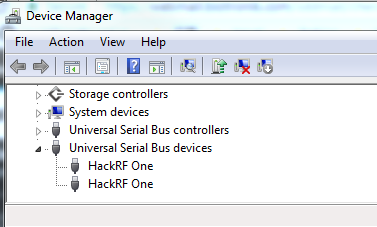
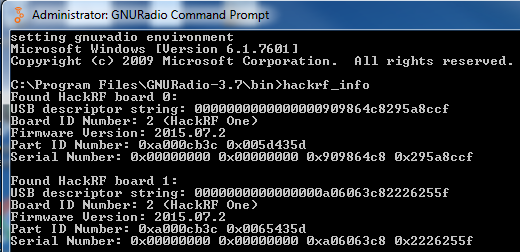
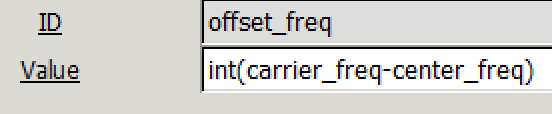
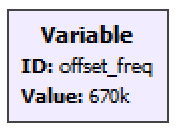
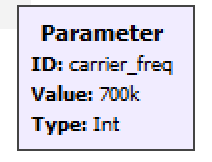
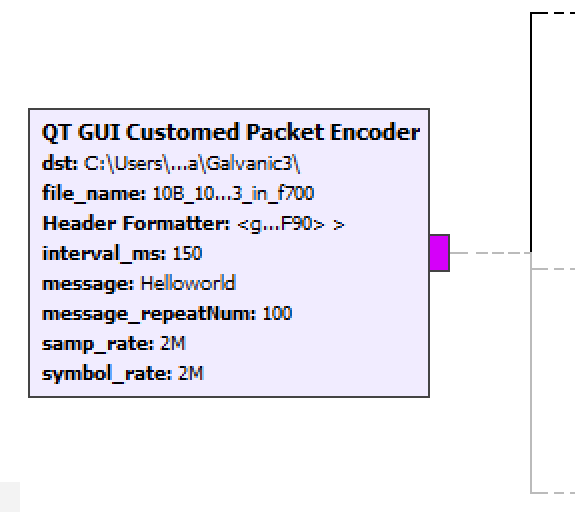
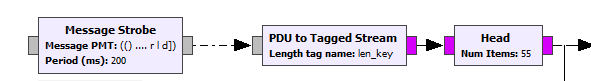
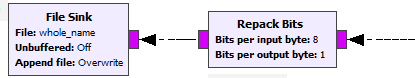
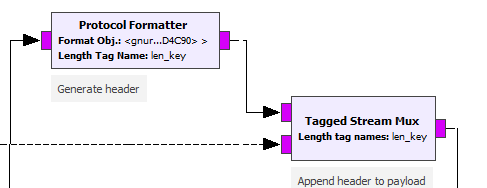
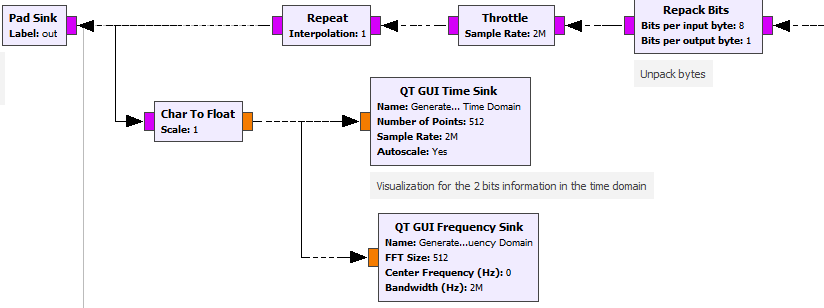
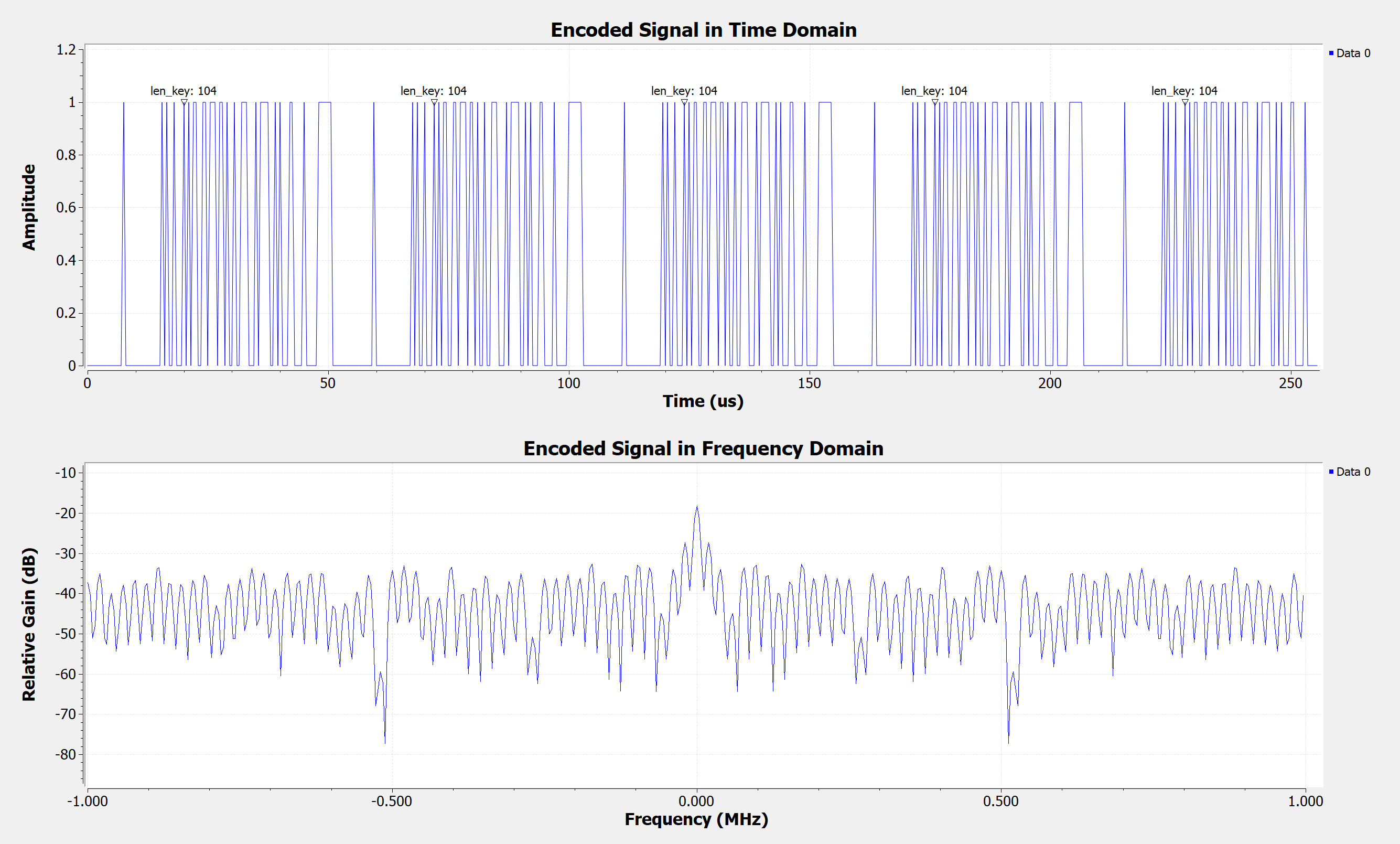
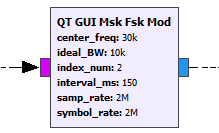
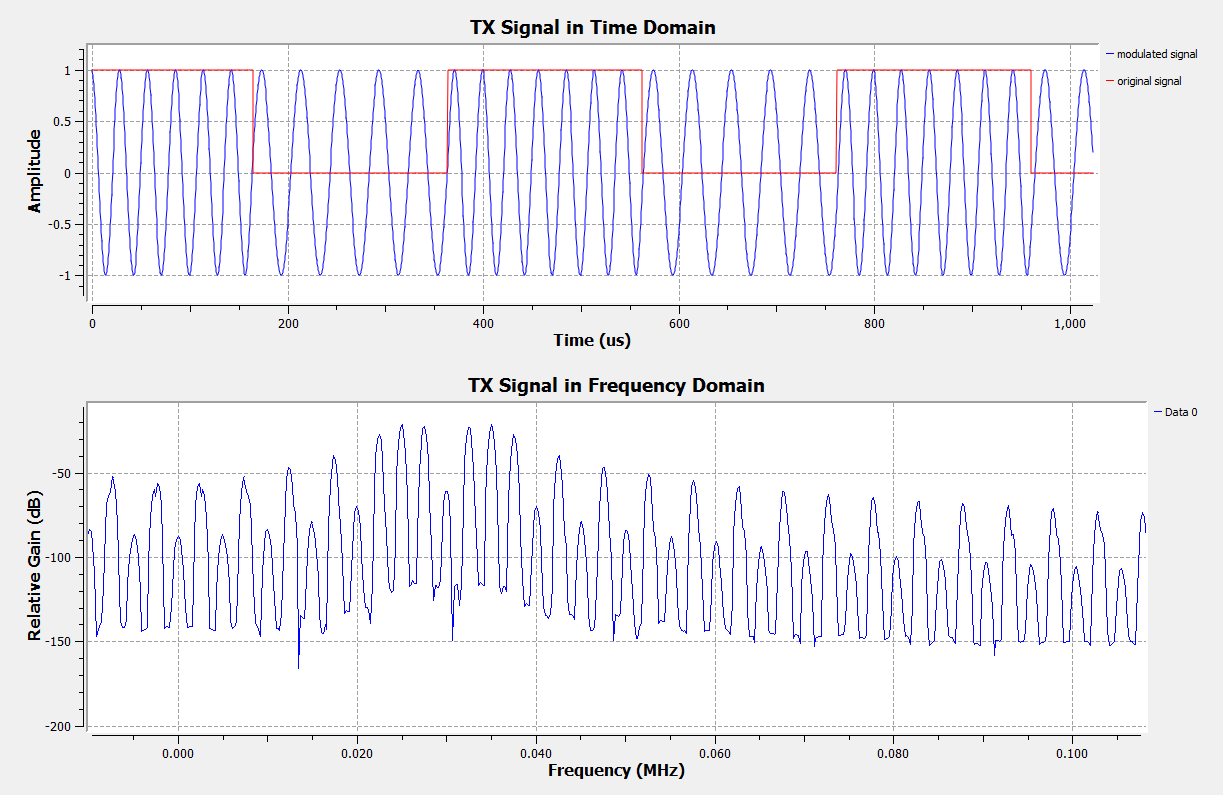
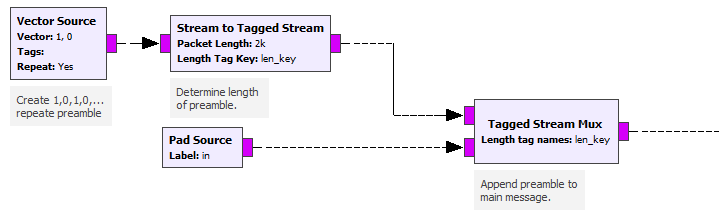
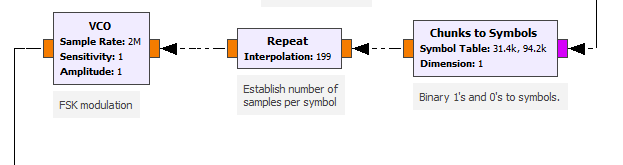
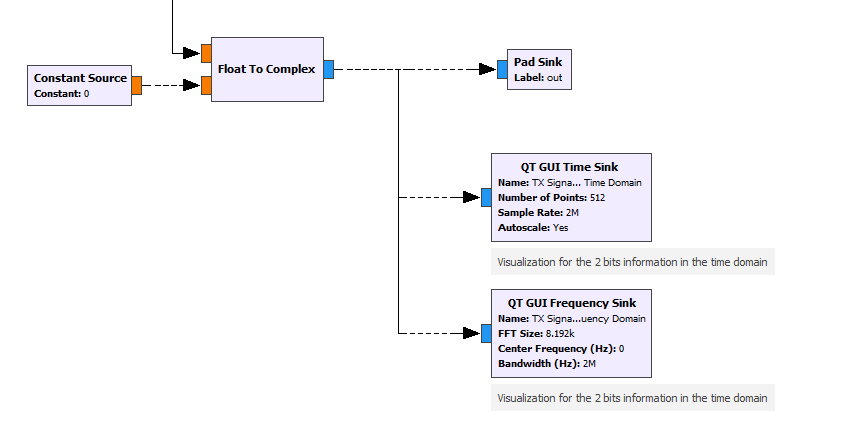
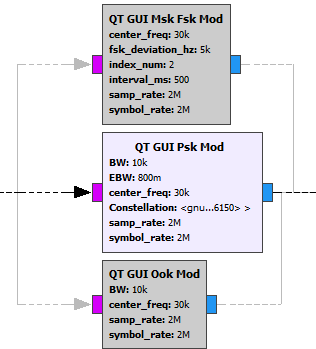
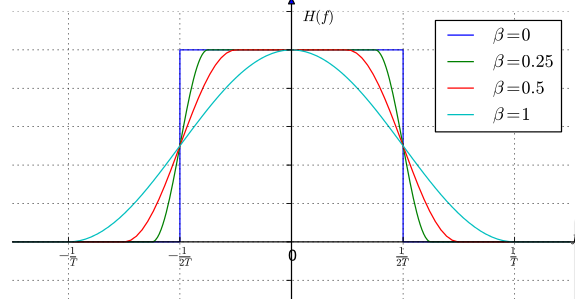
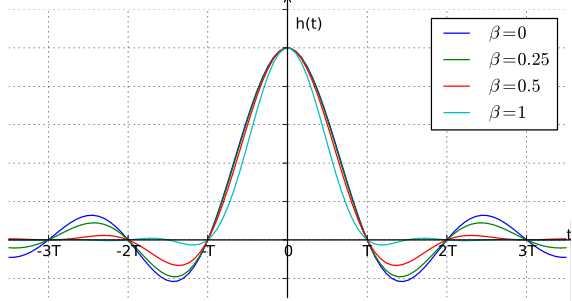
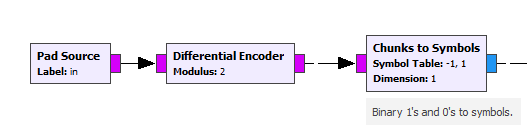
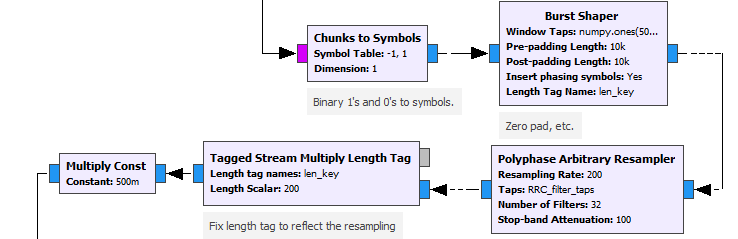
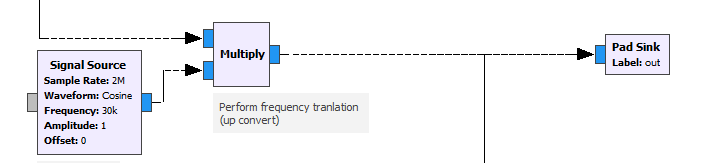
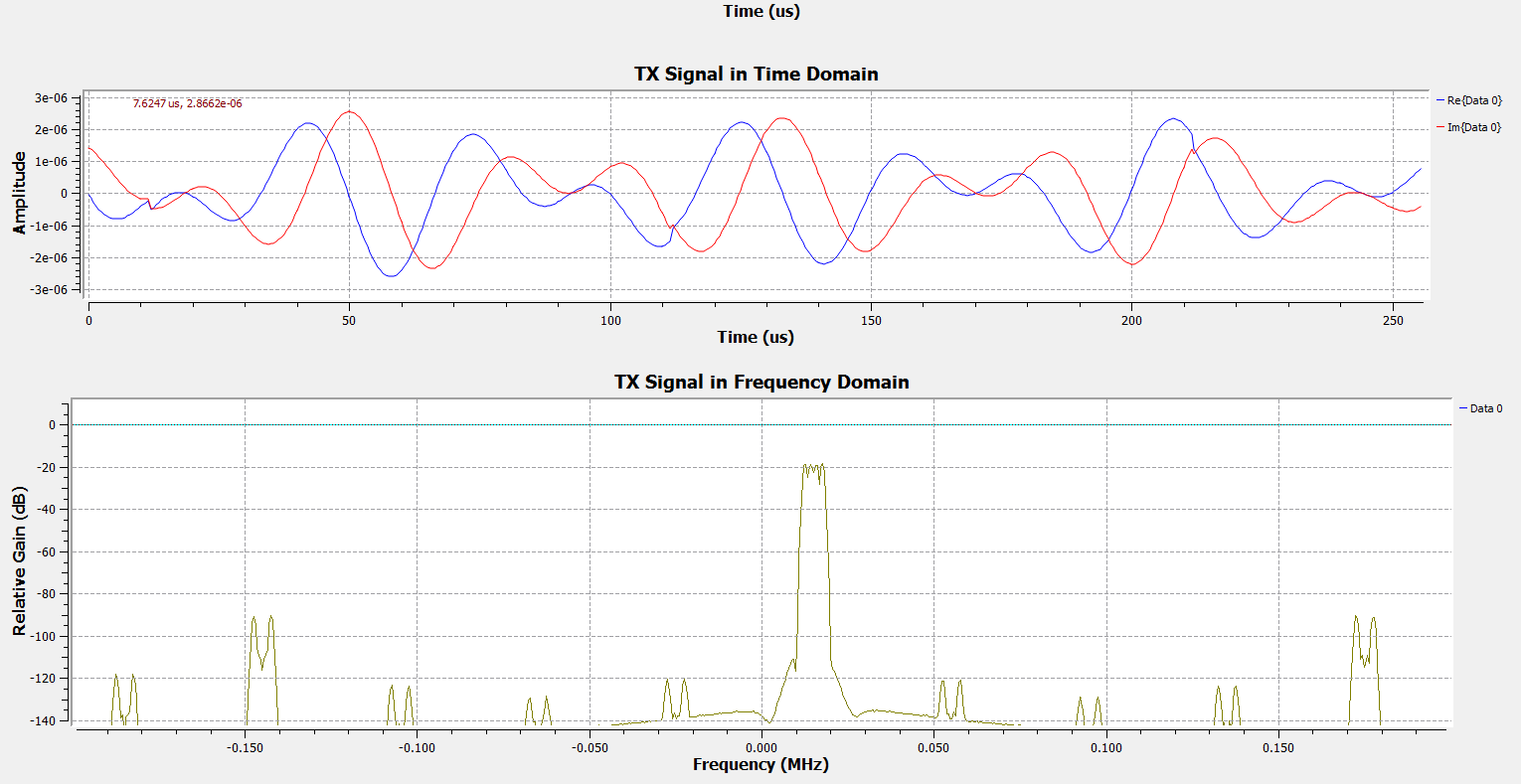
* **Environment Setting ( Windows 7):** 
  + Hackrf One USB Driver: zadig\_v2.0.1.160.7z (<https://sourceforge.net/projects/libwdi/files/zadig/>)
  + GNUradio Windows 64 Binary Version: (<http://www.gcndevelopment.com/gnuradio/>)
    - Because windows version is not officially supported, so my experience is using Out-of-Tree Tools (OOT) <https://wiki.gnuradio.org/index.php/OutOfTreeModules> will cause some issues. Other features work fine.
  + Verify the installation:
    - Plug in Hackrf One
    - You should see it appeared in Device Manager: 
    - Open the GNUradio Command Prompt, type *in ‘hackrf\_info’,*you should see information like this: 
    - Remember the last 8 digits like 2226255f, this is the address you want to call the specific hackrf in the future. Now we are ready to go!
* **System Overview:**
  + One the TX side, there is **Customed Packet Encoder, Modulation Libs and Osmocom Sink (Interface that talk to SDR radio hardware).** One the TX side, there is **Osmocom Source (Interface that talk to SDR radio hardware)., De-modulation Libs and Customed Packet Decoder**

**. A screenshot of a cell phone

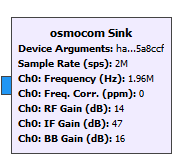
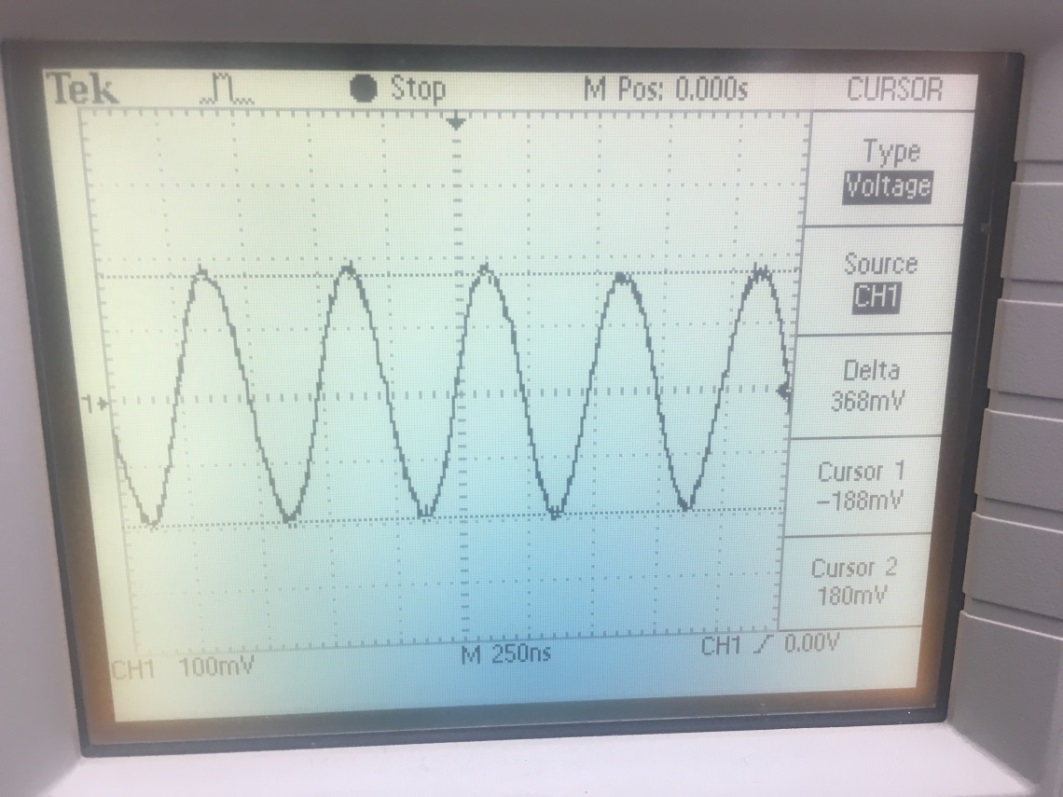
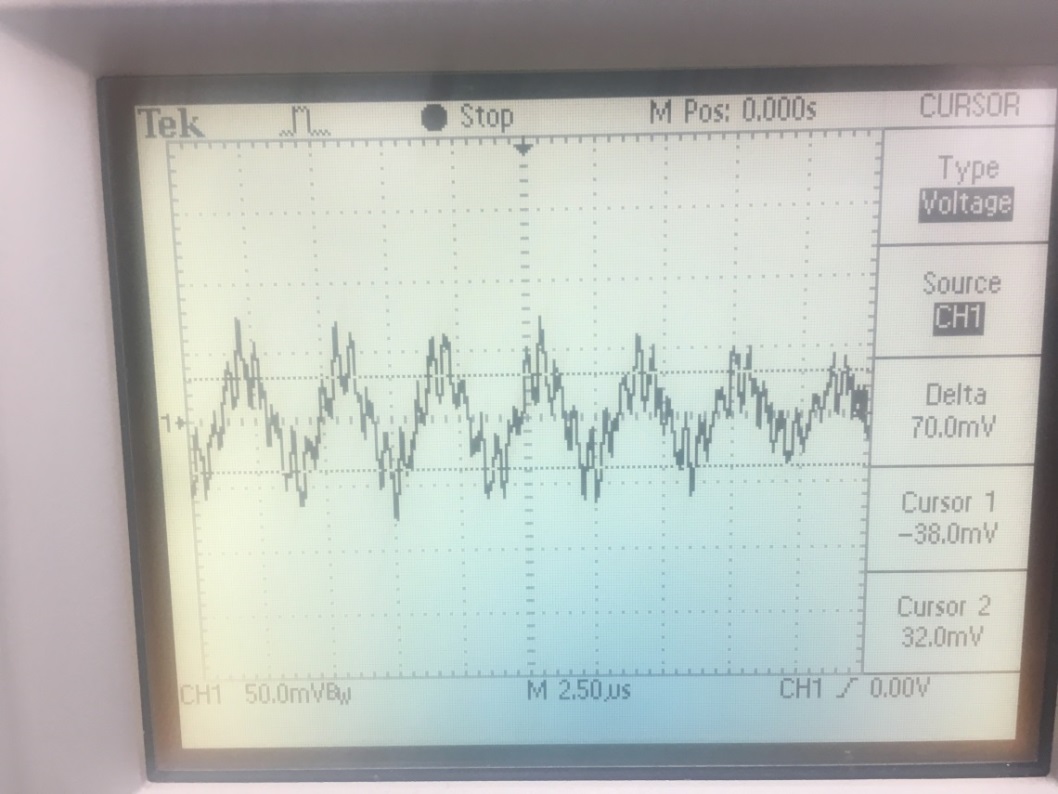
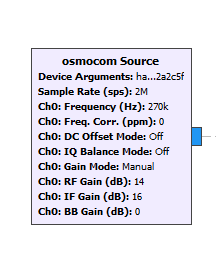
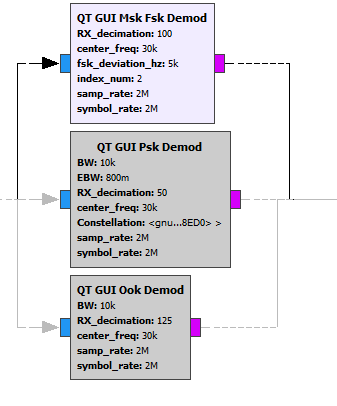
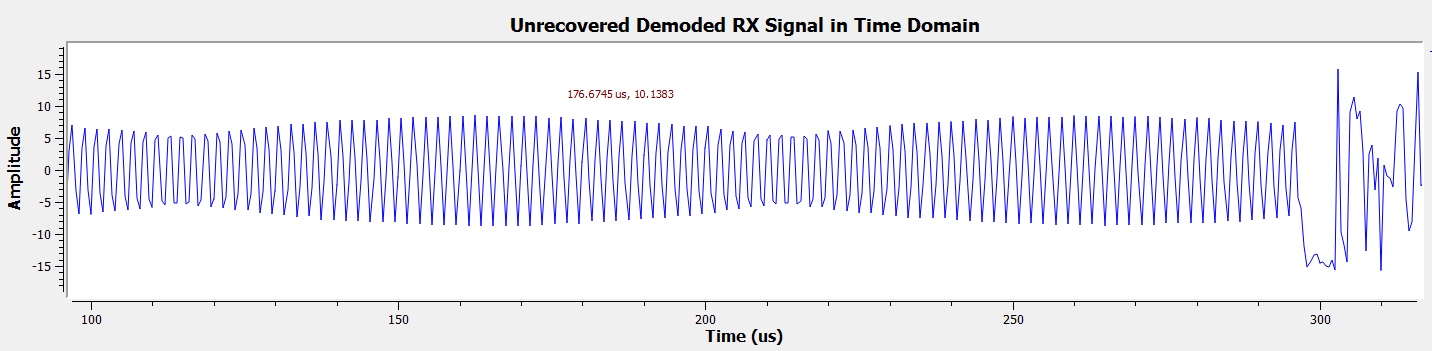
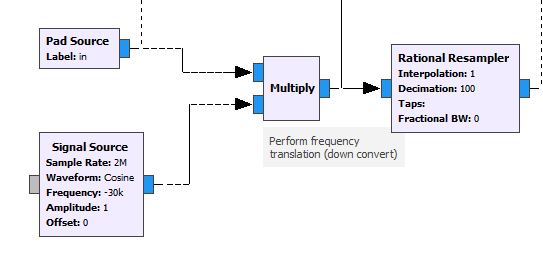
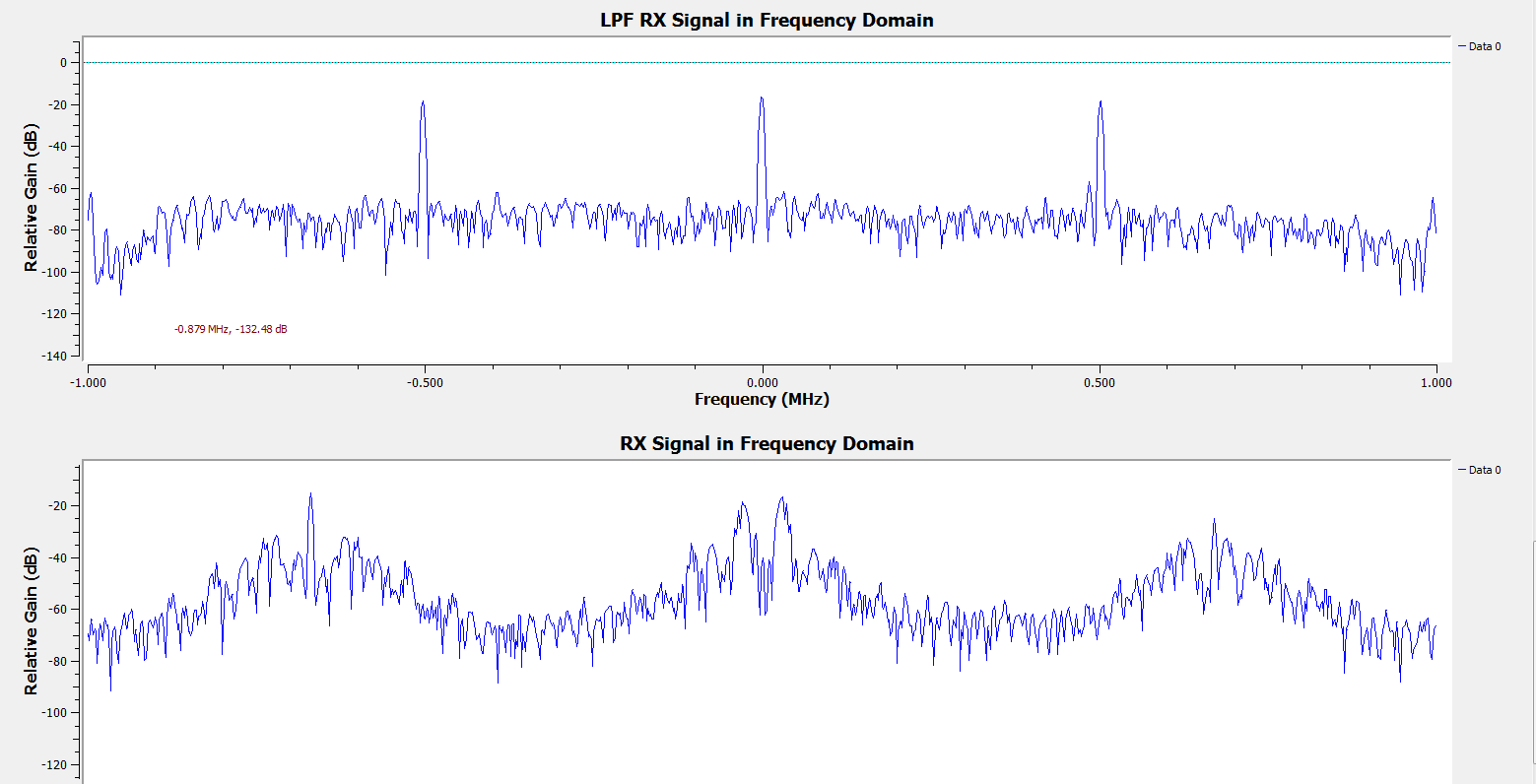
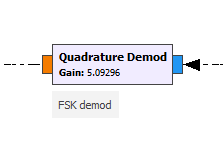
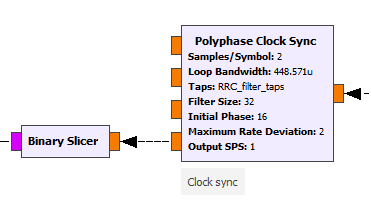
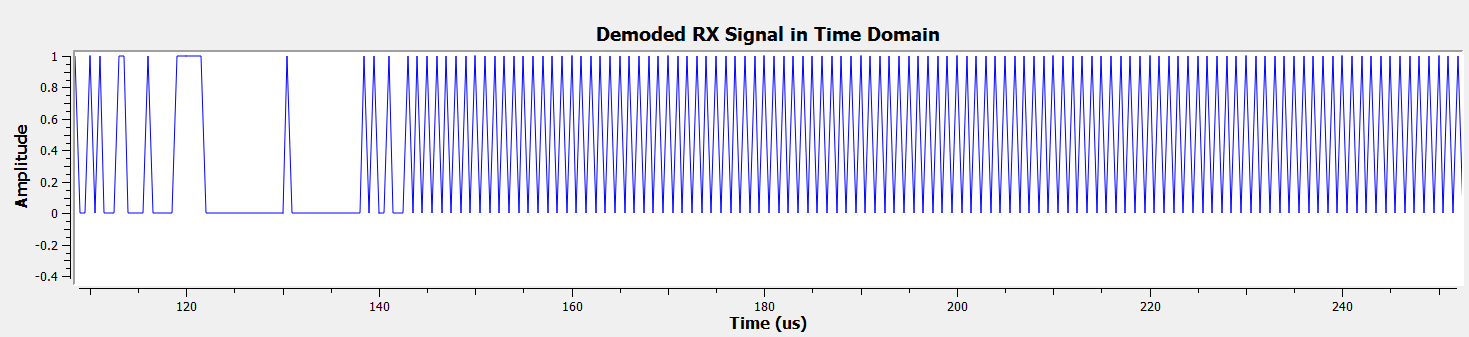
Description generated with very high confidence**

* + How to switch between different modulation/demodulation library?
    - In the GNUradio panel above there is enable/disable options:  (or you can right-click on the block and select enable/disable). Each time you should only enable one corresponding pair of modulation/demodulation block. For example in above picture I enable the FSK method:
  + How to change variant inside each block?
    - Please refer to the following sections for more detail.
  + How to define a global variable/parameter?
    - First the parameter block is the variant you can change and the variable block usually stores the value depends on the parameter block. (Make sure not to change them!): 
    - Some parameters/variables are used multiple times in different block. So instead of change values in these block each time. We define a global variables/parameter for that. For example here the ‘carrier\_freq’ has been used twice. So for convenient purpose, the ‘carrier\_freq’ has already been filled in needed blocks in my system. User just need to change the parameter block.
* **Customed Packet Encoder:**
  + Overview: This block transformed input text messages to packets of binary digits signals
  + 
  + Parameters:
    - **Dst**: Destination folder of all saved signal files
    - **Filename:** The name of the original digital signal transformed from test message
    - **Header Format:** Used to generate the access code that indicates the beginning of each packet of message.
      * GNUradio has build-in function to generate the access code *‘digital.header\_format\_default(digital.packet\_utils.default\_access\_code, 0)’* . Usually no need to adjust.
      * After receiving signals, the computer will check the access code first. If the error number of access code is below the threshold. (For more details about adjust this threshold, please refer to ***[Packet Decoder]*** section.) Then computer will regarded the following signal as target message. This helps us to synchronize the reference signal and the received signal for calculating the Bit Error Rate.
    - **Interval\_ms:**  The interval time between two messages. The unit is ms so ‘200’ means 200ms.
    - **Message:**  The actual text message you want to send. One character equals 1 Byte long. For example, send “Helloworld” is actually sending 80 bits signals.
      * Type: String.
    - **Message\_RepeatTime:** How many times to you want to send the same message.
      * To get an accurate Bit Error rate, I recommend repeat the message >= 100 times.
    - **Sample Rate**: The sample rate of generated signal, represents the number of discrete samples the computer will process per seconds.
      * Require to be >=2\*maximum\_frequency of processing signals.
      * The minimum sample frequency Hackrf One can deal with is 2MHz; other SDR hardware may have other limitations.
      * If later you want to apply the digital method in the chip, sample rate should not within the chip capacity, usually smaller than clock rate.
      * Sample rate is the major influence for power consumption
    - **Symbol Rate:** Number of symbols per second. For example, ‘a’ can be decode as ‘01100001’, if the symbol rate is 100 then each ‘1’/’0’ should last for 0.01 seconds.
      * The symbol rate should > actual need.
      * The symbol rate should < = sample rate, otherwise the computer is unable to process.
  + Workflow Details:
    - 
      * Generate text message for each Interval \_ms *(Interval \_ms = 200 here)* and convert that to digital stream. Use head to only transmit ‘len(message)\*(message\_repeatNum) ‘ digits. *(message\_repeatNum = 5, and len(message) = 11 here)*
    - 
      * Repack the message symbol from 1 byte per symbol into 8 bits with 1 bit per symbol. Save this original signal to the named file. This file can be used as reference for calculating Bit Error Rate later *(whole\_name = dst + filename here)*
    - 
      * Attach the access code to the beginning of the message.
    - 
      * Repack the access-code-attached message symbol from 1 Bytes per symbol into 8 bits with 1 bit per symbol. Send that to the next stage in Pad Sink and also visualize generated signals in Time Domain and Frequency Domain.
  + Example Result Screenshot: 
    - In the example screenshot I send 1 byte message ‘H’. Each message is a packet. The length of access code is always 96 bits. So, the total length of each packet is 104 (96 bits + 8 bits), which indicated on the screen tag at the beginning of each packet. There are 5 tagged packets of signals shown in the Time Domain because in this example we repeat the signal for 100 times and 5 times can be seen in this time range.
* **Modulation/FSK:** 
  + Overview: Frequency Shift Key Modulation algorithm.
  + 
  + Parameter:
    - **Center\_freq:** The center of f0 ( ‘0’ frequency) and f1 ( ‘1’ frequency).
      * For example, in the picture below, I use 25KHz to represent 0 and 35KHz to represent 1, so the center\_freq=30KHz
    - 
    - **Ideal\_BW:** Ideal BandWidth,|f1-f0|/2, in the above example, ideal\_BW = 10KHz
      * In the FSK algorithm, required *|f1-f0|=symbol\_rate\*index\_num/2* to keep orthogonality.
      * *Center\_freq – ideal\_BW/2 > 0*
      * Ideally, |f1-f0|, the ideal Band Width, is the minimum requirement for sample rate if the processing signal is already zero-centered, which means center\_freq =0 . In the practice, we want center\_freq to be some positive number in order to get a better gain. So the minimum requirement for sample rate is *ideal\_BW + center\_freq.*
    - **Index\_num**: All possible numbers that allow ‘0’ and ‘1’ be orthogonal to each other.
      * Must be an positive integer. Not all index\_num will work. Most times index\_num = 1 is too small.
      * When index\_num = 1, the algorithm is MSK (minimum shift key). My recommendation for the system here is index\_num =2.
    - **Interval\_ms**: refer to the **Customed Packet Encoder section**.
      * Must be the same.
    - **Sample\_rate:** refer to the **Customed Packet Encoder section**.
      * Must be the same.
    - **Symbol\_rate:** refer to the **Customed Packet Encoder section**.
      * Must be the same.
  + Dependent Variables:
    - **SPS**: The number of samples per symbol you want to up-sample from 1 sample per symbol.
      * =( symbol\_rate/ideal\_BW/2)\*(index\_num). Must be integer.
      * As I mentioned above, 2\*fsk\_deviation = *|f1-f0|=symbol\_rate\*index\_num/2.* However the symbol rate it required to fulfill the fsk\_deviation needs is usually lower than the actual symbol rate. So we repeat the each sample for *SPS* times to generate some “pseudo symbol rate”.
      * For example the original symbol rate is 2MHz which means each symbol last *1/2M = 0.5e-6 s*, by repeating SPS=200 times, each symbol last *1/2M\*200 = 1e-4 s,* which means the “pseudo symbol rate” *2M/200 = 10KHz.*
    - **Pream\_vec\_len:** 
      * *= samp\_rate\*interval\_ms\*1e-3/SPS*
      * Length of preamble vector. During the interval time, there is only noise existing instead of message. So receiver is unable to recognized targeted message timely when jump from the noise to received message. In order to avoid such error, I send 1,0,1,0 … during the interval time to synchronize the receiver first.
  + Workflow Details:
    - 
      * Attach the preamble 1,0,1,0… signal to the encoded signal packet created from **Customed Packet Encoder (from pad source).**
    - 
      * Match the ‘0’, ‘1’ to f0, f1. Repeat that for *SPS*times up-sampling the and use VCO (Voltage Controlled Oscilloscope) to generate the cosine wave with corresponding frequency according to the input voltage.
    - 
      * Because signal in reality is always in complex form. So change float to complex by inserting 0 into imaginary part. Then send it to the next stage as well as visualize it in the frequency/time domain.
* **Modulation/PSK:** 
  + 
  + Parameter:
    - **Ideal\_BW:**  **:** Ideal BandWidth,|f1-f0|/2, in the above example, ideal\_BW = 10KHz
      * In the PSK algorithm, the ideal case assumed each up-sampled symbol is being perfectly converted to cosine wave.
      * *Center\_freq – ideal\_BW/2 > 0*
      * Ideally, the ideal Band Width, is the minimum requirement for sample rate if the processing signal is already zero-centered, which means center\_freq =0 . In the practice, we want center\_freq to be some positive number in order to get a better gain. So the minimum requirement for sample rate is *ideal\_BW + center\_freq.*
    - **EBW:** Excess Bandwidth. Changing each square-like symbol to the perfect cosine wave is practically impossible. EBW is the tolerance allowed for excess bandwidth for non-ideal case.
      * Here EBW is a normalized factor based on BW (0.0-1.0). For example, 0.8 means 0.8\*ideal\_BW extra bandwidth required. Total required bandwidth = ideal\_BW \*(1+EBW)
      * Here I recommended set to 0.35-0.8
      * Usually lower EBW means more bandwidth efficient and more accurate the re-sampler is. But also means more power consumption.
      * Details about how each bandwidth in frequency domain corresponds to signal in time domain can be see here and the following graphs. (<https://en.wikipedia.org/wiki/Root-raised-cosine_filter>)

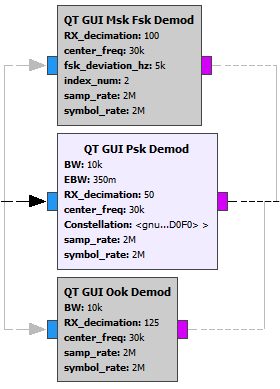
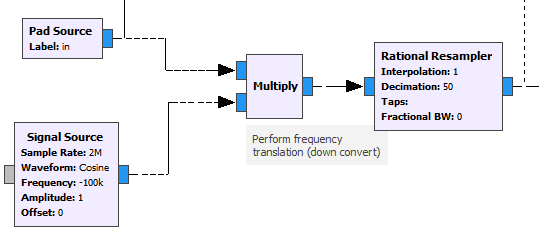
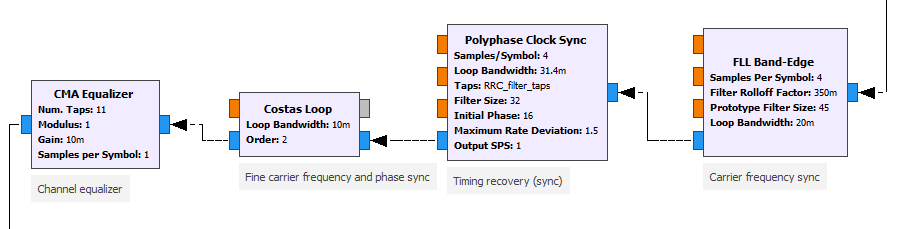
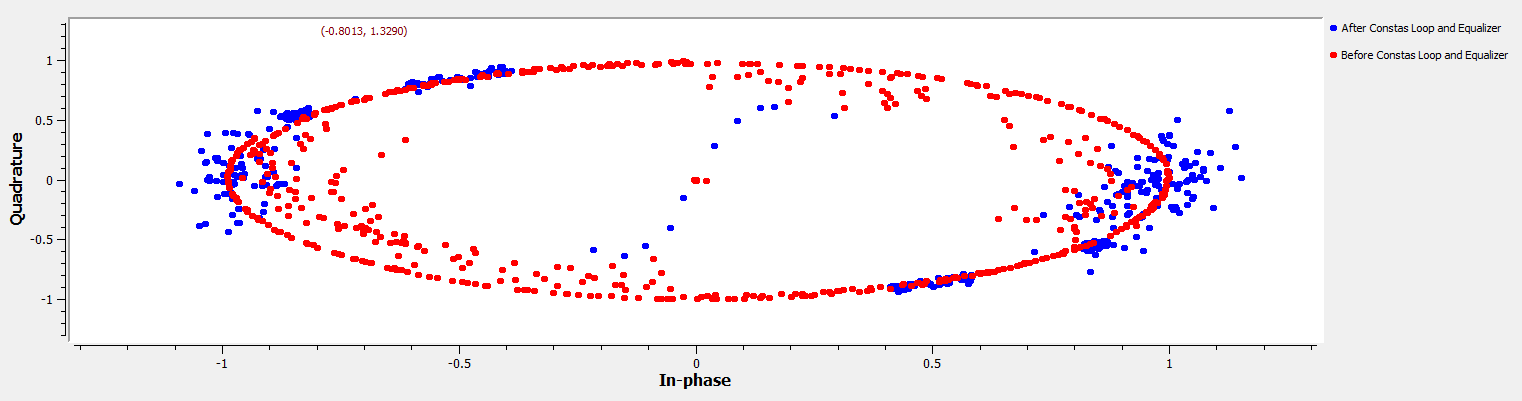
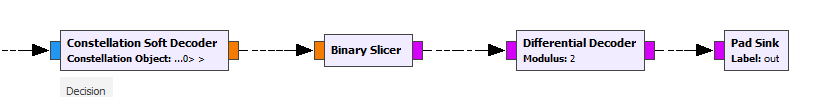
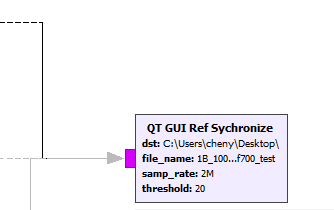
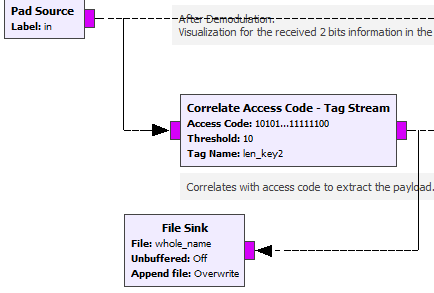
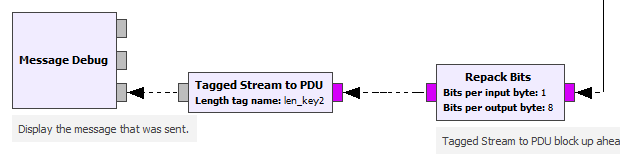
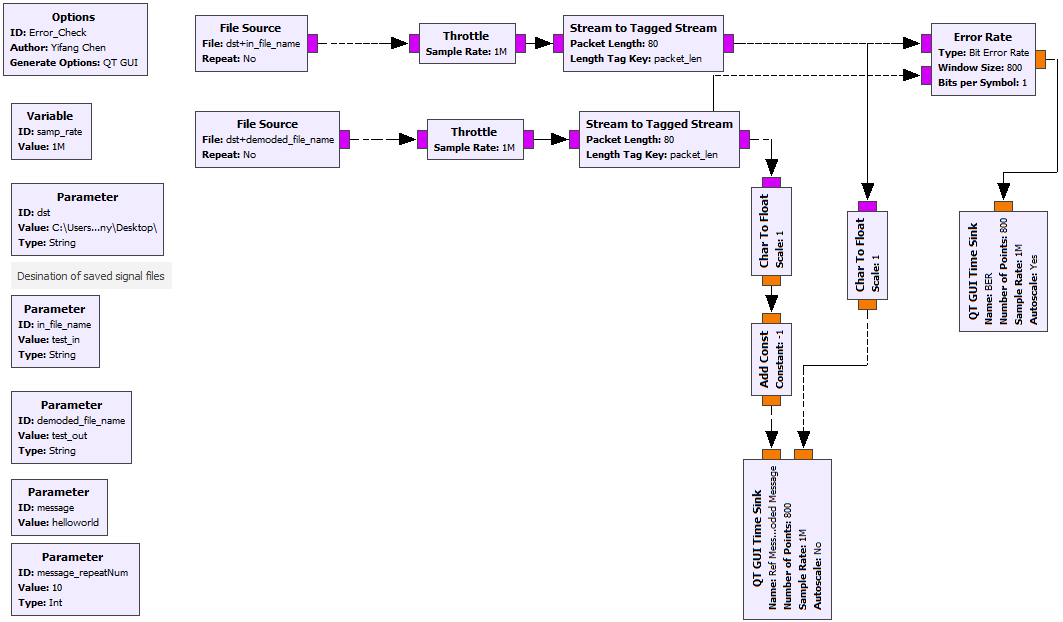
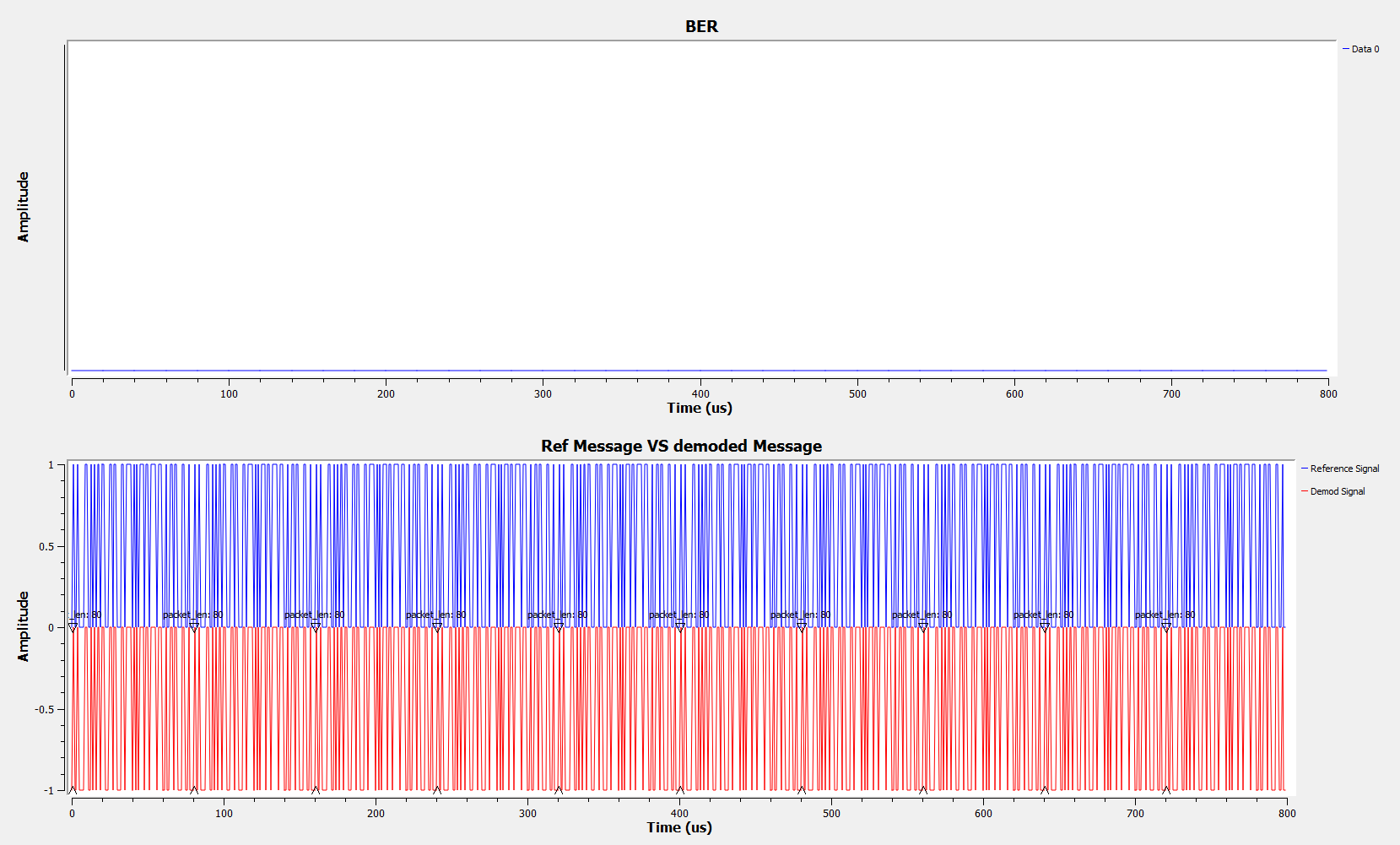
* + - **Center\_freq:** Center frequency.The computer generated cosine-like wave is always zero centered. The center frequency is the new frequency center you want to up-convert the signal to.
      * Center\_freq – (EBW+ideal\_BW)/2 >0
      * Usually no need to change, I recommended 30KHz here.
    - **Constellation:** Default parameter when using Binary PSK (BPSK), (-1,1).
      * No need to change.
    - **Sample\_rate:** refer to the **Customed Packet Encoder section**.
      * Must be the same.
    - **Symbol\_rate:** refer to the **Customed Packet Encoder section**.
      * Must be the same.
  + Dependent Parameter:
    - **SPS:** The number of samples per symbol you want to up-sample from 1 sample per symbol.
      * **=** symbol\_rate/ideal\_BW
      * For PSK, up-sampling the symbol in the time domain will narrow the bandwidth in the frequency domain, which both make transimission bandwidth efficient and allowed the low pass filter at Rx to filter out more noise.
      * For example, I ideal case, the original symbol rate is 2MHz which means each symbol last *1/2M = 0.5e-6* and the required bandwidth is 2MHz, by repeating SPS=200 times, each symbol lasts only *1/2M\*200 = 1e-4 s,* which means we narrow down the required bandwidth to *2M/200 = 10KHz*
    - **RRC Filter Taps:** A bank of defined Root Raised Cosine Filters help to convert between square-like signal and cosine-like signal
      * No need to change.
  + Workflow Details:
    - 
      * Differential Coding is used in no-coherent reception. PSK is very likely to invert the received signal without a way to recognized. By using differential encoding, the decoded value is not only depends on current signal but also previous signal, thus to avoid inverting problem
    - 
      * Match ‘0’,’1’ to the phase ‘-1’ (-pi) and ‘1’ (pi), do the BPSK modulation and up-sampling at the same time via Polyphase Arbitrary Resampler
    - 
      * Up-convert to its center\_freq. Refer to **FSK Modulation section** for more details
  + Result Screenshot:
    - 

Targeted frequency range. Total Bandwidth = BW\*(1+EBW) range

* **Interface to transmitter:**
  + 
  + Parameter:
    - **Device Argument:** The address number of hackrf. Type command “hackrf\_info” to check the hackrf you connected with.
      * Hackrf = …(last 8 digits)
    - **Sample Rate: :** refer to the **Customed Packet Encoder section**.
      * Must be the same.
    - **RF Gain (dB):** 0 or 14 dB; Amplifier gain for the final transmitting frequency
      * 14 dB is the recommended Value
    - **IF Gain (dB):** 0 to 47 dB in 1dB steps; Amplifier gain for intermediate frequency
      * 47dB is the recommended value
    - **BB Gain:**  Not used in TX side. Can fill in whatever value.
    - **In the TX side, there are not much difference in RF and IF gain.**
  + Dependent Variable:
    - **Ch0 Frequency:** 
      * *= carrier\_freq – center\_freq* (see the **Modulation section**)
      * Up-convert the frequency of generated signal to the desired carrier frequency using the analog method inside Hackrf One. For example the suggested carrier frequency for acoustic signal is 270KHz but we only generated 25KHz and 35KHz. So by setting Ch0 Frequency to 240KHz. We are actually transmitting 265KHz and 275 KHz.
  + Workflow Details:
    - Channel Model is used to simulate the channel environment and Osmocom Sink is the interface to control Hackrf as transmitter.
  + Example Result Screenshot:
    - This is what the transmitted signal looks like when the frequency is in the recommended range, for example Fs=2M,Fc=2.01M,Amp: 252mV-632mV/2, , (0dB RF Gain, 16dB IF Gain) 
    - This is what the transmitted signal looks like when the frequency is OUT OF the recommended range, for example Fs=2M, Fs=270K,Fc=280K, Amp: 60mV/2, (0dB RF Gain, 16dB IF Gain) 
    - Other in/out of range signals have similar behavior as these two examples.
* **Interface to Receiver** 
  + 
  + Parameter:
    - **Device Argument:** refer to the Interface to Transmitter
    - **Sample Rate:** refer to the signal generator section.
      * Must <= samp\_ rate of transmitter.
      * If smaller than the Hackrf act as a low pass filter. For example when the sample rate of the transmitter is 2M, which include 0-1MHz signals. 1M sample rate for the receiver will exclude all high frequency signals that are > 0.5MHz.
      * Here I choose the same sample rate as the transmitter. Please refer to … for more details.
    - **RF Gain (dB):** amplifier for the final transmitting frequency
      * Allowed range: 0 or 14 dB;
      * Here I choose the maximum 14
      * …
    - **IF Gain (dB):** 0 to 40 dB in 8dB steps. amplifier for intermediate frequency
      * Here I choose the 16dB because the bigger gain will introduce additional more strong low frequency noise;
      * …
    - **BB Gain:**  0 to 62 in 2dB steps; amplifier at base band stages
      * **…**
      * Here I choose 0dB because…
    - **In the RX side, RF gain should be the first choice.**
  + Dependent Variable:
    - **Ch0 Frequency: *= carrier\_freq – center\_freq (see the Modulation section)***
    - In order to recover the modulated Signals, this hould be exactly the same as transmitter (unless there is frequency offset in the channel which is unusual).
* **Demodulation\_FSK**
  + 
  + Parameters:
    - **RX\_decimation:**  This parameter serves the equivalent function as Low Pass Filter. The maximum frequency it can reserve is samp\_rate/RX\_decimation/2
      * Integer, SPS(of FSK Modulation) % RX\_decimation = 0
      * Fsk\_deviation\*2\*RX\_decimation < samp\_rate
      * After down-sampling (decimation), the frequency domain is extended. Suppose we originally has targeted signal from -5KHz to 5KHz and noise at 20 KHz, by decimation of 100, now +-5KHz becomes +-500KHz and 20KHz becomes 2MHz. Because computer con only capture the frequency <= samp\_rate/2, so we exclude the noise while still preserving the targeted signal. (See more screenshots on workflow section)
      * Please refer to <https://en.wikipedia.org/wiki/Decimation_(signal_processing)> for moew details
    - **Center\_freq:** Please refer to FSK Modulation section.
      * Must be the same as modulation block
    - **Fsk\_deviation:** Please refer to FSK Modulation section
      * Must be the same as modulation block
    - **Index\_num**: **:** Please refer to FSK Modulation section
      * Must be the same as modulation block
    - **Sample\_rate:** Please refer to FSK Modulation section.
      * Must be the same as modulation block
    - **Symbol\_rate:** Please refer to FSK Modulation section
      * Must be the same as modulation block
  + Dependent Variables:
    - **SPS**: Please refer to FSK Modulation section
      * Must be the same as modulation block
    - **Polyphase**
  + Workflow Details
    - 
      * Down-sample (Decimation) the received signal to exclude high frequency noise; Make it zero centered. From the example below, the signal contains 25KHz and 35kHz becomes -500KHz, 500KHz
    - 
      * Quadrature Demodulation is the specific demodulation algorithm for FSK. Buffer is needed to physically implement this demodulation. Please refer to …..
    - 
      * These blocks are used to recover the previous up-sampled signal to its original symbol rate and convert floating point to binary. **Sample/Symbol** **= SPS/RX\_decimation.** Remember in the previous step, in order to exclude the noise we already do down-sample once. So here we only recovered the remained up-sampled factors. (e.g.: Here my SPS = 200, RX\_decimation = 100, so the **Sample/Symbol** we need to recover is 2). PS. My recommendation is Samples/Symbol >=2.
  + Result Screenshot:
    - 

Preamble: 1,0,1,0,…

Target message

* **Demodulation\_PSK**
  + 
  + Parameters:
    - **RX\_decimation:** Please refer to FSK Demodulation Section.
      * Integer, SPS (of PSK Modulation) %RX\_decimation = 0
      * BW\*(1+EBW)\*RX\_decimation < samp\_rate
      * Because this is not frequency demodulation, so it is less subjective to high frequency. Thus in order to save power consumptions, the decimation number can be a little smaller.
    - **BW:** Please refer to **PSK Modulation**. Must be the same.
    - **EBW:** Please refer **to PSK Modulation**. Must be the same.
    - **Center\_freq:** Please refer to **PSK Modulation**. Must be the same.
    - **Constellation:** Please refer to **PSK Modulation**. Must be the same.
    - **Samp\_rate:** Please refer to **Interface to Receiver**. Must be the same.
  + Dependent Variables:
    - **SPS**: Please refer to **PSK Modulation**. Must be the same.
  + Workflow Details:
    - 
      * Down-sample and zero-centered. Please refer to **FSK Modulation.**
    - 
      * … (https://wiki.gnuradio.org/index.php/Guided\_Tutorial\_PSK\_Demodulation)
      * 
    - 
      * After Clean up all the noise and offset, call the demodulator and finally decode the differential coded signal into original one
* **Packet Decoder:**
  + 
  + Parameters:
    - **Dst:** Please refer to function generator. Must be the same
    - **File\_name:** Please refer to function generator.
    - **Samp\_rate:** Please refer to function generator. Must be the same.
    - **Threshold:** The maximum errors allowed when check the access code.
      * My recommendation is 20.
      * The access code indicate the beginning of our targeted message, thus to synchronize the received and demodulated message with the sent message. Too small threshold will only allow the extremely accurate transmission to go through, which let us unable to catch the bit error rate variance. Too big threshold will failed in synchronization.
  + Detailed Workflow:
    - 
      * Check the access code and save the synchronized message to the file for BER test later
    - 
      * Repack the 8 bits into 1 byte, convert the asci code into text and print out. Help us get a general idea of transmission before do quantized testing (BER)
* **Bit Error Check:** 
  + Bit Error Check is in a separate .grc file. You can do Bit Error Check once after you saved all signal files. 
  + Parameter:
    - **Dst:** Please refer to the **Signal Generator Section**. Must be the same.
    - **In\_file\_name:** Please refer to the **Signal Generator Section**. The name of original generated signal file you saved before.
    - **demoded\_file\_name:** Please refer to the **Packet Decoder Section**. The name of demodulated signal file you saved before.
    - **Message:** Please refer to the **Signal Generator Section.** The message you sent in the file you want to assessed. R.g. “helloworld”
    - **Message\_repeatNum:** Please refer to the **Signal Generator Section.** The message repeat times you sent in the file you want to assessed.
      * **Message** and **Message\_repeatNum** are all help to estimate the length to calcualte error rate
  + Result Screenshot:
    - Ideal case: 
    - Non-ideal case: 