jest wyrobienie jej sobie samemu, najlepiej przez poznanie modułu w praktyce. Co niniejszym czynię.

Z tego co wyczytałem w necie ogólnie moduły radiowe **RF** sprawiają sporo różnego rodzaju kłopotów osobom , które pragną je poznać i używać. Myślę, że w **98 %** problemy biorą się z błędnej lub niepełnej konfiguracji toru radiowego przez co drastycznie spada zasięg transmisji lub w ogóle jest problem aby nawiązać łączność z innym modulem **RF**. Dodatkowym utrudnieniem jest brak elementarnej wiedzy na temat parametrów transmisji radiowej , skoro nic na ten temat nie wiemy to jak mamy zajarzyć konfigurację toru radiowego ?? . Warto w tym kontekście zapoznać się z podstawowymi pojęciami takimi jak **dewiacja**, **nośna**, **kanał radiowy**, **pasmo**. Warto też mieć pojęcie do czego służy **preambuła**, **whitening**, **kodowanie Manchester**. Warto też wiedzieć jak jest modulowany sygnał, jakie zalety ma modulacja np. **GFSK** . Ponieważ jest to mój pierwszy moduł radiowy z jakim w praktyce przyszło mi się zmierzyć dlatego w pierwszej kolejności szukałem dostępnych materiałów u producenta czyli firmy **Silicon Labs**. Materiały na których oparłem poznanie modułów są zawarte w linkach poniżej artykułu. Szczególnie wart polecenia jest dokument **AN537** w którym opisano w przystępny sposób ważniejsze funkcjonalności modułu. A w dokumencie **AN415** znajdziemy przykład , który można zaimplementować do pierwszej wymiany danych drogą radiową. W sumie trzeba przyznać , że **Silicon Labs** ma bardzo dobrą dokumentację i tyczy to praktycznie całego obszaru portfolio tej firmy w tym **MCU** z rdzeniem **Cortex Mx**

Testy przeprowadzam na moich płytkach developerskich dla **MCU PIC32MM**. Moduły **Si4432** osadziłem na płytkach do gniazda **PICbus** :



Na rewersie płytki znajduje się cyfrowy czujnik temperatury **MCP9808**. Połączenia jakie zastosowałem pomiędzy **PIC32MM** a **Si4432** wyglądają następująco :

**PIC32MM** --> **Si4432**

**RB3** --> **SDI**  
**RA9** --> **SDO**  
**RB8** --> **SCLK**  
**RC6** --> **nSEL**  
**RB9** --> **nIRQ**

A kompletny opis pinów modułu **Si4432** który posiadam na obrazku poniżej :



O MNIE



**PICmajster**

Hobbysta. Miłośnik wszystkiego co dobre , zdrowe , służy ludziom i pokojowi na świecie. Każda chwila życia to jak wygrana w totolotku.... a jeśli tę chwilę przeżyjemy w miłości do wszystkiego co nas otacza to wygrana jest podwójna..... Kontakt: picmajster.blog@gmail.com

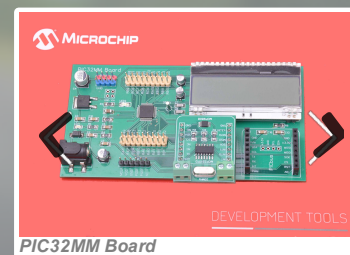
[Wyświetl mój pełny profil](#)

MOJA LISTA BLOGÓW

**Mikrokontrolery ARM**

Jak wyświetlić kolorowy obrazek na wyświetlaczu 1.8 TFT ILI9163.

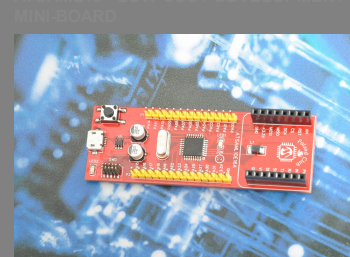
LOW COST DEVELOPMENT



● ● ● ● ● ● ● ●

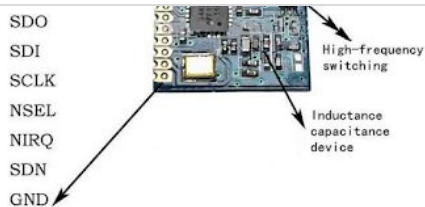


ATSAM11 - LOW COST DEVELOPMENT



ATSAM11 - LOW COST DEVELOPMENT





O rozbudowanych możliwościach **SI4432** niech świadczy ilość dostępnych rejestrów dokładnie **127**. Ta ilość rejestrów może trochę na początku deprymować tym bardziej, że nie jest to mikrokontroler. Wszystko jest jednak kwestią obycia się z tym dobrodziejstwem i nocy nie przespanych z małżonką. Na pewno przy takiej ilości rejestrów łatwo o błąd lub pominięcie czegoś istotnego w konfiguracji sam jestem ciekawy czy uda mi się za pierwszym podejściem uruchomić komunikację pomiędzy dwoma modułami.

Moduł generalnie mnie zafascynował swoimi możliwościami a z ciekawych opcji jakie można na szybko wymienić to m.in **255 kanałów** transmisyjnych w ramach wybranego pasma, **4 bajtowy adres węzła** dla którego jest przeznaczona wiadomość, oprócz nadania adresu węzła mamy dodatkowo możliwość filtrowania wiadomości wykorzystując do tego **4 bajty synchronizacyjne, CRC dla ramki**. Kurcze trudno mi nawet ogarnąć jakie tu mamy olbrzymie możliwości w konfigurowaniu przepływu informacji. Aż mi dech zaparło :) Do pełni szczęścia brakuje jakiś sprzętowych potwierdzeń przesłanych informacji ale tutaj możemy sobie samemu wydumać jakiś mechanizm jeśli będzie taka potrzeba. Pierwszym skojarzeniem jakie się nasuwa na myśl jeśli chodzi o wykorzystanie modułów jest możliwość zbudowania radiowej implementacji **RS485**.

Abstrahując od jakichkolwiek ustawień pierwszym kroczkiem jest zapoznanie się ze strukturą zapisu i odczytu do modułu a tu z pomocą przychodzi nam dokumentacja :

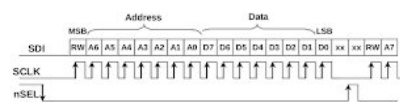


Figure 3. SPI Timing

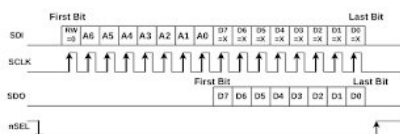


Figure 4. SPI Timing—READ Mode

Z modułem "rozmawiamy" za pomocą **SPI**. Parametrem granicznym dla **SCLK** jest **10 MHz**, warto to mieć na uwadze i nie podkręcać zbytnio zegara **SPI** po stronie **MCU**. Z powyższych timingów wynika w sumie prosta struktura ramki komunikacji z modułem, aby cokolwiek nadać lub odebrać z modułu musimy na linii **nSEL** ustawić stan niski potem wysyłamy pierwszy bajt, który na **7 bitach (A0..A6)** zawiera adres rejestru do którego się odwołujemy a na **8-mym bicie (R/W)** kodujemy informację czy chcemy odczytać dane z rejestru (**R/W=0**) czy zapisać dane do rejestru (**R/W=1**). Drugim przesyłanym bajtem jest dana, zapisywana do rejestru lub odczytywana z niego.

Możemy w tym momencie zająć się zbudowaniem funkcji zapisującej i odczytującej do modułu. Tak abyśmy mogli zacząć jak najszybciej rozmawiać z modułem. Poniżej obraz z kodem tych funkcji.

```

100 void SpiWriteRegister (uint8_t reg, uint8_t value)
101 {
102     /*Select the radio by pulling the nSEL pin to low*/
103     SI4432_nSEL = 0;
104     /*Write the address of the register into the SPI buffer of the MCU
105     (important to set the MSB bit)*/
106     SPI2_ExchangeBit(reg);
107     /*Write the new value of the radio register into the SPI buffer of the MCU*/
108     SPI2_ExchangeBit(value);
109     /*Deselect the radio by pulling high the nSEL pin*/
110     SI4432_nSEL = 1;
111 }
112
113 uint8_t SpiReadRegister (uint8_t reg)
114 {
115     uint8_t rData;
116     /*Select the radio by pulling the nSEL pin to low*/
117     SI4432_nSEL = 0;
118     /*Write the address of the register into the SPI buffer of the MCU
119     (important to clear the MSB bit)*/
120     SPI2_ExchangeBit(reg);
121     /*Write a dummy data byte into the SPI buffer of the MCU. During sending
122     this byte the MCU will read the value of the radio register and save it
123     in its SPI buffer.*/
124     rData = SPI2_ExchangeBit(0xFF); /*Write dummy data into the SPI register*/
125     /*Deselect the radio by pulling high the nSEL pin*/
126     SI4432_nSEL = 1;
127     /*Read the received radio register value and return with it*/
128     return rData;
129 }

```

Do obsługi **SPI** wykorzystałem funkcje automatycznie generowane przez **MCC** czyli najprościej i najszybciej jak się da. W sumie korzystałem tylko z jednej funkcji **SPI2\_Exchange8bit**, która potrafi wysłać bajt po **SPI** i odczytać bajt z bufora sprzętowego **SPI**. **Uwaga !!** standardowa konfiguracja **SPI** jaką otrzymamy dla **Mastera** z konfiguratora **MCC** ustawia w rejestrze **SPIxCON** bit **SMP** na 1. Aby poprawnie rozmawiać po **SPI** z modułem **SI4432** ten bit musi być wyzerowany. Jeśli tego nie zrobimy otrzymamy przekłamanie danych na linii **SDI**. Najszybszym testem czy dane są poprawnie odbierane to odczyt zawartości rejestrów **0x00** i **0x01** modułu **SI4432** :

**SI4432\_REG\_00\_DEVICE\_TYPE** tutaj odczytamy wartość **8**  
**SI4432\_REG\_01\_VERSION\_CODE** tu odczytamy wartość **6**.



## TECHNICAL

### ARCHIWUM BLOGA

- ▶ 2022 (11)
- ▶ 2021 (14)
- ▶ 2020 (14)
- ▶ 2019 (55)
- ▼ 2018 (44)
  - ▶ grudnia (3)
  - ▶ listopada (2)
  - ▶ października (2)
  - ▶ września (4)
  - ▶ sierpnia (2)
  - ▶ lipca (3)
  - ▼ czerwca (2)
    - Chwila przerwy na wyprawę pod namiot....kierunek ...
    - SI4432 moduł RF firmy Silicon Labs - zajęcia prakt...
  - ▶ maja (7)
  - ▶ kwietnia (7)
  - ▶ marca (4)
  - ▶ lutego (4)
  - ▶ stycznia (4)

- ▶ 2017 (54)

### ROZDZIAŁNE POSTY



**CAN - interfejs komunikacyjny prawie doskonały.**

Ponieważ cały czas odkrywam mikrokontrolery PIC, w szczególności rodzinę 16-bitowców. Nie może zabraknąć rozpoznania interfejsu komunika...



**PIC32MM - prace nad płytą "developerską" zakończone.**


Ponieważ stałem się szczęśliwym posiadaczem mikrokontrolerów PIC32MM w obudowach 48 pin (TQFP) i 28 pin (SOIC 208mils). Stąd myśl ab...



**DOGM204W-A - wyświetlacz LCD matrycowo punktowy 4.82 mm 4 x 20. Electronic**



The diagram illustrates the nesting of error detection and correction techniques. It consists of three nested horizontal brackets. The outermost bracket is purple and labeled 'Manchester'. Inside it is a black bracket labeled 'Whitening'. The innermost bracket is red and labeled 'CRC'.



**PIC32MM - SPI + DMA  
zajęcia praktyczne.**

*Poprzednia randka z SPI i  
32-bitowym mikrokontrolerem  
firmy Microchip PIC32MM  
została zrealizowana przy wykorzystaniu  
funkcji wwa...*

SUBSKRYBUJ

 Posty

 Komentarze



Figure 39. Operation of CRCWhiteningManchester Encoding Across Various Fields

Skupmy się jednak na przykładzie i dalszych krokach konfiguracyjnych z nim związanych.

Pierwszym elementem funkcjonalnym ramki jest **Preambuła**, jest to **4 bitowa** sekwencja **0101** złączona do kupy w kolejnych bajtach, w przykładzie 5 kolejnych bajtów. **Preambuła** ma na celu wstępną synchronizację nadajnika z odbiornikiem i razem ze słowem synchronizacyjnym **Synchron Pattern** stanowi konieczny element każdej ramki. Długość preambuły ustawiamy na **9 bitach** w rejestrze **0x33** i **0x34** po resecie otrzymujemy wartość **0x8** czyli długość preambuły wynosi na starcie **8 x 4 bity = 32 bity** i zajmuje **4 bajty**. W przykładzie z dokumentacji mamy natomiast preambułę o długości **40 bitów = 5 bajtów = 10 sekwencji 4 bitowych 0101** i tu od razu widzimy błąd w dokumencie **AN415** na **stronie 12**. Widzimy tam błędny zapis do rejestru nie zgodny z zamierzeniem :

```
SpiWriteRegister(0x34, 0x09); //write 0x09 to the Preamble Length register
```

a powinno być

```
SpiWriteRegister(0x34, 0xA); //10 x 4 bity = 5 bajtów w preamble
```

Teraz pytanie skąd mamy wiedzieć jakiej długości preambułę ustawić ? wiemy tylko że na **9 bitach** możemy zakodować **511 sekwencji 4 bitowych** czyli **255,5 bajta**. Z pomocą przychodzi nam tabelka z optymalnymi wartościami dla wybranego wariantu modulacji :

Table 14. Minimum Receiver Settling Time

Mode	Approximate Receiver Settling Time	Recommended Preamble Length with 8-Bit Detection Threshold	Recommended Preamble Length with 20-Bit Detection Threshold
(G)FSK AFC Disabled	1 byte	20 bits	32 bits
(G)FSK AFC Enabled	2 byte	25 bits	40 bits
(G)FSK AFC Disabled + Antenna Diversity Enabled	1 byte	---	64 bits
(G)FSK AFC Enabled + Antenna Diversity Enabled	2 byte	---	8 byte
OOK	2 byte	3 byte	4 byte
OOK + Antenna Diversity Enabled	8 byte	---	8 byte

Note: The recommended preamble length and preamble detection threshold listed above are to achieve 0% PER. They may be shortened when occasional packet errors are tolerable.

W tabelce widzimy różne tryby modulacji z opcją, tryb wybrany przeze mnie i skonfigurowany w kalkulatorze to **(G)FSK AFC Disabled**. Opcja **Automatic Frequency Control (AFC)** to taki mechanizm, który kompensuje odchyłki częstotliwości nośnej pomiędzy nadajnikiem a odbiornikiem wynikające np. z temperatury, tolerancji elementów pasywnych, odchyłki kwarca etc. Ja to wyłączam aby nie zaciemniać zbyt dużo i tak trochę rozbudowanej konfiguracji. Trzeba jednak pamiętać aby po opanowaniu **Si4432**, ten tryb włączać do konfiguracji modułu.

W tabelce dla wybranego wiersza odczytujemy zalecaną długość preambuły, w naszym przypadku mamy dwie możliwości **20 bitów** lub **32 bity** w zależności jak wybierzemy długość progu detekcji **Detection Threshold**. Ło matko a co to jest ten próg detekcji i gdzie się to ustawia ???? Uprzejmie informuję, że próg detekcji ustawiamy w rejestrze **0x35** na **5 bitach**. Jeśli ustawimy próg detekcji np **0x5** to odbiornik uzna, że preambuła jest prawidłowo odebrana jeśli odbierze poprawnie **5 x 4 bitowe** sekwencje preambuły **0101** i wtedy ustawi flagę poprawnie odebranej preambuły. Czyli próg detekcji jest minimalną ilością sekwencji **4 bitowych** jakie trzeba rozpoznać na całej długości preambuły aby uznać ją za poprawny sygnał preambuły.

Z tabelki wybieram opcję długości preambuły **32 bity** dla **20 bitowego** progu czułości. Czyli jak **20 bitów** preambuły zostanie poprawnie przez odbiornik rozpoznanych na długości **32 bitowej** to moduł ustawi nam w rejestrze statusu odpowiednią flagę, możemy z tej informacji skorzystać lub nie.

Zatem w przykładzie z dokumentacji gdzie długość preambuły ustawiono na **5 bajtów (40 bitów)** zmienimy to na nasz wybór czyli **4 bajty (32 bity)**, bo te ustawienie jest optymalne dla wybranej modulacji **GFSK** i wyłączonej opcji **AFC**.

Tu póki pamiętam mała dygresja, warto wiedzieć, że moduł **Si4432** ma potężne narzędzia (opcje) do optymalizacji przesyłu danych tak aby transmisja była "pewna". Do takich opcji należą **m.in AFC**, kodowanie **Manchester**, **Whitening**.

Z powyższych dywagacji na temat preambuły do funkcji inicjalizacyjnej wrzucamy zapisy :

```
/*Preamble Length 8 x 4 bits = 32 bits*/
SpiWriteRegister(SI4432_REG_34_PREAMBLE_LENGTH, 0x08);

/*Preamble Detection Threshold 5 x 4 bits = 20 bits
 *res_offset
 */
SpiWriteRegister(SI4432_REG_35_PREAMBLE_DETECTION_CONTROL, 0x2A);
```

Kolejnym elementem wysyłanej ramki jest słowo synchronizacyjne **Synchron Pattern** o długości do **4 bajtów**, w przykładzie z dokumentacji **AN415** widzimy dwa bajty i tak też zostawiamy. Jeśli po stronie odbiornika zdefiniowana przez użytkownika wartość słowa synchronizacyjnego nie będzie zgodna z wartością w odebranej ramce to ramka będzie ignorowana i nie odczytamy jej zawartości. Tu nasuwa się od razu myśl na wykorzystanie tej funkcjonalności, mianowicie możemy za pomocą słowa synchro adresować wiadomość i odbiera ją tylko konkretny węzeł lub grupa węzłów reszta ignoruje i nie odbiera tej wiadomości.

Tu wspomnę, że mamy jeszcze do dyspozycji 4 bajtowy nagłówek, który stricte służy wyłącznie do adresowania, w przykładzie nagłówek jest wyłączony dla uproszczenia i tak też zostawiamy i my.

Słowo synchro w przykładzie ma **dwa bajty** długości i są to konkretne wartości **0x2D** i **0xD4**.

Zapis do rejestru długości słowa synchro :

```
/*Synchronization Word 3 and 2*/
SpiWriteRegister(SI4432_REG_33_HEADER_CTRL2, 0x02);
```

Wpisanie konkretnych wartości słów synchro :

```
/* Configure sync Word 3 Value = 2D
```

## PIC32

[PIC32MZ - blog hobbyisty](#)

[Getting started with the PIC32](#)

[PubNub Data Stream Network](#)

[Peripheral Library Code Examples](#)

[PIC32 - zestawienie](#)

[PIC32MM - sympatyczna rodzina MCU dobra na start i prosta do opanowania.](#)

[Bootloader biblioteka Microchipa dla 16 i 32 bit](#)

## PIC18

[PIC18 - rozbudowane peryferia analogowe](#)

[PIC 8-bit Tutorial](#)

[Cyfrowy regulator PID na dsPIC33](#)

[MPLAB Code Configurator cz.1](#)

[MPLAB Code Configurator cz 2](#)

[PIC18 - Tutorial](#)

[PIC18 - zapis/odczyt EEPROM](#)

[PIC18 - XC8 using C coding](#)

[PIC 8-bit Projekty](#)

[PIC 8-bit Tutoriały](#)

[PIC24/ds33 - zestawienie MCU 16-bit](#)

[PIC18F seria K40 - bogato wyposażone 8-bit](#)

[PIC24F seria GB2 - ochrona danych](#)

[PIC18 EP Kurs I2C](#)

[PIC18 EP Kurs A/D](#)

[PIC18 EP Kurs LCD Graf.](#)

[PIC18 EP Kurs Nokia 3510i](#)

[PIC18 EP Kurs LCD Znak](#)

[PIC18 EP Kurs Wstęp](#)

[PIC18 EP Kurs PWM](#)

[PIC18 EP Kurs Interfejs](#)

[PIC18 EP Kurs I/O](#)

[PIC 8-bitów garść informacji](#)

[Kompilatory C i C++](#)

[PIC24 - prezentacja](#)

[Strefa Bibliotek dla PIC24](#)

[PIC24 - narzędzia](#)

[PIC - biblioteki do wyświetlaczy TFT](#)

[dsPIC® Digital Signal Controllers](#)

[PIC Biblioteka Graficzna - opis](#)

[PIC - biblioteki z neta 0](#)

[PIC Projekty z neta 6](#)

[PIC Projekty z neta 5](#)

[PIC Projekty z neta 4](#)

[PIC Projekty z neta 3](#)

[PIC Projekty z neta 2](#)

[PIC Projekty z neta 1](#)

[PICkit3 i 2 porównanie](#)

[PIC kit 3 programator - datasheet](#)

[PIC24 - ciekawa broszura](#)

[16-bit PIC® MCU Architecture](#)

[PIC24HJ128GP502 - dane i dokumentacja](#)

[PICkit4](#)

[PICkit3](#)

## PIC16

[Struktury](#)

[Język C - WikiBooks](#)

[Język C - Tutorial](#)

[C/C++ Language Programming Library Reference](#)

[Biblioteki standardowe języka C](#)

```

* Configure sync Word 2 Value = 04
*/
SpiWriteRegister(SI4432_REG_36_SYNC_WORD3 , 0x20);
SpiWriteRegister(SI4432_REG_37_SYNC_WORD2 , 0x04);
SpiWriteRegister(SI4432_REG_38_SYNC_WORD1 , 0x00);
SpiWriteRegister(SI4432_REG_39_SYNC_WORD0 , 0x00);

```

Przed analizą kolejnego fragmentu ramki do wysłania, bazującej na przykładzie z dokumentu **AN415**, mała dygresja. Ogarnięcie dostępnych opcji modułu **SI4432** i ich skonfigurowanie nie jest trywialnym zadaniem i przynajmniej jest czasochłonne, nie mniej mam nadzieję, że moduł odwdzięczy mi się poprawną i bezproblemową pracą. W dalszych planach apetyt mi rośnie na poznanie kolejnych nowocześniejszych modułów **SI446x**.

Kolejnym elementem ramki jest **Payload length**, tu podajemy długość elementów z pola **DATA (Payload)** do dyspozycji mamy jeden bajt czyli w jednej ramce możemy przesłać maksymalnie **255 bajtów danych**. Ale bufor **FIFO** odbiorczy i nadawczy ma rozmiar **64 bajty**, są mechanizmy w module, które umożliwiają wysyłanie poprzez bufor **FIFO** większej ilości danych niż **64 bajty**, a nawet strumieniowanie danych ale nie będę tego wątku analizował bo by z tego artykułu wyszła zbyt obszerna powieść.

Ponieważ w przykładzie wysyłanych jest **8 bajtów danych** do rejestru **0x3E** wpisujemy taką właśnie cyfrę.

```

/*DATA Length Transmit*/
SpiWriteRegister(SI4432_REG_3E_PACKET_LENGTH , 0x08);

```

Po stronie odbiornika długość odebranej ostatnio ramki odczytamy w rejestrze **0x4B**.

Przedostatnim elementem przykładowej ramki, którą spróbujemy wysłać jest pole **DATA (Payload)**. W tym polu przesyłamy zakodowane w **ASCII** słowo **BUTTON1 (42,75,74,74,6F,6E,31)** i na końcu znak specjalny **CR (0D)**. Generalnie wysyłanie za pomocą modułów **RF** danych w formacie **ASCII** a nie binarnych wydaje się być optymalnym podejściem. Dane ładujemy do bufora **FIFO**, którego adres w module to **0x7F**. Bufor jest automatycznie inkrementowany za każdym nowym zapisem do rejestru. Zatem zapis naszych danych do przesłania w buforze **FIFO** będzie wyglądał tak :

```

/*fill the payload into the transmit FIFO, 8 bytes*/
//write 0x42 ('B') to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x42);
//write 0x55 ('U') to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x55);
//write 0x54 ('T') to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x54);
//write 0x54 ('T') to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x54);
//write 0x4F ('O') to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x4F);
//write 0x4E ('N') to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x4E);
//write 0x31 ('1') to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x31);
//write 0x0D (CR) to the FIFO Access register
SpiWriteRegister(SI4432_REG_7F_FIFO_ACCESS , 0x0D);

```

Aby szybko wyczyścić bufor **FIFO TX** lub **RX** zastosujemy poniższe funkcje :

```

void SI4432_ResetTxRxFifo(void)
{
    SpiWriteRegister(SI4432_REG_06_OPERATING_MODE2 , SI4432_FRCRFX);
}

void SI4432_ResetRxFifo(void)
{
    SpiWriteRegister(SI4432_REG_06_OPERATING_MODE2 , SI4432_FRCRRX);
}

```

Aby wysłać dane z bufora mamy do dyspozycji dwie opcje , manualna i automatyczna. Ja wybieram opcję manual i robimy to poniższym zapisem :

```

/*enable transmitter
 * Ready Mode - 0b
 * Tx on in Manual Transmit Mode. (Automatically cleared to FIFO mode once the packet is sent.)
 */
/*the radio forms the packet and send it automatically*/
SpiWriteRegister(SI4432_REG_06_OPERATING_MODE1 , 0x00);
/*the packet has flown to the receiver*/

```

Opcja automatyczna polega na tym , że w chwili zapełnienia bufora **FIFO**, czyli zapisania do niego **64 bajtów danych**, nastąpi automatyczna wysyłka tych danych. Tę opcję ustawimy w rejestrze **0x08** bit **autotx**.

Ostatnim elementem ramki jest suma kontrolna **CRC**, możemy ustawić tutaj tak aby dotyczyła ona tylko danych lub danych plus nagłówek **Header/Adress** i bajt **Payload length**. Ja wybieram opcję minimalistyczną czyli **CRC** niech moduł liczy tylko dla danych. Mamy różne metody wyliczania **CRC**, ja wybrałem **CRC16-IBM**. Ustawienia dotyczące obsługi **CRC** znajdziemy w rejestrze **0x30** a status tj. **CRC Error** w rejestrze **0x03**

```

/* Select Turn On Packet Handler - ON ,
 * Select LSB/MSB First - MSB
 * Enable CRC
 * CRC Over Data Only
 * CRC TYPE - CRC16-IBM
 */
SpiWriteRegister(SI4432_REG_30_DATA_ACCESS_CONTROL , 0xA0);

```

Słów kilka na temat przerwań. Moduł ma jedno wyjście **nIRQ**, które służy do generowania przerwań jak sama nazwa wskazuje. Ogólnie korzystamy z tego z grubsza tak , że po wystąpieniu przerwania sprawdzamy statusy w rejestrach **0x03** i **0x04** i stąd wiemy co odpaliło przerwanie. W rejestrach **0x05** i **0x06** możemy dokładnie ustalić na co ma przerwanie reagować. Jeśli chcemy np. aby przerwanie było generowane tylko jeśli przyjdzie prawidłowa ramka danych to ustawimy to bitem **enpkvalid** w rejestrze **0x05**

```

/*Set IRQ
 * Enable Valid Packet Received*/
SpiWriteRegister(SI4432_REG_05_INTERRUPT_ENABLE1 , 0x02);

```

Po odczytaniu statusu w obsłudze przerwania , możemy odczytać bufor **FIFO** modułu lub przepisać go do bufora pomocniczego w programie i tam poddać dalszej analizie. Pin **nIRQ** modułu mamy podpięty do nóżki **RB9** po

Język C w pigułce  
 Programowanie w jęz.C  
 Operacje bitowe 1  
 Operacje bitowe 2  
 Wskaźniki 1  
 Wskaźniki 2  
 Wskaźniki 3  
 Wskaźniki na funkcję 1  
 Wskaźniki na funkcję 2

RTOS  
 Microchip przykłady projektów  
 Wzorce Projektowe  
 CAN - izolacja zasilania  
 EasyEda - miniTutorial  
 Microchip Developer Help  
 AI  
 Jak działa Bufor Cykliczny  
 Zephyr RTOS for IOT  
 CAN - jak to działa  
 HackDay.IO  
 MICROCHIP ADVANCED PART  
 SELECTOR  
 CoRTOS - lekki RTOS dla małych MCU  
 ADC - przydatne info na start  
 Nadpróbkowanie - oversampling  
 e-books analiza sygnałowa  
 LINUX - komendy  
 GIT - przewodnik  
 freeRTOS dla dociekliwych  
 freeRTOS STM32  
 freeRTOS dla SAM  
 freeRTOS dokumentacja  
 RTOS dla Cypiska  
 Biblioteki elementów Altium PCB  
 Ciekawy projekt  
 Visual Code  
 Round Robin

Linux  
 RPI - Essential\_Bash\_v2  
 BASH - programowanie  
 LINUX - podstawy

FORMULARZ KONTAKTOWY

Nazwa

E-mail \*

Wiadomość \*

Wyślij

WYKŁADY

Jeśli chcesz współtworzyć tego Bloga lub napisać jakiś artykuł to zapraszam. Jestem otwarty na propozycje, sugestie, pomysły.  
 If you want to co-create this blog or write an article, please feel free to do so. I am open to proposals, suggestions and ideas.



stronie **PIC32MM** a tu mamy z kolei przerwanie **INT2**, które musimy skonfigurować w programie aby reagowało na zboczne opadające. W obsłudze tego przerwania będziemy odczytywać statusy modułu **Si4432** i odpowiednio na nie reagować, ja docelowo będę reagował na status odebranej poprawnie ramki i status błędnego **CRC**. Uwaga flagi statusów kasują się dopiero po ich odczytaniu przez **MCU**.

Ta podróż przez rejestry modułu **Si4432** przynajmniej, że jest ekscytująca a odkrywanie coraz to nowych i zaskakujących opcji fascynujące. Jeśli uda mi się poprawnie skomunikować te moduły to je normalnie polubię na maksa.

Jeszcze póki pamiętam to ustawienie mocy nadajnika czyli to co tygrysy lubią najbardziej, maksymalna moc **+20dBm** i jest to naprawdę jak na taki mały modułik bardzo dużo. Ja do pierwszych prób ustawię małą moc a robię to tak :

```
/*Set TX Power -10dB ...+20dBm step 3dBm / 0x000 minimum ...0b11 maximum*/
SpiWriteRegister(SI4432_REG_6D_TX_POWER, SI4432_TX_POWER_10dB);
```

Warto jeszcze cokolwiek mieć pojęcie o trybach pracy modułu. Informację na ten temat znajdziemy ładnie opisane w datasheet a wszystkie zebrane tryby znajdziemy w poniższej tabelce :

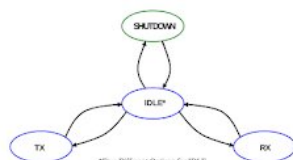


Figure 7. State Machine Diagram

Table 11. Operating Modes Response Time

State/Mode	Response Time to		Current in State (mode)
	TX	RX	
Shut Down State	16.8 ms	16.8 ms	15 nA
Idle States:			
Standby Mode	800 µs	800 µs	450 nA
Sleep Mode	800 µs	800 µs	1 µA
Sensor Mode	800 µs	800 µs	1 µA
Ready Mode	200 µs	200 µs	800 µA
Tune Mode	200 µs	200 µs	8.5 mA
TX State	NA	200 µs	30 mA @ +13 dBm
RX State	200 µs	NA	18.5 mA

Po resecie modułu trybem domyślnym jest tryb **Ready**. Tryby aktywujemy w rejestrze **0x07**. Po stronie modułu nadającego ramkę posłużymy się trybem **TX** a po stronie modułu odbierającego ustawimy tryb **RX**. Jeśli ramka zostanie nadana lub odebrana moduł wychodzi automatycznie z trybu **TX** lub **RX** i wraca do trybu domyślnego.

Pierwsze koty za płoty udało się wysłać ramkę z danymi w eter i potwierdzić to ustawioną flagą przez moduł. Czyli 50 % roboty za mną. Wydaje się, że wszystkie ustawienia zatrybiły prawidłowo co mnie cieszy. W sumie jak na tak dużą ilość ustawień, przez które trzeba było się przegryźć, bezproblemowe zadziałanie modułu można uznać za sukces. Teraz zabiorę się za przygotowanie kodu dla części odbiorczej aby odebrać ramkę i się z tego faktu cieszyć ogromnie :) Im dłużej obcuje z modulem **Si4432** i jego dokumentacją tym bardziej go lubię.

Poniżej główny motyw kodu testującego wysłanie ramki :

```
/*Read interrupt status registers. It clear all pending interrupts and the nIRQ pin goes back to high*/
STATUS1 = SpiReadRegister(SI4432_REG_0E_INTERRUPT_STATUS1); /*Read the Interrupt Status1 register*/
STATUS2 = SpiReadRegister(SI4432_REG_0F_INTERRUPT_STATUS2); /*Read the Interrupt Status2 register*/

/*Enable transmitter
 * Sleep Mode: ON
 * TX on in Manual Transm. Mode. (Automatically cleared in RXD mode once the packet is sent.)
 */
/*The radio forms the packet and send it automatically*/
SpiWriteRegister(SI4432_REG_07_PACKET_LENGTH, 0x00);

while (1)
{
    if (nIRQ_Flag)
    {
        /*Read interrupt status registers*/
        STATUS1 = SpiReadRegister(SI4432_REG_0E_INTERRUPT_STATUS1); /*Read the Interrupt Status1 register*/
        STATUS2 = SpiReadRegister(SI4432_REG_0F_INTERRUPT_STATUS2); /*Read the Interrupt Status2 register*/
        /*If STATUS1 & SI4432_INTERRUPT1 is equal, packet is sent successfully*/
        if (STATUS1 & SI4432_INTERRUPT1)
        {
            /*Clear interrupt flag*/
            nIRQ_Flag = 0;
        }
    }
}
```

Wysłaliśmy jedną ramkę za pomocą wpisu w rejestr **0x07**. W pętli głównej czekamy, aż moduł po wysłaniu prawidłowym ramki ustawi flagę sprzętową i wygeneruje opadającym zboczem przerwanie na pinie **nIRQ**. Przerwanie to przyjmuje na klatkę pin **RB9** w **PIC32MM**, z pinem tym jest skorelowane przerwanie **INT2**. Aby zminimalizować kod w przerwaniu **INT2** ustawiam tam tylko flagę programową **nIRQ\_flag** i w pętli głównej programu czekam aż się ona pojawi. Jeśli flaga programowa zostanie w przerwaniu **INT2** ustawiona, wchodzimy do warunku w którym odczytujemy wszystkie statusy modułu a w następnym kroku sprawdzam selektywnie czy flaga modułu dotycząca prawidłowo wysłanej ramki została ustawiona, jeśli tak to wyświetlam napis **"Send Packet"**. Ten prosty programik umożliwił mi uzyskanie informacji o tym, że ramka poszła w eter, nie mam analizatorów widma etc aby w inny sposób to stwierdzić. Takie sprzętowe oflagowanie modułu przez producenta kapitalnie ułatwia jego uruchomienie.

Kod przerwania **INT2** :

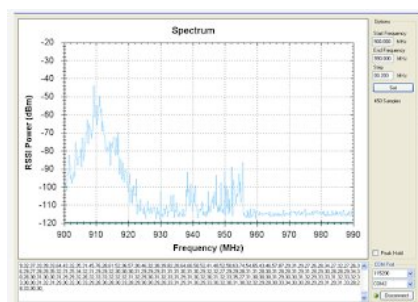
```
volatile unsigned char nIRQ_Flag = 0;

/*
 * Interrupt handler for EX_INT2 - INT2
 */
void __attribute__((weak)) _interrupt_2_isr(void)
{
    nIRQ_Flag = 1;
    EX_INT2_interruptFlagClear();
}
```

Kurcze udało się odebrać pakiet na razie nie wiem jaki ale wiem że na 100 % pochodzi z mojego modułu nadającego :) testowałem tylko czy do odbiornika trafia poprawnie odebrany pakiet. Poniżej fragment testowego kodu po stronie modułu, który odbiera dane :

```
/*Read interrupt status registers. It clear all pending interrupts and the nIRQ pin goes back to high*/
STATUS1 = SpiReadRegister(SI4432_REG_0E_INTERRUPT_STATUS1); /*Read the Interrupt Status1 register*/
STATUS2 = SpiReadRegister(SI4432_REG_0F_INTERRUPT_STATUS2); /*Read the Interrupt Status2 register*/

/*Enable receiver
 * Sleep Mode: ON
 * RX on in Manual Transm. Mode. (Automatically cleared if nIRQ pin is disabled and a valid packet received.)
 */
/*The radio forms the packet and send it automatically*/
SpiWriteRegister(SI4432_REG_07_PACKET_LENGTH, 0x00);
```



W międzyczasie odwiedziłem forum **Silicon Labs** i tam natknąłem się na wątek :  
[wątek na forum Silicon Labs](#) z którego wynika, że odczyt poziomu **RSSI** powinien nastąpić **po słowie synchronizującym** a ja odczytuję ten poziom po przesłaniu całej ramki, trzeba ten wątek sprawdzić.

Why there is no simple precise solution is used universally, and try to explain the approximate condition between signal (PSNR) and quality (perceptual).

Generally,  
db = -50 db  $\rightarrow$  100% quality  
db = -100 db  $\rightarrow$  0% quality

For PSNR signal between -10db and -130db,  
quality  $\rightarrow 2^P \times 10^{db/10}$   
PSNR  $\rightarrow$  (perceptual)  $\times 100$

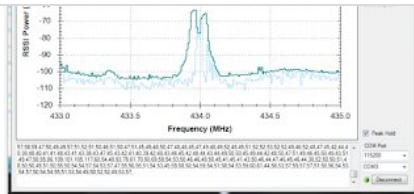
For example:  
High quality: 80%  $\rightarrow$  -50db  
Medium quality: 50%  $\rightarrow$  -70db  
Low quality: 20%  $\rightarrow$  -85db  
Unusable quality: 0%  $\rightarrow$  -100db

[illegible]

**1 kondygnacja w budynku / 113**  
**30 cm / 173**  
**6 cm / 201**  
**1 cm / 230**

Figure 31 is a line graph titled "RSSI vs Input Power". The X-axis is labeled "In Pow [dBm]" and ranges from -120 to 20. The Y-axis is labeled "RSSI" and ranges from 0 to 250. The graph shows a red line representing the RSSI value as a function of input power. The line starts at approximately (-120, 10) and increases linearly, reaching a plateau of about 220 dBm for input powers greater than 0 dBm.





Tutaj natomiast wydaje się wiarygodny post na temat zasięgów w terenie otwartym, wiemy jaka antena, jaka prędkość i moc post o zasięgu RFM22=SI4432

Ja na chwilę obecną uzyskuje pewną transmisję w budynku przez **dwa stropy** w pionie na **3 cm** wypierdki antenie drucianej, wydaje się zatem, że jeśli dać lepszą antenę typu np : **RF-ANT01R-433** to uda się przebić **4 stropy**.

Nie udaje mi się natomiast przebić z sygnałem przez dwa skrajne miejsca w dużym budynku jednorodzinny oddzielone jednym stropem z kilkoma ścianami po drodze. Na wyżej wymienionej antenie podejrzewam, że nie byłoby z tym problemu ale jeśli zależy nam na miniaturyzacji to nie tędy droga.

Jeszcze pokombinuję z długością preambuły, dam więcej czasu na synchronizację zobaczymy jak to pomoże. W sumie gdyby udało się pokryć te dwa skrajne punkty w budynku to byłbym całkowicie zadowolony a na razie jestem zadowolony w **80 %** :). Rozciągnąłem troszkę anteny wypierdki **3 cm**, zyskałem kilka metrów poprawy. Zależy mi jednak na miniaturyzacji a nie stojące pały przy modułach. Coś mi się widzi, że bicie rekordów w zasięgach i przebijalności ścian modułami **RF** nie jest zbyt uniwersalnym rozwiązaniem jeśli chodzi o budynek, tu się bardziej sprawdzają moduły o małych relatywnie mocach pracujące w sieciach typu **Mesh**. **Zigbee** to jest to co byłoby optymalne no ale cena takich modułów nie zachęca. Gdyby w modułach **RF** zaimplementować sieć **Mesh** lub coś podobnego to byłby strzał w **10-tkę**.

Poniżej podaję optymalne długości dla anteny w postaci zwykłego drutu lub kabla dla **433 MHz** :

**1/4 ćwierć falowy - 16,63 cm**

**1/2 pół falowy - 33,26 cm**

**pełnofalowy - 66,51 cm.**

**Długości anteny wyliczone w/g wzoru :  $300/f/(n*k)$  gdzie :**

**f** - częstotliwość w MHz

**n** - n-ty podział fali

**k** - współczynnik skrócenia = **0,96**

tylko uwaga na kolejność działań !!

Spróbuję zrobić test na zwykłym kabelku, w/g powyższych wyliczeń, dla ciekawości. Bo może ten wypierdek **3 cm**, który dostałem z modulem jest jako antena do kitu i to co osiągam na nim to i tak jest giga super a ja tylko marudzę i szukam dziury w całym :)

Ło! no teraz dopiero moduł pokazał pazur. Wywaliłem antenki zakupione z modulem do kosza a w ich miejsce dałem wyliczony w/g wzoru powyżej kawałek zwykłego kabelka z instalacji alarmowej. Kawałek o długości ok **33 cm** czyli długość skrojona z połową fali. No panie golonka teraz mam pokrycie piękne całego dużego domu (**+20dBm**) wszcz, wzdłuż i w poprzek. Nieuszkodzone ramki docierają mi do skrajnych miejsc w najbardziej odległych zakamarkach domu.

Sygnał musiał się przebić przez jeden strop i ok 5 ścian.

Teraz już jestem prawie zadowolony. **33 cm** druciku cienkiego da się zminiaturyzować.

No ale apetyt rośnie w miarę jedzenia, spróbuję ramki wyemitować i odebrać na druciku ok **66 cm** (pełna fala) przy czym zmniejszę moc do **14 dBm**

Kluczem do sukcesu w modułach **RF** jest jak widać nie tylko dobra/poprawna konfiguracja toru radiowego ale i odpowiedni dobór anteny. Antena nie tylko musi spełniać dokładnie co do milimetra warunki długości ale ważne jest dopasowanie impedancyjne anteny do wejścia antenowego modułu. Z punktu widzenia impedancji byle druk nie będzie do końca optymalnym wyborem ale akceptowalnym.

Na pełnofalowym druciku tj długości ok **66 cm** i zmniejszeniu mocy z **20dBm** do **14dBm** nie uzyskałem pełnego pokrycia budynku, szkoda bo miałem na to nadzieję.

W sieci znalazłem bardzo ciekawą i kompletną bibliotekę, która ma zaimplementowaną sieć **Mesh** (rewelacja !!!) dla różnych modułów **RF** w tym dla **RFM22 = SI4432**. Biblioteka napisana w **C++** i tu by trzeba było zaprzyjaźnić się z tym językiem i kompilatorem **XC32++** lub przepertować ją do **C**. **Radio Head**

Mając sieć **Mesh** dla **SI4432** byłoby to genialne, małymi mocami moglibyśmy pokryć dowolny obszar i nie trzeba byłoby martwić się, że za mało pary mamy aby przebić n-tą ścianę czy strop. Wtedy byłaby to doskonała i tania alternatywa dla np. **Zigbee**.

W dokumentacji modułu natrafiłem na taką tabelkę :

RF Sensitivity	P <sub>FA, 2</sub>	(BER < 0.1%) (2 Mbps, GFSK, BT = 0.5, Δf = 25 kHz)	---	-121	---	dBm
	P <sub>FA, 40</sub>	(BER < 0.1%) (40 Mbps, GFSK, BT = 0.5, Δf = 250 kHz)	---	-106	---	dBm
	P <sub>FA, 100</sub>	(BER < 0.1%) (100 Mbps, GFSK, BT = 0.5, Δf = 500 kHz)	---	-104	---	dBm
	P <sub>FA, 125</sub>	(BER < 0.1%) (125 Mbps, GFSK, BT = 0.5, Δf = 625 kHz)	---	-101	---	dBm

w której pokazane jest dla jakich parametrów prędkości i dewiacji uzyskamy optymalną czułość odbiornika. Ustawiłem zatem zgodnie z danymi z tabelki prędkość **2 kbps** i dewiację **5 kHz** i się zdziwiłem. Przy mocy **14dBm** i druciku pół falowym ok.**33 cm**. sygnał pokrył mi cały budynek wszcz, wzdłuż i w poprzek no i zaczynam być bardziej zadowolony :)

Za bardzo ten artykuł zaczyna mi się rozwiekać dlatego czas na stop klatkę i jakieś delikatne podsumowanie :) No i oczywiście program.

Generalnie moduł **SI4432** sprawił mi dużą radość w fazie poznawania i testów.

Dokumentacja jest przyjazna i nawet z przyjemnością ją czytałem. Początkowy stres związany z ilością rejestrów jakie mamy na pokładzie stosunkowo szybko minął wraz z przyswajaniem kolejnych działów dokumentacji. Ogromną pomocą w ujeżdżeniu modułu był dostarczany przez producenta firmę **Silicon Labs**, kalkulator. Bez niego byśmy byli trochę bez szans. Nie odkryłem wszystkich opcji transmisji takich jak np. strumieniowanie danych czy wogóle opcji modułu np. wewnętrzny czujnik temperatury, przetwornik **ADC**.

Jak na pierwsze moje starcie z modułami **RF**, uważam, że poszło mi stosunkowo łatwo. Trochę podświadomie mam niedosyt w zakresie zasięgów szczególnie po tym co się czyta w necie, ale już wiem, że to co ludzie piszą o modułach **RF** trzeba dzielić minimum przez dwa. Na początku wyobrażałem sobie, że na antence, którą kupiłem w zestawie z modułami czyli **3 cm** wypierdek w kształcie sprężynki nawiąże łączność z inną galaktyką :) Rzeczywistość okazała się trochę brutalniejsza. Nie robiłem testów na przestrzeni otwartej bo nie jest mi potrzebna obecnie taka wiedza. Bardziej mnie interesują właściwości modułu w budynku. A tutaj można powiedzieć, że moduł **SI4432** spełnił ostatecznie moje oczekiwania i pokrył mi zasięgiem cały duży budynek jednorodzinny mocą **+14dBm** i antenką drucikiem **33 cm**. Aby uzyskać pełne pokrycie budynku trzeba było zejść z prędkością transmisji do **2kbps**. Jeśli nie mamy potrzeby robić transmisji głosowej lub video a przesyłać jedynie dane z czujników etc to i tak świat i ludzie taka prędkość. Dodatkowo mając w perspektywie możliwość implementacji sieci **Mesh** bo taki soft znalazłem w necie, moduł **SI4432** z mocą **+20dBm** może nam pokryć naprawdę spory kawałek przestrzeni zabudowanej np kilka domków jednorodzinnych etc. I to wszystko za niewielkie pieniądze. Dodatkowo ciekawą możliwością jest sterowanie modulem bezpośrednio z **RasberyPI** co umożliwia nam zbudowanie np. **Gatewaya** z dostępem do internetu etc. I nie wiem przypadkiem czy się nie szarpnę na taki projekt. Znając moduł **SI4432** możemy płynnie przejść do jego następcy **SI446x**.

Na koniec w linkach pojawią się dwa projekty tworzące całość, dedykowane dla **PIC32MM** i środowiska **MPLABX IDE**. Projekty ściągamy z **GitHuba** do **MPLAB X IDE** za pomocą opcji **Clone**. Projekt testowy dla modułu nadającego konfiguruje moduł **SI4432** i wysyła co **1 s**, ramkę testową z napisem **BUTTON1** + znak specjalny. Jeśli pakiet zostanie prawidłowo wysłany otrzymamy potwierdzenie na wyświetlaczu w postaci napisu **"Send Packet"**.

Projekt dla modułu odbierającego ma za zadanie utrzymać moduł w trybie odbioru **RX** i przyjąć nadesłane dane do bufora pomocniczego. Na wyświetlaczu odbiornika zobaczymy treść nadesłanej ramki czyli tekst **BUTTON1** + znak specjalny (na wyświetlaczu to będzie kropka) obok poziom **RSSI** dla odebranej ramki. W pliku nagłówkowym **si4432.h** w obu projektach znajdziemy przyjazne nazwy dla wszystkich rejestrów modułu **RF**.

Życzę powodzenia w przygodzie z modułami **RF**.

Pozdrawiam  
picmajster.blog@gmail.com

Linki :

[SI4432 - Datasheet](#)  
[SI4432 - Register Descriptions](#)  
[SI4432 - AN537](#)  
[SI4432 - AN415](#)  
[SI4432 - Kalkulator do ustawień toru radiowego etc](#)  
[Projekt GitHub - dla modułu nadającego](#)  
[Projekt GitHub - dla modułu odbierającego](#)  
[Artykuł uzupełniający RFM22=SI4432](#)

Autor: [PICmajster](#) o 12:24

## 12 komentarzy:



**Witek** 21 czerwca 2018 10:36

już czytam ;-)

[Odpowiedz](#)



**Witek** 8 lipca 2018 09:24

Dla **PIC32mm0256GPM** pin **RB3** jest też podłączony do **TDI/JTAG** w pliku **MCC.c** zmieniłem **JTAG=ON** na **JTAG=OFF** i zaczął pin działać normalnie

```
// FICD
```

```
#pragma config JTAGEN = OFF // JTAG Enable bit->JTAG is enabled
```

```
#pragma config ICS = PGx1 // ICE/ICD Communication Channel Selection bits->Communicate on  
PGEC1/PGED1
```

[Odpowiedz](#)



**PICmajster** 8 lipca 2018 18:56

W załączonych plikach na GitHub dla Si4432, JTAG jest wyłączony. Warto sobie taki odruch nabyć jak korzystamy z MCC i nawet jak nie wykorzystujemy pinu RB3.

Pozdrawiam  
PICmajster

[Odpowiedz](#)

▼ [Odpowiedzi](#)



**Witek** 9 lipca 2018 10:15

robiłem od początku, dzięki temu teraz zdałem sobie sprawę jakie pułapki czekają ;-). W PIC32mm064gpl28 nie było problemów z JTAG

---

[Odpowiedz](#)



**PICmajster** 9 lipca 2018 11:29

W MPLABX-IDE 4.15 i wtyczce MCC 3.45.1 w zwykłym masz JTAG wyłączony standardowo więc jedna "pułapka" mniej :)

Pozdrawiam  
PICmajster

[Odpowiedz](#)



**Witek** 9 lipca 2018 17:48

skąd te rzeczy wiesz?

[Odpowiedz](#)



**PICmajster** 9 lipca 2018 18:31

Organoleptyka stosowana :)

Pozdrawiam  
PICmajster

[Odpowiedz](#)

**Anonimowy** 31 marca 2020 16:12

Bardzo fajny poradnik!

Znalazłem jednak mały błąd:

"11dBm (czyli połowa dostępnej mocy)". Połowa dostępnej mocy to 17dBm, 11dBm to raptem troszkę ponad 12mW. Skala decybelowa nie jest liniowa!

[Odpowiedz](#)



**PICmajster** 3 kwietnia 2020 11:51

Dziękuję za słuszną uwagę już poprawiłem. Zapraszam również do przeczytania artykułu o SI4463, to ulepszona wersja m.in w torze radiowym.

[Odpowiedz](#)

▼ [Odpowiedzi](#)

**Anonimowy** 3 kwietnia 2020 20:33

:) SI4463 jest niedostępny w detalicznej sprzedaży, poza tym ogamąłem 4432 w maksymalnej formie - z filtrowaniem nagłówek, tworzeniem podsieci - i przeprotestowałem na Cortexa stm32. Dodatkowo na plus jest fakt, że jest wersja 500mW - RFM23BP :)  
Gratuluję wiedzy i chęcią dzielenia się nią! Zaję fajny blog!

---

[Odpowiedz](#)



**PICmajster** 4 kwietnia 2020 07:56

ooo ! to fajowo jak tak daleko zaszleś w komitywie z modulem. Jest biblioteka sieciowa do tego modułu łącznie chyba z Mesh-em. Więc można tutaj robić nawet zaawansowane konfiguracje sieciowe pomimo, że moduł jest relatywnie prosty. Ja mam zamiar swoją dziergać pod Cortex M23 i PIC32MM ale dla SI4463, też mi się marzy sieć na modułach RF.

[Odpowiedz](#)

**Marek** 29 września 2021 02:17



Wszystkie informacje bardzo pomocne

[Odpowiedz](#)



Wpisz komentarz

[Nowszy post](#)

[Strona główna](#)

[Starszy post](#)

Subskrybuj: [Komentarze do posta \(Atom\)](#)

ŁĄCZNA LICZBA WYŚWIETLEŃ

7 6 2 8 5

Obsługiwane przez usługę [Blogger](#).