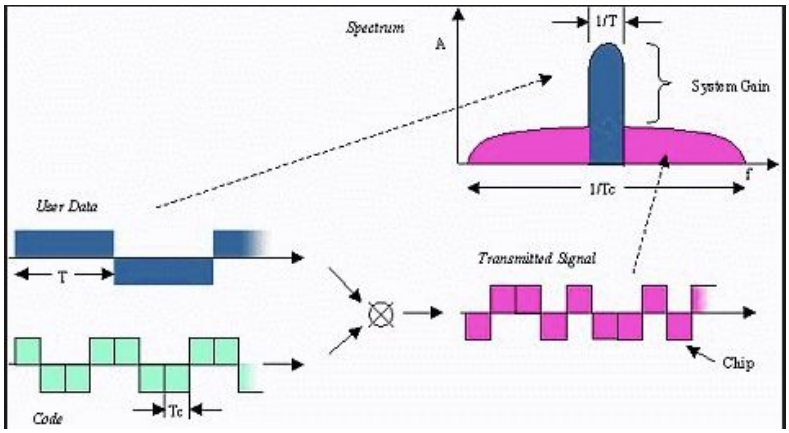
Direct Sequence Spread Spectrum

Arunabh Ghosh 150070006
Chinmay Talegaonkar 15D070046

Index -:

DSSS



Direct sequence spread spectrum is a spread spectrum transmission technique, which makes use of discrete pseudo-random sequences of 1s and -1s to spread the spectral content of the message signal.

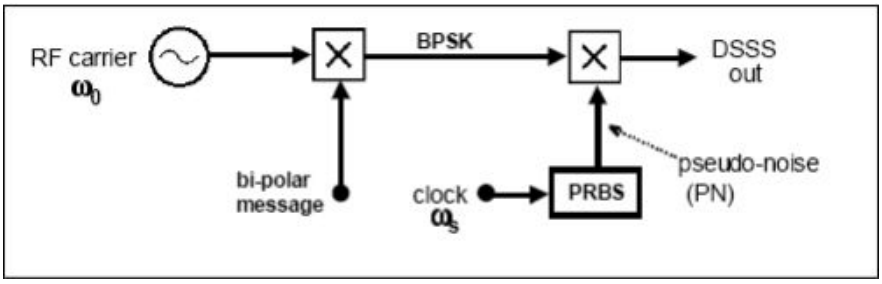
Given a stable clock, and a long sequence, it may be shown that the spectrum of a pseudo-random binary sequence generator (PRBS) is a good source of these carriers. A second PRBS generator, of the same type, clocked at the same rate, and appropriately aligned, is sufficient to regenerate all the required local carriers at the receiver demodulator.

Instead of the total transmitted power being concentrated in the bandwidth of the message signal, the multiple carriers generated by dot multiplying the pseudo-random sequence with the

message signal spreads the power uniformly across the whole frequency spectrum. The signal-to-noise ratio for each DSBSC is very low (well below 0 dB). To recover the message from the transmitted spread spectrum signal all that a receiver requires is thousands of local carriers, at the same frequency and of the same relative phase, all those at the transmitter. All these carriers come from a pseudo-random binary sequence (PRBS) generator. In the spread spectrum context, the PRBS signal is generally called a PN – pseudo noise - signal, since its spectrum, approaches that of random noise. Having the correct sequence at the receiver means that the message contributions from each of the thousands of minute DSBSC signals combine in phase – coherently - and add up to a finite message output. Otherwise, they add with random phases, resulting in a (very) small, noise-like output.

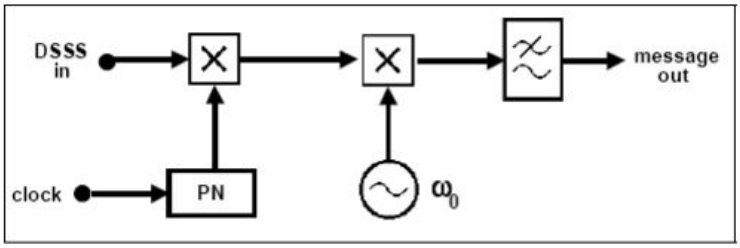
Modulation and Demodulation

DSSS modulation -: The message signal (converted to BPSK)l is multiplied to the carrier wave which we intend to use for transmission. This product is now dot multiplied with the pseudo-random sequence as to produce the spread spectrum signal.



The implementation can be better visualized in the flow graph. We have made a block for the same in GNUradio. The block has one input (the message signal), and two outputs one of which is the final output (input multiplied to the random sequence) and the other output is the pseudo-random sequence. The code and details are given in the references section

DSSS demodulation -: The modulated signal is multiplied to the same pseudo-random sequence with which it was modulated, and this product is then multiplied with the conjugate of the carrier wave signal used for modulation. We have implemented a DSSS_demod block, which takes the 2 outputs of the DSSS modulator block as inputs. Due to delay in receiving the random sequence from the output of the modulator block, we decided to implement the DSSS_demod simply using GNU radio blocks.

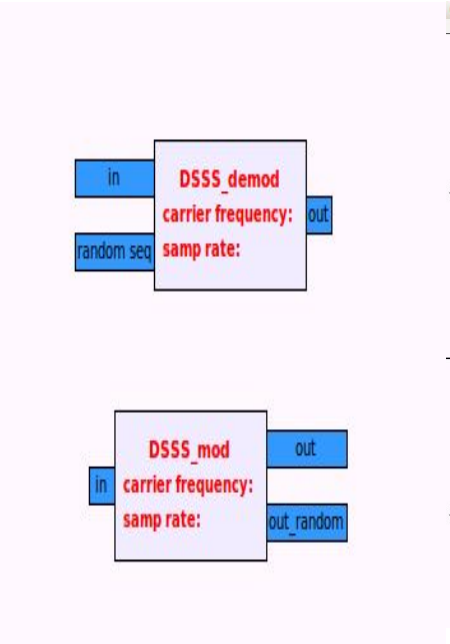


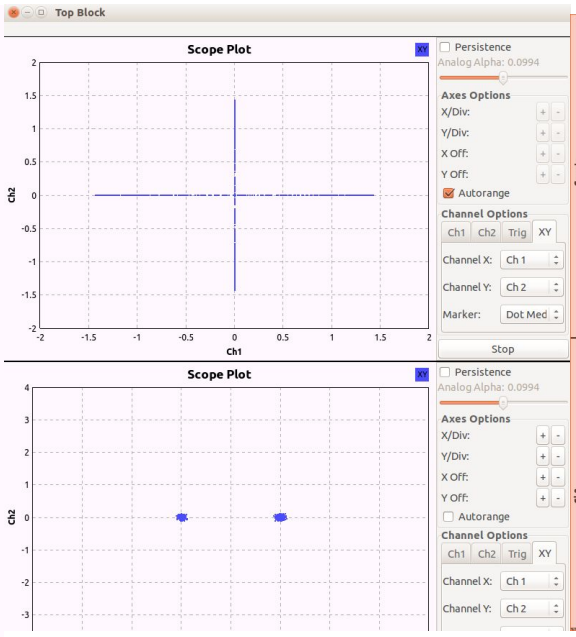
Implementations for the Project

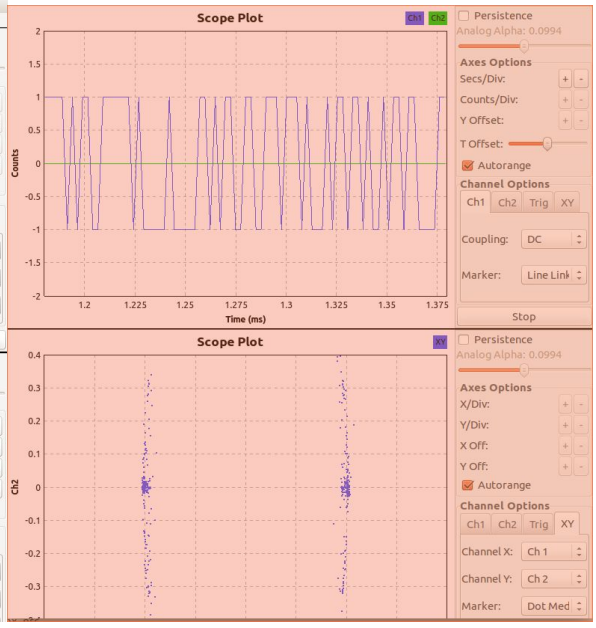
---

1. Implemented a flowgraph for demonstrating the working of DSSS modulation and demodulation schemes using GNU radio blocks. Transmitted and Received BPSK signals (also checked with adding noise) accurately using this flow graph. The file for this flowgraph is *DSSS_mod_demod.grc.*
2. Made the DSSS_mod block in gnu radio, and demodulated using GNU radio blocks only. This gives an accurate reception of constellations. The file for this is *partial_DSSS_mod.grc (image b)*
3. Used the custom-made DSSS_mod and DSSS_demod block in gnu radio to perform modulation and demodulation of BPSK signals. We get some spread in the demodulated BPSK signals from the DSSS_demod block, due to delays. This is done in the file *DSSS.grc*

a) b) c)







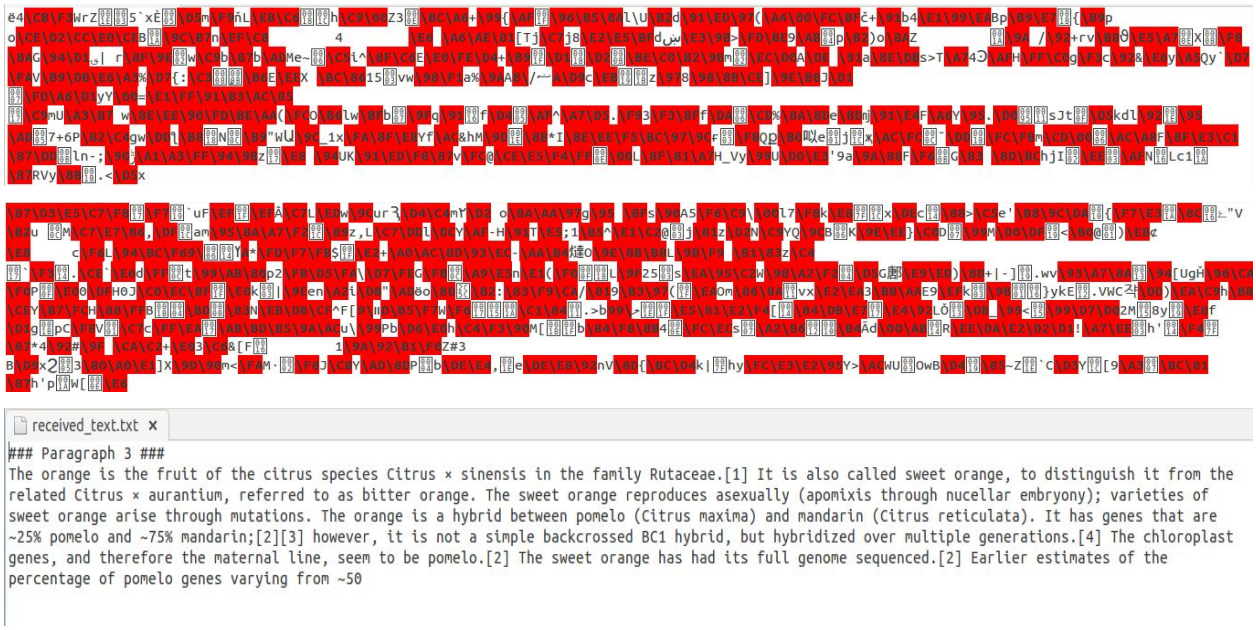
a) DSSS mod and demod Custom blocks

b) The image shows BPSK modulated (upper) And demodulated signals (lower) In 2.

c) The image (lower) shows the constellation which is approx BPSK in 3.

Results

1. We saw that the DSSS algorithm for modulating and demodulating works well in principle for modulating and demodulating BPSK signals, using the GNU radio blocks as a proof of concept
2. Designed and implemented the custom DSSS_mod and DSSS_demod blocks in GNU radio.
3. Used the blocks to demonstrate a security application of DSSS, by transmitting 3 text files using separate DSSS transmitters, and ensuring that the reception is unique for each file at a transmitter



Here we see that the last image is the correct reception of the file 3 by receiver 3, while the others are receptions by receiver 1 and 2. This proves the advantage of orthonormal pseudo-random sequences. Since the Random sequences used by each of them are unique and orthonormal, only the signal which was modulated using the random sequence used by a particular receiver can be detected completely using that receiver. Files corresponding to the text messages modulation and demodulation are *text_mod1.grc, text_mod2.grc, text_mod3.grc and text_demod.grc.*

Applications

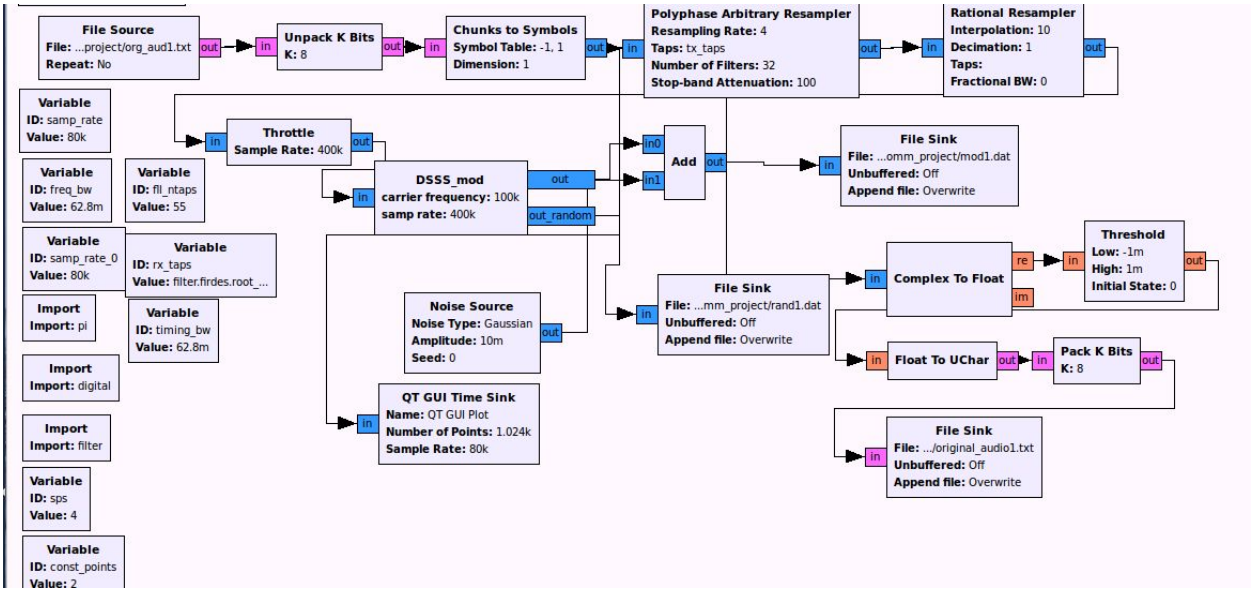
DSSS finds widespread applications in many areas like CDMA transmission, signal jamming, to avoid signal interception and to be used in satellite communications. All these applications make use of the property of DSSS that multiple signals can be spread across the same frequency spread, and can be uniquely decoded using the unique RN sequence corresponding to a signal.

Implementation details

We face the following challenges primarily in the implementation of the DSSS scheme for transmission of text files.

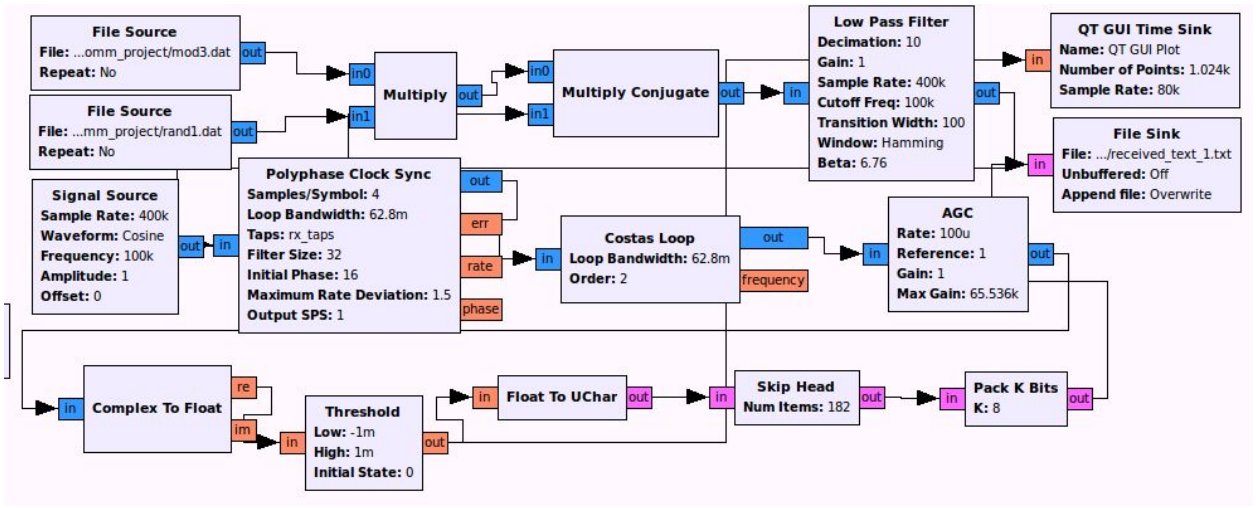
- **Polyphase clock sync** -: We found that when receiving a signal using a polyphase clock sync, there are some garbage values added to the start of the message. Hence, this problem leads to incorrect byte representation of the binary file. It is this problem, which makes the song unplayable.
- Delay in sending the random sequence to the earlier made demod block, because of which we chose to implement it using GNU radio blocks only.
- Tuning the number of heads in the **'skip-head'** block, to remove the garbage inserted by the polyphase clock sync. This is comparatively easier for text than for sound files.

Text file modulation -:



Every text file (org_aud1,2,3.txt) has a corresponding grc file (text_mod1,2,3.grc) for DSSS modulation. The text modulation flowgraph likes the above figure. The text file is read byte-wise and hence is unpacked to 8 bits using the **Unpack K bits** block. The bits are then mapped to BPSK constellation using the standard **chunks to symbols, resamplers, etc blocks**. The BPSK symbols are then modulated using the **DSSS_mod** block. The random sequence and the output modulated signal, generated by this block are stored in files *rand1(2,3).dat* and *mod1(2,3).dat* files respectively.

Text file demodulation -:



The *text_demod.grc* file contains 3 such units as shown in the figure above. We take mod3.dat and rand1.dat here using the **file source** block (only as an example), **multiply** them, and the output is then multiplied to a conjugate of the of the carrier wave (from the signal source) using the **multiple conjugate** block. The output is passed through the **LPF, Polyphase clock sync, Costas loop** and **AGC** blocks as to get the demodulated output in the form of the required constellation. We take the real part of the output using the **complex to float block** (since BPSK maps symbols to -1 and 1 which are real). We also use the **Threshold** block to map the BPSK signals to binary 0,1 bits. **The Float To Uchar** converts the float output to character output, which is required for reading text files. **Skip Head** block is used to remove the garbage bits added to the start of the file by the Polyphase clock sync to the demodulated output, because of the mapping of 8 bits to a byte is corrupted, and we get garbage values at the output. The tuning for the skip head is first done manually checking between 0 to 7. For the value for which we get some legible characters, we then advance that value by counts of 8, to then see what gives the best result. A better method, compared to manual tuning would be to check for a pilot sequence corresponding to the start of the input in the file and then use it to find the beginning of the corrupted output given by the polyphase clock sync. After skipping the number of bits specified in the Skip head block, **Pack K Bits is** used to pack the 8 bits into a byte, which are then written to the output files recieved_text(_1,_2,_3).txt.

Installing the GNU radio blocks

The other GRC files are straightforward to understand. To include the custom-made blocks into GNU radio on your system, use the folders *gr-block_test* and *gr_demod.*

Go the build directory in either of folders. Then use the following commands in succession

a. cmake ../
b. make
c. sudo make install
d. Sudo ldconfig

If this does not work, go to the CMakeLists.txt file in the *lib* folder in the above-given folders. You will see a comment 'this has been added as ...' . Remove that line and try the process again. If it does not work, refer to the GNU radio docs.

If this also does not work, make a dummy block of your own, of type sync (-sync) and required arguments. Use the *python (* block_test.py (for mod block) and DSSS_demod.py (for demod block)) and *grc* folders (to get the required XML files) in the above-given folders to get the required code for the DSSS_mod and DSSS_demod blocks. Refer to the GNU radio docs for a better idea of how to incorporate our blocks or their code in a custom block in case the installation fails.