

Real-Time 2FSK Demodulation and Processing

While it is possible to simply use pre-made hardware for communications, they fail to be useful over long distances. In the event that parameters of transmission/reception need to be adjusted, the CC1011 device along with its driver are not as sensitive as we would like. So, the demodulation process must be replicated in GNU Radio using a much more versatile USRP N200 SDR.

Understanding how the accompanying driver¹ for the CC1011 in the Arduino UNO functions was an interesting challenge, as it involved directly looking at the incoming signal and figuring out what its components were.

Arduino UNO and CC1011 Driver

```
ELECHOUSE_cc1101.Init();  
ELECHOUSE_cc1101.setGDO0(gdo0);  
ELECHOUSE_cc1101.setCCMode(1);  
ELECHOUSE_cc1101.setModulation(0);  
ELECHOUSE_cc1101.setMHZ(433.92);  
ELECHOUSE_cc1101.setSyncMode(2);  
ELECHOUSE_cc1101.setPA(-15);  
ELECHOUSE_cc1101.setCrc(0);  
Serial.println("Tx Mode");
```

Figure 1: Configuration settings in Tx Arduino code.

The transmission portion of the driver had several settings to change, like modulation type, sync word, and CRC (Cyclic Redundancy Check).

Modulation was set to 2-FSK², as it is the simplest digital modulation scheme to deal with, simply transmitting one frequency for a 0, and another nearby frequency for a 1.

The Sync Mode is important for proper sampling, as its purpose is to have a computer's clock period synchronize with the incoming signal's bit period, so that bits are sampled as best as possible. The bit processing portion also uses this sync word to determine whether a bit change was noise, or an actual signal coming through.

Transmission power (setPA) was set to -15 dBm (0.03mW), to avoid saturating the SDR's preamplifier, which can introduce noise and other spurious signals. This information can be found in the UBX-40 daughterboard datasheet³.

CRC was disabled, as the added complexity of bit error detection and correction can be dealt with at a later time.

¹ https://github.com/LSatan/SmartRC-CC1101-Driver-Lib/blob/master/examples/CC1101%20default%20examples/Old_Method_with_GDO/cc1101_Transmitt_Hello_World_minimal/cc1101_Transmitt_Hello_World_minimal.ino

² https://en.wikipedia.org/wiki/Frequency-shift_keying

³ <https://www.ettus.com/all-products/ubx40/>

Demodulating the Signal

The output signal of the CC1011 was demodulated with GNU Radio, using several blocks designed for an FSK modulated signal.

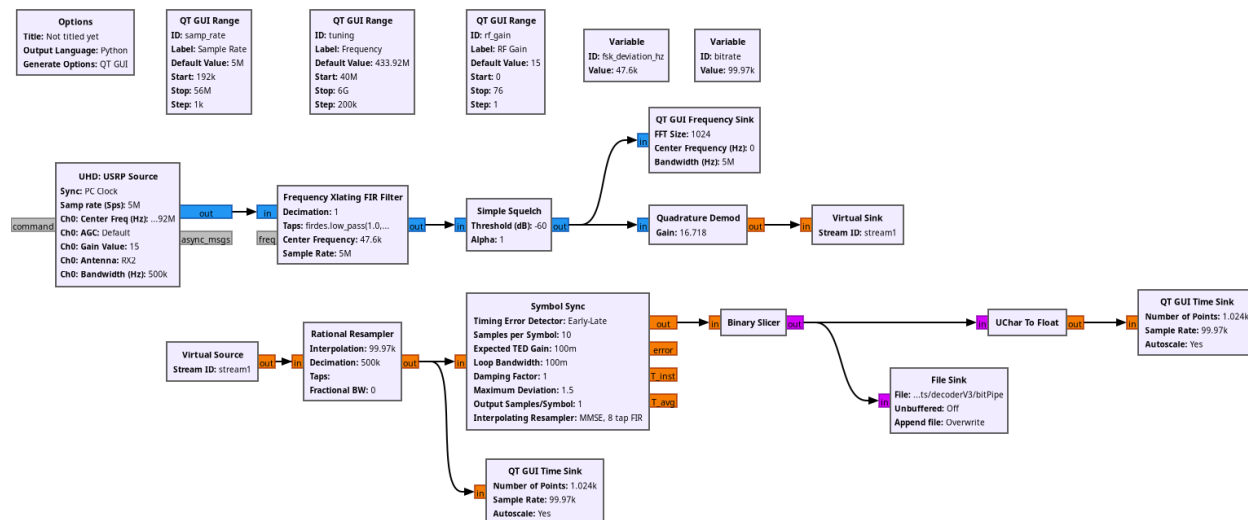


Figure 2: GNU Radio block 2-FSK demod block diagram.

After the USRP Source block, the Frequency Xlating FIR Filter⁴ is the first main component of the filtering process. At this stage, the signal has been down converted to baseband, so we see a spike at 0Hz, and a spike at $2 \cdot \Delta f$ (Hz), the frequency deviation set by the transmitter. These two frequencies determine the symbols 1 and 0. This filter is responsible for centering, or “translating” our center frequency to be directly in the center of both spikes, so 2-FSK demodulation can be performed.

The signal is then fed into a Quadrature Demod⁵ block, which samples the signal, producing a float for frequencies below the center, and the negative for frequencies above the center⁶.

Because the signal was sampled at 5MHz, and the bitrate is only 99.97kBaud⁷ (specified by Tx device), the data must be resampled to avoid needlessly processing large amounts of data. This block interpolates by 99,970 and decimates by 500,000 producing a signal with ten samples per symbol (sps).

⁴ https://wiki.gnuradio.org/index.php/Frequency_Xlating_FIR_Filter

⁵ https://wiki.gnuradio.org/index.php/Quadrature_Demod

⁶ When viewing the output of this block using a QT GUI Time Sink block, we see a very round waveform, opposed to the expected square wave. This is due to raised cosine pulse shaping, designed to minimize the effect of inter-symbol interference in high bit rate signals. The “peaks” mentioned are referring to the peaks and valleys produced by such pulse shaping.

⁷ Baud is similar to symbols per second, which for 2FSK is equal to bitrate.

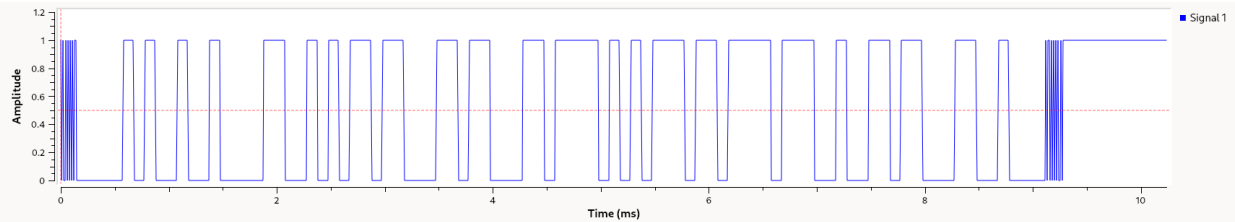


Figure 3: Output of Rational Resampler, passed through a binary slicer with no symbol synchronization.

This signal is then fed to the symbol sync block, which uses the previously mentioned sync word to adjust its sampling rate to extract bits at their peaks. This then produces a rate of 1sps, exactly what we need to extract the bits using the binary slicer block.

Pieces of the Signal

With the settings chosen for transmission, the signal is composed of four segments. The first segment is a simple on/off pulse train, serving as a clock that the symbol sync block locks to. The first few bits of this clock end up being lost, so another synchronization segment is necessary to make sure bytes are read in phase.

The second segment is a special word, constant for all transmissions, produced by the CC1011. This word was extracted by looking directly at the signal to see what it is, it was then written down and used in a program to pack bits into bytes.

The third segment is just one byte, telling the processor how many bytes of data are in the fourth segment of the signal, before it ends.

Packing Bits into Bytes

While GNU Radio features many helpful tools for signal processing, it does not have tools for doing useful things with the extracted bits. To process the bits, a file sink block must be used, which can be told to sink the bits into a FIFO file, a special unix file type that acts as a named pipe, sending data from one process to another⁸.

Decoder V1

Decoder V1 is an extremely basic approach, not relying on the sync word, resulting in a 1/8 chance that the message would show up coherently. All it does is let GNU Radio use the “Pack K Bits” block, regardless of phase, and print the resulting ASCII character if it is a readable one.

```
cusli@fedora:~/Documents/decoder$ ./a.out
Opening...
Opened
ND-!Hello
World?9Vv&C
9Vv&CjB[UVZ
C+ccy(c"ND-
!)Hello Worl
ld_9Vv&EUZr
!1M9Vv&CUVZ
C+ccy(c"jB[
Hello World
9Vv&AUVC+c
cy(c!9Vv&EN
D-!jU1267+6
29Vv&CUZr!1
MUU"@?ND-!]
```

Figure 4: Output of Decoder V1, occasionally in phase.

⁸ <https://www.youtube.com/watch?v=2hba3etpoJg> Some sample code used for FIFO reading/writing.

Decoder V2

Decoder V2 does not rely on the pack K bits block, instead using its own function. It uses a state machine to serially process the data. The issue with this design is that it hard codes the expected data length, so messages are confined to a specific character requirement.

State 0 is the idle state, where the process waits for a bit to change. When nothing is being sent, the symbol is a 1, so the transition to state 1 happens when a bit is flipped to 0.

State 1 is the locking state, where bits are serially fed into a buffer, and each time the buffer is compared to the sync word to see if a transmission is actually occurring or if the bit flip was only the result of noise. If the sync word is detected, the state transitions to state 2. If the buffer size runs out, the flip is deemed to be noise and the state transitions back to state 0.

State 2 is where data collection happens, feeding bits serially into a data buffer. When the counter specified in advance runs out, the resulting bits are passed to a bits to byte function, packing every eight bits into a byte, so that they can be read as ASCII characters and printed to the console, resulting in a message. After this, the state transitions back to state 0 to await another signal.

Decoder V3

Decoder V3 is very similar to Decoder V2, however includes another state to extract the length in bytes of the incoming data. State 2 of V2 becomes state 3 of V3, and the length extraction of V3 becomes state 2.

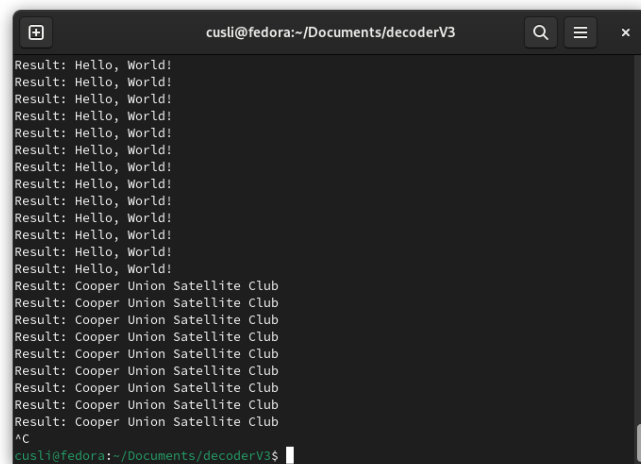


Figure 5: Output of Decoder V3, all messages in phase in real time.

State 2 is very similar to state 3, although its only purpose is to extract the length segment of the signal, to be used later in state 3 as the upper bound of a counter. This state has no failure condition, it always transitions to state 3.

State 3 of V3 functions in nearly the same way as of V2, except that the upper limit of the counter is defined by state 2, instead of being hard coded in advance, resulting in arbitrary message length capability.

All of the above versions of the decoder can be found in the Github for this project⁹.

⁹<https://github.com/JayTWilliams/GroundStation/tree/main/Documentation/Spring24/BitProcessing/FinalDecoder>

Conclusion

With capability for messages of any length in byte form, any kind of data can be processed this way. This program can be used for image transmission, for use in the e-paper module, as well as thermocouple data transmission.

While useful, this program is unfortunately very specific to our application, and may be necessary to significantly modify it to accommodate other signals, even with the same modulation scheme. Sync words will be different, and most importantly, the structure of the signal may be entirely different, requiring a total overhaul of the state machine. The GNU Radio blocks will stay mostly the same across different 2FSK signals, only requiring some parameters to be changed.

Written by Jay Williams, April 20th, 2024