Niskopoziomowa analiza sygnałów radiowych

Artur „Lacky” Łącki
lackylab.pl
alacki93@gmail.com

- Zawodowo inżynier systemów wbudowanych i analityk podatności.
- Nie zajmuję się zawodowo radiokomunikacją. Jest to wyłącznie hobby.
- Uwagi do moich obserwacji/wniosków mile widziane.
- Pytania najlepiej zadawać w trakcie prezentacji.
- Część rzeczy jest omówione ogólnie ze względu na ograniczenia czasowe.

Czemu radio?

- Bo można.
- Radiokomunikacja to ciekawy kawał wiedzy technicznej.
- Można dorabiać „interfejsy” do taniej elektroniki radiowej.
- Bo SDR jest względnie tani.
- Bo można.

Pogoda za oknem

Czas ostatniego pomiaru: 2025-02-05 20:40:05+01:00

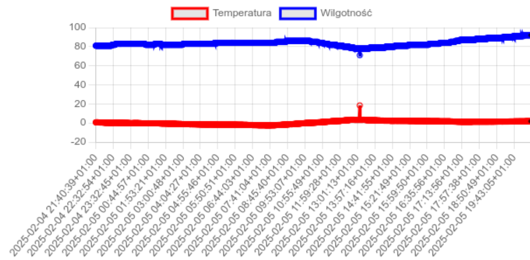
Temperatura: 1.8°C

Wilgotność: 92%

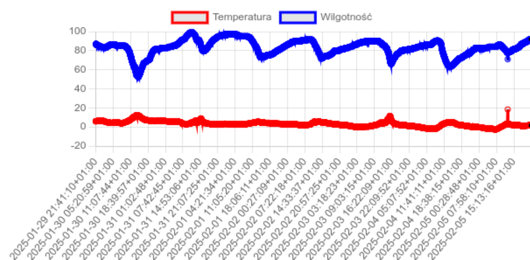
Bateria: OK

ESP8266 + tani (<10zł) moduł radiowy na 433,92MHz

Ostatnia doba



Ostatni tydzień



Przykład w jaki sposób wiedza o komunikacji radiowej może przydać się w majsterkowaniu. Poprzednim razem dorobiłem własny odbiornik do taniej marketowej stacji meteorologicznej. Dzięki temu możliwe stało się archiwizowanie pomiarów i ich podgląd przez stronę WWW. Całość udało się zrealizować na ESP8266 i tanim odbiorniku radiowym. Całość zestawu była tańsza niż najtańszy SDR.

Cel

- Stworzenie programowego odbiornika do pilota od taniego alarmu.
- Nauka GNU Radio.
- Pilot sterował alarmem w starym Seicento (niech mu korozja lekką będzie [*]).
- Transmisja radiowa na bazie układu scalonego PT2262.

Stary pilot od autoalarmu, który stał się celem analizy w GNU Radio.

PT2262

DESCRIPTION

PT2262 is a remote control encoder paired with PT2272 utilizing CMOS Technology. It encodes data and address pins into a serial coded waveform suitable for RF or IR modulation. PT2262 has a maximum of 12 bits of tri-state address pins providing up to 531,441 (or 312) address codes; thereby, drastically reducing any code collision and unauthorized code scanning possibilities.

APPLICATIONS

- Car Security System
- Garage Door Controller
- Remote Control Fan
- Home Security/Automation System
- Remote Control Toys
- Remote Control for Industrial Use

FEATURES

- CMOS Technology
- Low Power Consumption
- Very High Noise Immunity
- Up to 12 Tri-State Code Address Pins
- Up to 6 Data Pins
- Wide Range of Operating Voltage: Vcc = 4 ~ 15V
- Single Resistor Oscillator
- Latch or Momentary Output Type
- Available in DIP and SOP

↑ Brak kodów dynamicznych

Producent twierdzi, że układ można wykorzystać do pilotów systemów alarmowych, ale osobiście odradzam. Układ nie wspiera żadnej formy kodów dynamicznych. Bez żadnego problemu można nagrać transmisję i wykonać „replay attack” albo dorobić swój własny pilot.

PT2262 - transmisja

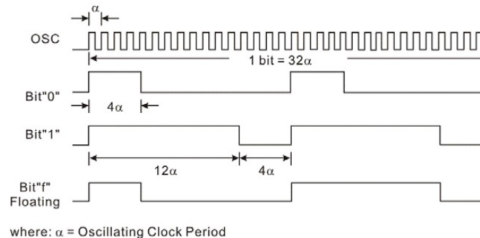
Trójstanowa logika →

CODE BITS

A Code Bit is the basic component of the encoded waveform, and can be classified as either an AD (Address/Data) Bit or a SYNC (Synchronous) Bit.

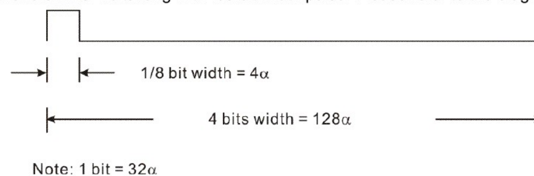
ADDRESS/DATA (AD) BIT WAVEFORM

An AD Bit can be designated as Bit "0", "1" or "f" if it is in low, high or floating state respectively. One bit waveform consists of 2 pulse cycles. Each pulse cycle has 16 oscillating time periods. For further details, please refer to the diagram below:



SYNCHRONOUS (SYNC.) BIT WAVEFORM

The Synchronous Bit Waveform is 4 bits long with 1/8 bit width pulse. Please refer to the diagram below:

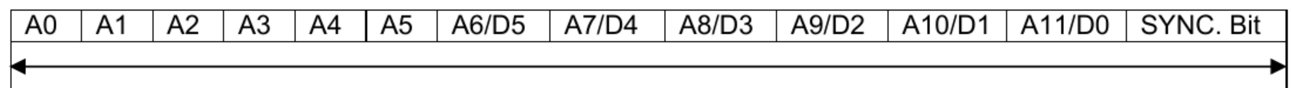


Pilot wysyła dane przy pomocy najprostszej modulacji ASK (OOK). Wyjątkowe jest to, że pilot używa trójstanowej logiki. Trzeci stan jest nazywany w dokumentacji „f", ale później w swoich skryptach oznaczałem go jako bit o wartości „2". Poszczególne stany są kodowane za pomocą długości dwóch impulsów. I tak:

- 0 – dwa krótkie impulsy,
- 1 – dwa długie impulsy,
- 2 – krótki impuls i długi impuls.

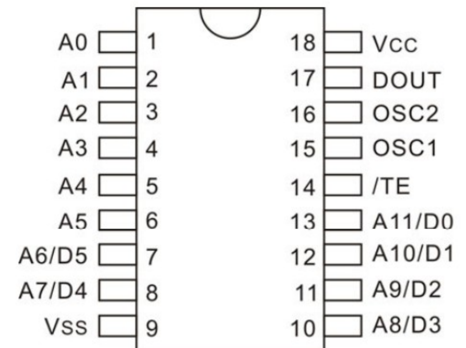
Do tego dochodzi jest bit synchronizacyjny, który składa się z jednego krótkiego impulsu.

PT2262 - transmisja



One Complete Code Word

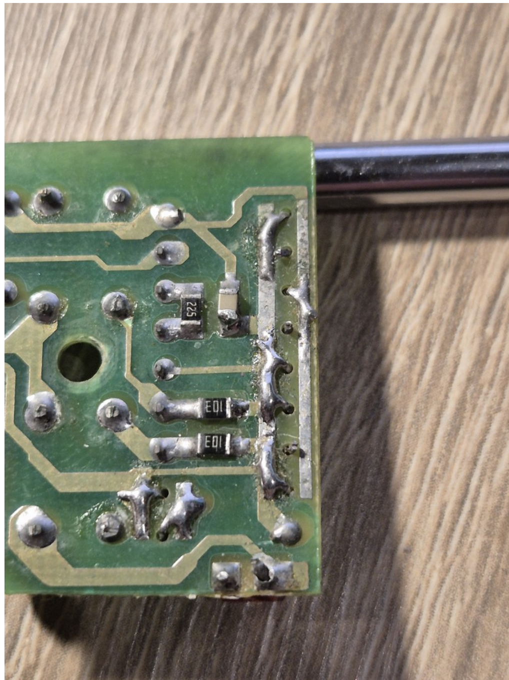
0 Data:	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	Sync Bit
1 Data:	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	D0	Sync Bit
2 Data	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	D1	D0	Sync.Bit
3 Data	A0	A1	A2	A3	A4	A5	A6	A7	A8	D2	D1	D0	Sync.Bit
4 Data	A0	A1	A2	A3	A4	A5	A6	A7	D3	D2	D1	D0	Sync.Bit
5 Data	A0	A1	A2	A3	A4	A5	A6	D4	D3	D2	D1	D0	Sync.Bit
6 Data	A0	A1	A2	A3	A4	A5	D5	D4	D3	D2	D1	D0	Sync.Bit

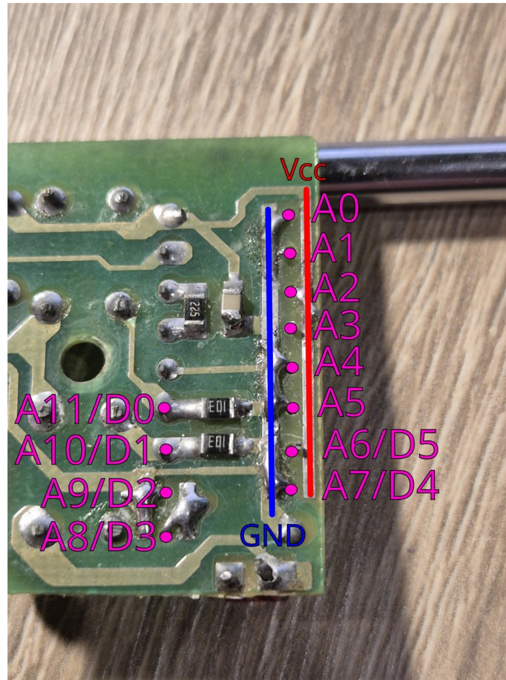


PT2262
PT2262-S18

W praktyce każda wysyłana ramka składa się z 13 „bitów”, które wysyłają stany odczytane na poszczególnych nóżkach układu. W założeniu producenta piny A powinny odpowiadać adresowi odbiornika, z którym sparowany jest pilot, a piny D mają odpowiadać stanowi przycisków na pilocie. W praktyce widać, iż producent pozwala, żeby piny D służyły również za piny adresowe.

Programowanie pilota



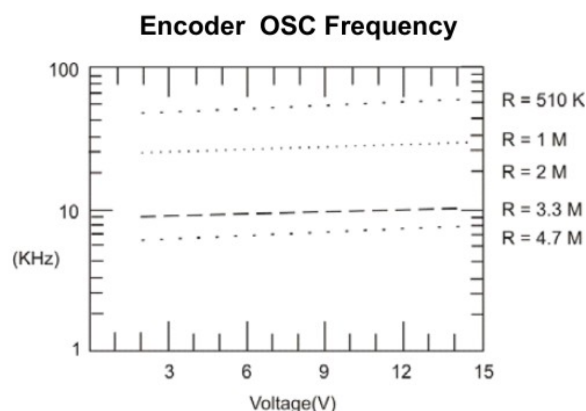


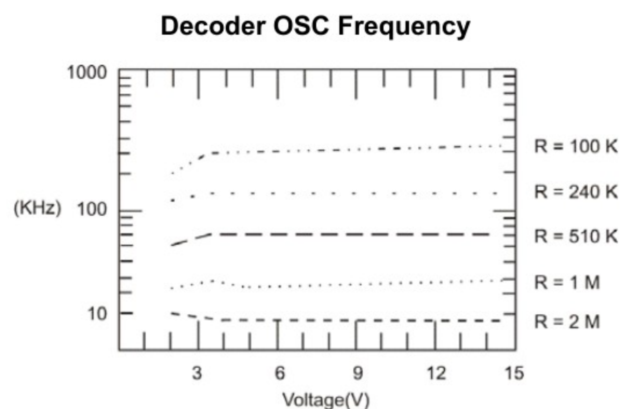
Kod pilota:
0212002010xx

„Programowanie" pilota sprowadza się do połączenia pinów z GND (bit 0), Vcc (bit 1), pozostawienie nieprzylutowanymi (bit „f"). W ramach cięcia kosztów w tanich pilotach odbywa się to przez łączenie odrobiną cyny, ale można również zakupić piloty z trójstanowymi DIP switchami, którymi można łatwo zmienić adres odbiornika.

Na zdjęciu jest przykład jak odczytać adres pilota. Jako „x" zostały oznaczone bity, gdzie jest wysyłany stan przycisków.

Programowanie pilota





Suggested oscillator resistor values are shown below.

PT2262	PT2272
4.7 M	820 K*
3.3 M	680 K*
1.2 M	200 K**

Note:
* -- Operates when PT2272's Vcc=5V to 15V
** -- Operates when PT2272's Vcc=3V to 15V

Na poprzednich slajdach było widać, że czas trwania impulsu to wielokrotność współczynnika alfa, który jest powiązany z częstotliwością wewnętrznego oscylatora. Szybkość jego pracy modyfikuje się przez przylutowanie odpowiedniego rezystora. Producent proponuje trzy pary rezystorów dla nadajnika i odbiornika (układ PT2272). Mój testowy pilot miał wlutowany rezystor o oporze trochę powyżej 2MΩ. Nadajnik i odbiornik muszą być sparowane pod kątem częstotliwości pracy oscylatora.

Nagranie transmisji

W większości wypadków domowe urządzenia radiowe korzystają z pasm ISM (Industrial, Scientific, Medical), które są nielicencjonowane.

Typowe częstotliwości:

- 315MHz
- 433,92MHz
- 868MHz
- 2,4GHz

Nagranie transmisji

Universal Radio Hacker (URH) - https://github.com/jopohl/urh

Oprócz GNU Radio polecam zainteresować się programiem Universal Radio Hacker. Ma on dużo mniejsze możliwości, ale dzięki temu jest też dużo prostszy w opanowaniu. Mój poprzedni projekt odbiornika do stacji meteo był prototypowany w całości w URH.

Słowo o zapisie

- *.complex16s – surowy zapis danych z SDR.
- W praktyce binarny zapis liczb zespolonych: I (część rzeczywista) i Q (część urojona).
- Więcej o próbkowaniu IQ https://pysdr.org/content/sampling.html
- Upewnijcie się jak wg twórców danej aplikacji wygląda dwubajtowa próbka IQ.

Na potrzeby początkowej analizy polecam zapisać próbki transmisji radiowej do plików *.complex16s. Jest to bardzo prosty format, który jest surowym zapisem danych z SDR (nie posiada nagłówka, sumy kontrolnej, metadanych itp.).

Jeżeli nagrywacie transmisję w jednej aplikacji, a później analizujecie ją w innej, to upewnijcie się jak poszczególne aplikacje interpretują pojęcie „16 bitowej liczby zespolonej”.

URH:

3.1 Importing a signal

Apart from recording, a signal can be added to Interpretation tab via File → Open. File ending determines how URH handles the signal. URH understands these file endings:

- `.complex` files with complex64 samples (32 Bit float for I and Q, respectively). This is the default signal file format and will also be used in case the file has no ending at all.
- `.complex16u` using two unsigned 8 Bit integers for I and Q
- `.complex16s` using two signed 8 Bit integers for I and Q
- `.wav` files can be imported, but must not be compressed, i.e., they should be PCM.
- `.csv` e.g. from USB oscilloscope can be imported using a CSV wizard available with File → Import → IQ samples from csv.

GNU Radio:

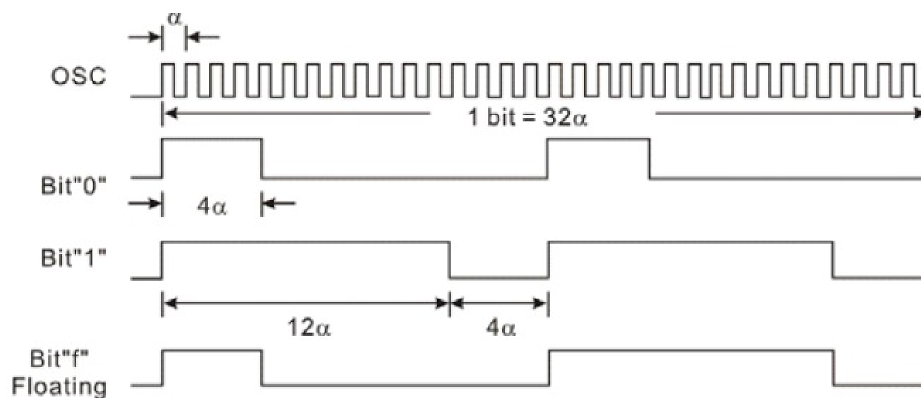
Complex samples stored as interleaved 16-bit integers:

```
[ I sample 0: 16 bit int ][ Q sample 0: 16 bit int ][ I sample 1: 16 bit int ][ Q sample 1: 16 bit int ][ I samp
```

- Na slajdzie są fragmenty dokumentacji URH i GNU Radio.
- Wg URH 16 bitowa bitowa zmienna zespolona to: 8 bitów części rzeczywistej i zaraz po niej 8 bitów części urojonej.
- Wg GNU Radio jest to 16 bitów części rzeczywistej i 16 bitów części urojonej.

Dekodowanie

- Bit 0 – 10001000
- Bit 1 – 11101110
- Bit f (2) – 10001110



Pojawia się pytanie jak te ciągi impulsów przedstawić, symbolizujące bity, przedstawić w pamięci aplikacji. Ja wykorzystałem właściwość, iż wszystkie długości impulsów, przerw, całkowity czas trwania są podzielne przez 4α. W efekcie jeżeli przez czas 4α był stan wysoki, to przedstawiam go jako „1”, jeżeli był stan niski to przedstawiam go jako „0”. Ponieważ całkowita długość jednego bitu to 32α, to mogę go zapisać jako zwykłe 8 komputerowych bitów. Później taki zapis jest konwertowany wprost do liczb 0, 1, 2 (f) i 3 (bit synchronizacyjny).

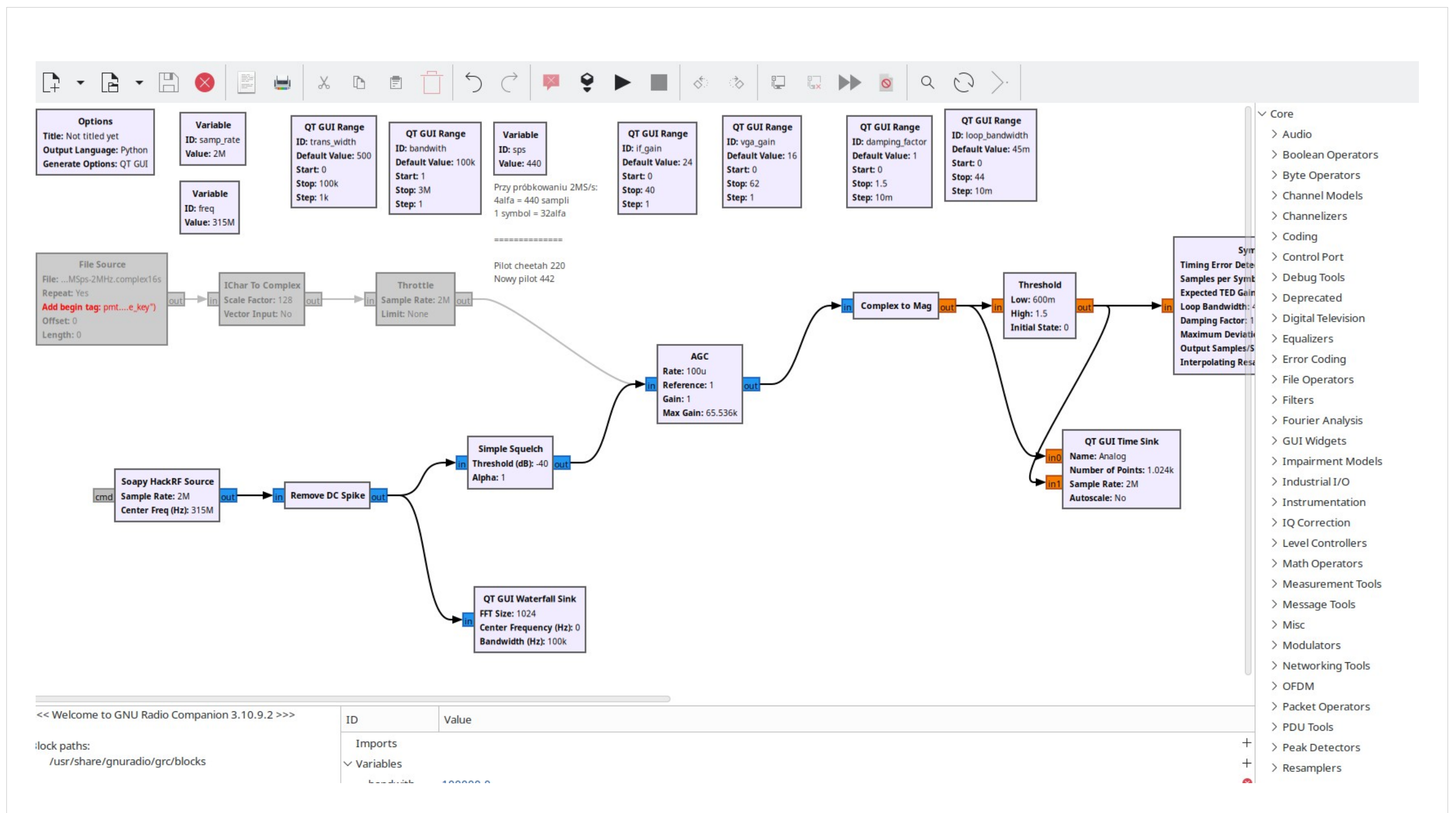
GNU Radio

- Program do DSP (nie tylko SDR).
- Nie bierze jeńców ;_;
- Nierówny poziom dokumentacji.
- Nawet jeżeli jest dokumentacja, to może zakładać, że znacie teorię z DSP i nie będzie jej tłumaczyć.

Po moich doświadczeniach z GNU Radio mogę stwierdzić, że to program, który jest fajny do DSP, a nie tylko stricte SDR.

O GNU Radio należy myśleć jak o Matlabie albo LabView. Program zakłada, iż posiadasz pewną wiedzę teoretyczną na temat przetwarzania sygnałów i nie będzie Ci jej tłumaczyć. Bez tej wiedzy po otworzeniu okna GNU Radio nie będziesz mieć zielonego pojęcia od czego zacząć. Jest on wyłącznie narzędziem inżynierskim i nie zastąpi braków wiedzy w temacie.

Niestety przygotujcie się, że dokumentacja GNU Radio momentami jest biedna, a (przynajmniej mi) ciężko było znaleźć realną pomoc w internecie.

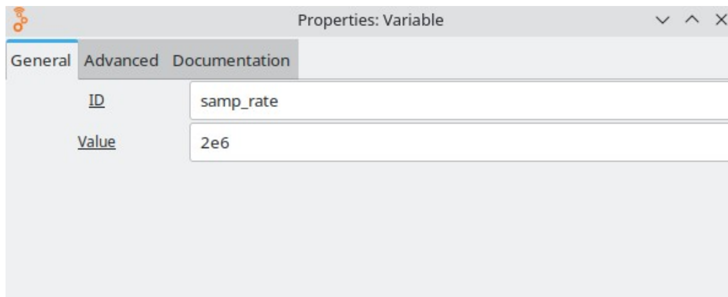


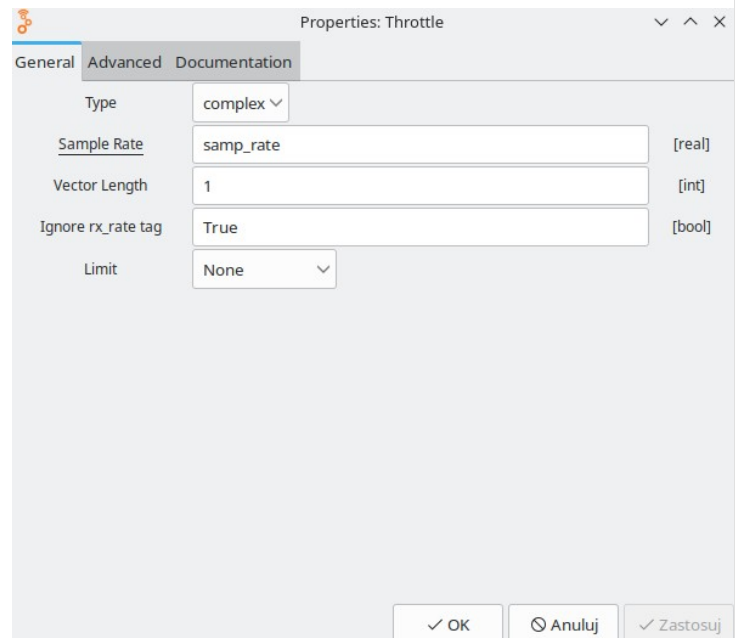
W GNU Radio projekt tworzymy układając i łącząc kolejne bloki, które przetwarzają sygnał. Przy tworzeniu należy pamiętać, iż całość działa na zasadzie „potoku danych”, czyli kiedy drugi blok przetwarza jakieś dane, to blok pierwszy już pracuje nad kolejną ich porcją.

Zmienne

Variable
ID: samp_rate
Value: 2M

Variable
ID: freq
Value: 315M



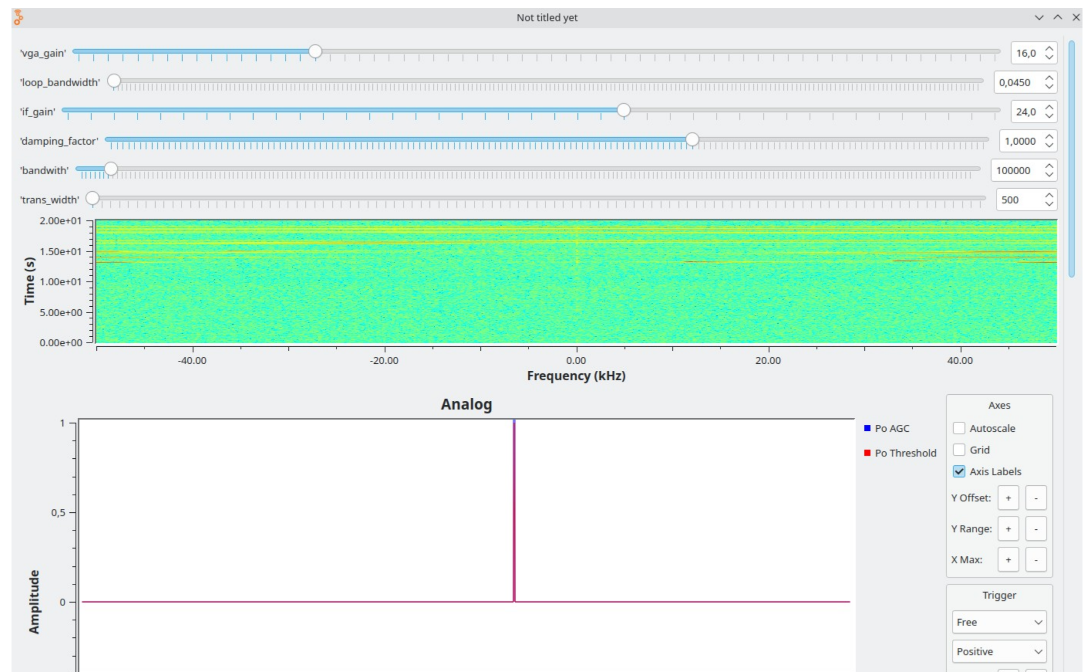


Podstawowym typem bloków są zmienne. Można używać w nich notacji naukowej. Jeżeli w jakimś innym bloku chcemy użyć danej zmiennej to jako wartość podajemy ID danej zmiennej.

Wizualizacja

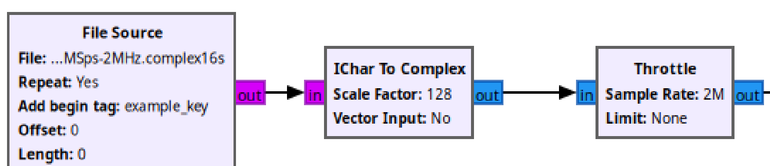
QT GUI Range
ID: if_gain
Default Value: 24
Start: 0
Stop: 40
Step: 1

QT GUI Range
ID: vga_gain
Default Value: 16
Start: 0
Stop: 62
Step: 1



GNU Radio udostępnia nam różne formy wizualizacji danych (np. słynny wodospad) albo kontrolki umożliwiające modyfikację zmiennych na żywo podczas pracy naszego projektu.

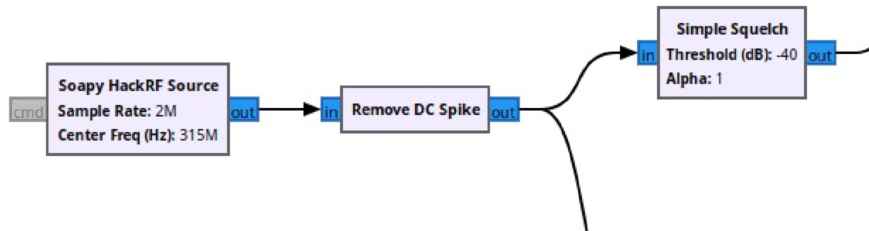
Źródło sygnału - plik



- **File source** – Ścieżka do pliku, informacje o strukturze, opcjonalne tagowanie itd.
- **IChar To Complex** – Konwersja próbek IQ na potok danych zmiennoprzecinkowych.
- **Throttle** – Ograniczenie częstotliwości próbkowania.

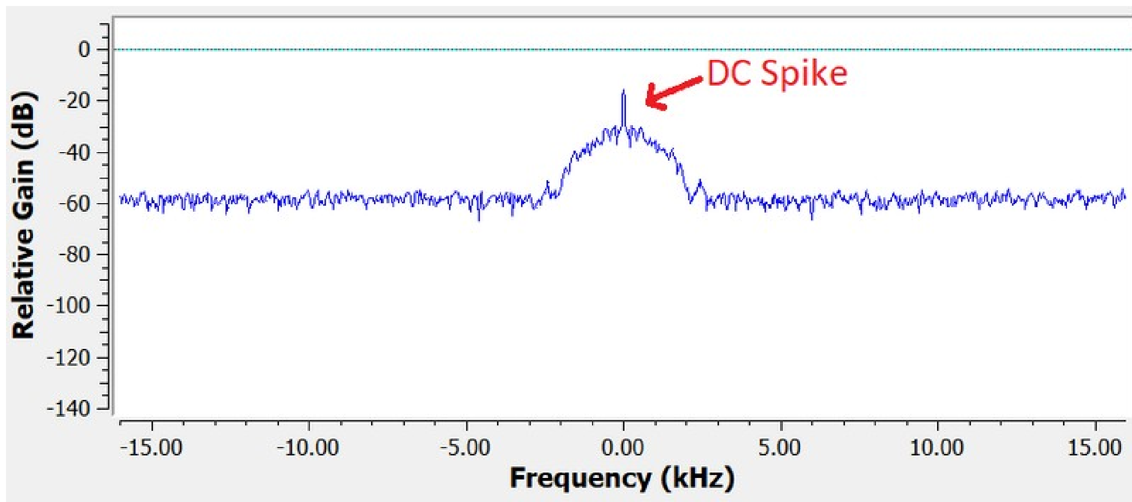
Sygnał w naszym projekcie może pochodzić z pliku (np. *.complex16s) lub rzeczywistego urządzenia. W tym przypadku widać ładowanie danych z pliku. Poza samym załadowaniem pliku należy to przekonwertować z surowych bajtów na typ liczb zespolonych GNU Radio. Ponieważ ładujemy z pliku, to musimy blokiem Throttle sztucznie ograniczyć częstotliwość próbkowania.

Źródło sygnału - SDR

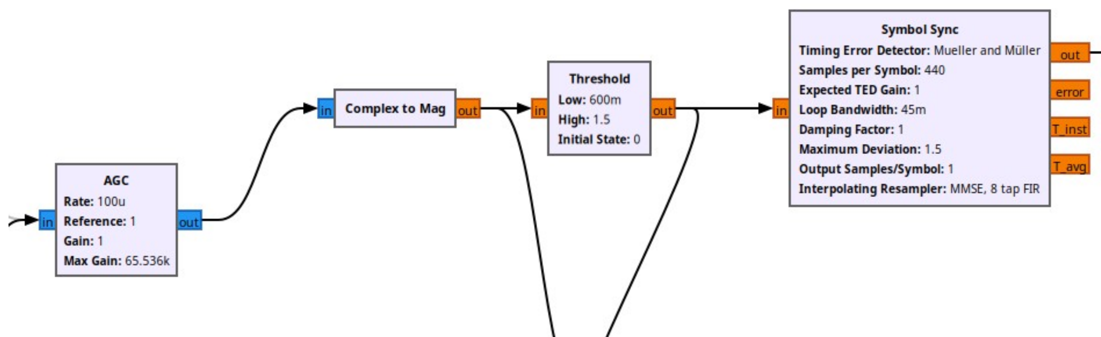


- **Soapy HackRF Source** – Wykorzystaj API Soapy do komunikacji z SDR https://github.com/pothosware/SoapySDR
- **Remove DC Spike** – Usuwa zakłócenia mające charakter napięcia stałego (DC)
- **Simple Squelch** – Wyciszanie szumów i zakłóceń.

Każdy SDR posiada swoją własną bibliotekę umożliwiającą napisanie programu, który z niego korzysta. Tylko jeżeli chcecie w swoim programie dodać wsparcie dla kilku modeli SDR, to stajecie przed podobnym problemem jak twórcy gier na MS-DOS chcący dodać obsługę dźwięku. Macie na rynku kilka standardów i każdy musicie osobno implementować. Soapy jest projektem, który daje jedno wspólne API dla różnych modeli SDRów.


DC Spike
Relative Gain (dB)
0
-20
-40
-60
-80
-100
-120
-140
-15.00
-10.00
-5.00
0.00
5.00
10.00
15.00
Frequency (kHz)

Odzyskiwanie symboli

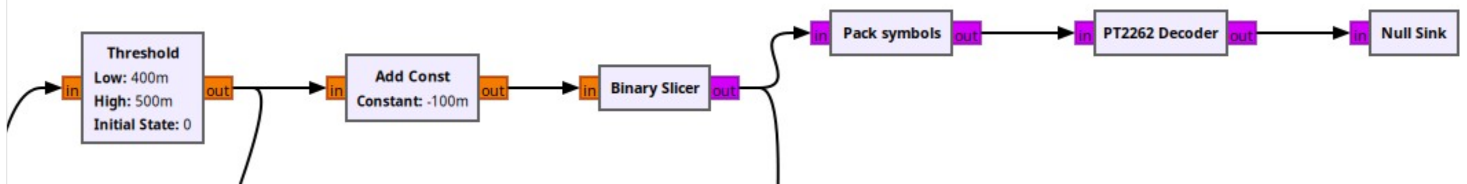


- **AGC –** Automatyczna kontrola wzmocnienia.
- **Complex to Mag** – Obliczanie modułu liczby zespolonej. W praktyce: $\sqrt{I^2+Q^2}$
- **Threshold** – Ustawianie progu sygnału.
- **Symbol Sync** – Synchronizacja symboli. Pozwala określić początek i koniec danego symbolu z uwzględnieniem przesunięć czasowych i zmianie długości. Słowo klucz: TED (Timing Error Detector).

W realnym świecie sygnał radiowy „pływa” i nigdy nie będzie sytuacji, gdy czasy trwania impulsów, amplituda itp. będą idealne. Zadaniem bloku „Symbol sync” jest wyodrębnienie przesyłanych symboli pomimo takich odstępstw.

Sam algorytm „Mueller and Muller” pomimo iż jest znany od lat 70-tych i jest wspominany jako podstawowy algorytm rozwiązujący ten problem, to ciężko znaleźć o nim informacje po polsku (albo ja nie umiem szukać). Na ten moment sposób działa tego bloku jest dla mnie trochę tajemnicą, ale chciałbym w przyszłości zgłębić ten temat.

Końcowe przetwarzanie



- **Add Const** – Dodaj (odejmij) stałą do sygnału.
- **Binary Slicer** - Zamiana sygnału analogowego na pojedyncze bity. Wartości wejściowe **poniżej** zera są konwertowane na bity o wartości 0.
- **Pack symbols** – Skrypt Pythona pakujący pojedyncze bity do symboli.
- **PT2262 Decoder –** Skrypt Pythona. Parsowanie symboli.

Kiedy odzyskaliśmy symbole z transmisji radiowej, to musimy przekonwertować sygnał na ciąg pojedynczych zer i jedynek (do tej pory z perspektywy GNU Radio jest to nadal spróbkowany sygnał analogowy. Samo dekodowanie bitów na konkretne wciśnięcia pilota początkowo próbowałem zrealizować na gotowych blokach GNU Radio, ale w pewnym momencie uznałem, iż to jest bez sensu (albo nie umiem) i postanowiłem stworzyć dwa bloki. „Pack symbols" zbiera pojedyncze klasyczne bity i konwertuje je w „trójstanowe" bity. „PT2262 Decoder" składa bity radiowe w całe ramki i wyświetla w konsoli informację o wciśniętym na pilocie przycisku.

Własne skrypty

Podstawowe zasady:

- Wasz blok (skrypt) działa cały czas podobnie jak inne bloki.
- __*init*__ jest wykonywany tylko raz, przy starcie przetwarzania.
- *work* jest wykonywany zawsze gdy pojawią się nowe dane.
- Używajcie NumPy gdzie tylko możecie (zwłaszcza zamiast pętli).
- GNU Radio daje wam gotowe bufory na dane wejściowe (input_items) i dane wyjściowe (output_items).
- Rozmiary tych buforów mogą być różne przy każdym uruchomieniu *work*.
- Funkcja *print* może skutecznie zawiesić cały graf.

```
1  """
2  Embedded Python Blocks:
3
4  Each time this file is saved, GRC will instantiate the first class it finds
5  to get ports and parameters of your block. The arguments to __init__  will
6  be the parameters. All of them are required to have default values!
7  """
8
9  import numpy as np
10 from gnuradio import gr
11
12
13 class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
14     """Embedded Python Block example - a simple multiply const"""
15
16     def __init__(self, example_param=1.0):  # only default arguments here
17         """arguments to this function show up as parameters in GRC"""
18         gr.sync_block.__init__(
19             self,
20             name='Embedded Python Block',   # will show up in GRC
21             in_sig=[np.complex64],
22             out_sig=[np.complex64]
23         )
24         # if an attribute with the same name as a parameter is found,
25         # a callback is registered (properties work, too).
26         self.example_param = example_param
27
28     def work(self, input_items, output_items):
29         """example: multiply with constant"""
30         output_items[0][:] = input_items[0] * self.example_param
31         return len(output_items[0])
32
```

Kiedy w GNU Radio wybierzecie utworzenie swojego bloku w Pythonie, to program wstawi wam już gotowy szablon. Najważniejsza jest funkcja „work”, która jest uruchamiana dla każdej nowej próbki danych. Jeżeli wasz blok do działania potrzebuje pełnego bloku danych, to musicie na własną rękę zaimplementować jakąś formę buforowania, bo nie macie wpływu ile danych przekaże GNU Radio do funkcji work. Raz to może być tablica 10 wartości, a innym razem 5, kolejnym 15.

Należy pilnować szybkości działania waszego kodu. Zbyt duża ilość wywołań „print” albo zbyt długa pętla może skutecznie zawiesić cały projekt.

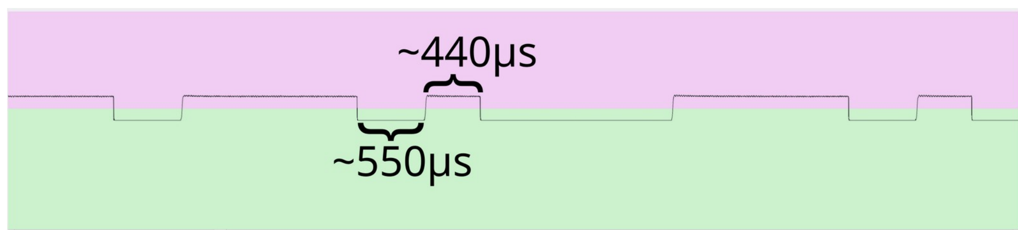
Demo

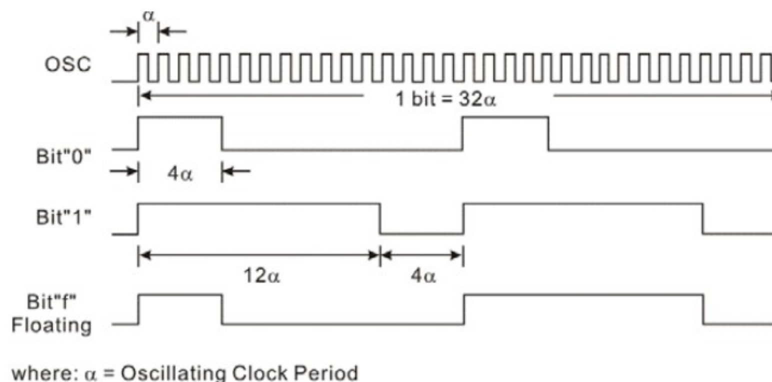
Inny pilot

- Pilot kompatybilny z PT2262…
- ...ale w środku zamiast PT2262 siedzi SC2262.
- Wg dokumentacji układ jest w pełni kompatybilny…
- ...chyba, że jednak nie.

W ramach ciekawości zakupiłem inny pilot, żeby sprawdzić czy mój projekt potrafi dekodować również inne urządzenia zgodne z PT2262. Okazało się, że pilot nie używa oryginalnego układu, a jakiś zamiennik.

Inny pilot





Niestety zamiennik nie trzyma czasu trwania impulsów. Wg dokumentacji czas trwania krótkiej przerwy powinien być równy czasowi trwania krótkiego impulsu (4α). Oryginalny układ realizował to założenie, ale zamiennik już ma na tyle dużą rozbieżność, że blok „Symbol sync” nie potrafi sobie z tym poradzić. Moim celem na przyszłość jest znalezienie sposobu na poradzenie sobie z tą rozbieżnością.

Co dalej

- Dalsze uzupełnianie braków wiedzy teoretycznej.
- Poprawa kompatybilności z dwoma pilotami.
- Stworzenie aplikacji niezależnej od GNU Radio.
- Być może LuaRadio https://luaradio.io/

Pytania?