OFDM Tranmission and Reception of Packets using GNU-Radio and USRP - *Communications Lab Project*

Kshitiz Bansal[1] **and Vishrant Tripathi**[1]

[1]Electrical Engineering, IIT, Bombay
[*]Roll no - 130070029 and 130070035

ABSTRACT

The aim of our project was to develop GRC-flowgraphs to transmit and receive packet data through Orthogonal Frequency Division Multiplexing (OFDM). We built our own blocks in GNU-Radio for cyclic prefixing and power allocation in sub-carrier bands. We also demonstrated some advantages and disadvantages of OFDM compared to other digital modulation techniques.

Introduction

OFDM is a frequency-division multiplexing (FDM) scheme used as a digital multi-carrier modulation method. A large number of closely spaced orthogonal sub-carrier signals are used to carry data on several parallel data streams or channels. Each sub-carrier is modulated with a conventional modulation scheme (such as quadrature amplitude modulation or phase-shift keying) at a low symbol rate, maintaining total data rates similar to conventional single-carrier modulation schemes within the same bandwidth.

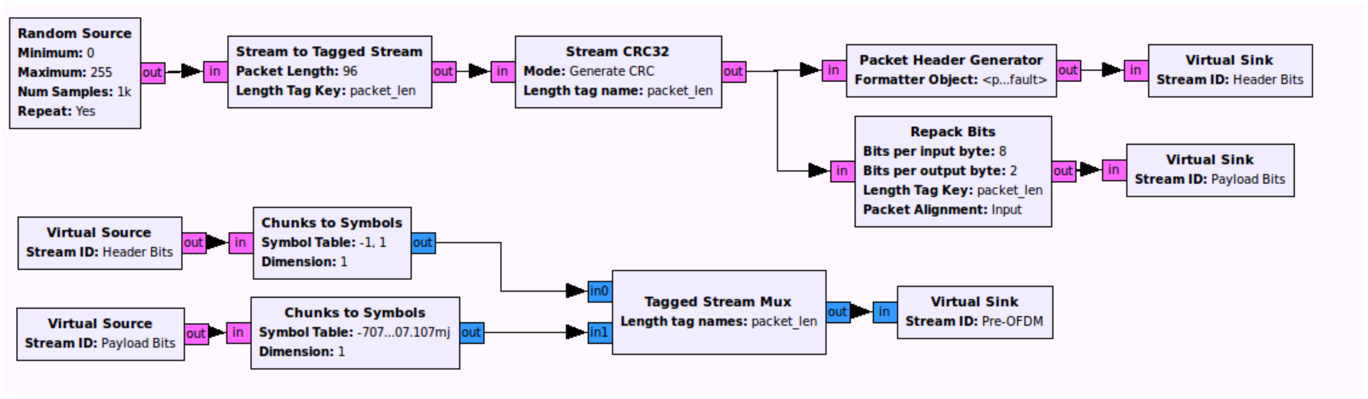
The primary advantage of OFDM over single-carrier schemes is its ability to cope with severe channel conditions, eliminate Inter Symbol Interference (ISI) using cyclic prefixes and better spectrum usage. Due to these advantages OFDM has become a standard in applications involving wideband digital communication like wireless networks (802.11 wifi), 4G, etc.

Flowgraphs

Transmission

The transmission flowgraph is shown in Figures 1 and 2. We convert a stream of digital data into packets and append 'headers/tags' to these packets for identification. This is followed by converting the data into a constellation scheme as shown in Figure 1. Further processing to convert it into OFDM signal to transmit through the channel is done by the flow graph shown in Figure 2.

A group of channel-coded bits are gathered together (1 for BPSK, 2 for QPSK, 4 for 16-QAM, etc.) and mapped to corresponding constellation points. At this stage, the data are represented as complex numbers and they are in serial. Known pilot symbols mapped with known mapping schemes can be inserted at this moment. A serial to parallel conversion is then applied and IFFT operation is performed on the obatined parallel complex data. The output data of this operation are grouped together again, as per the number of required transmission subcarriers. Cyclic prefix is inserted in every block of data according to the system specification and the data are multiplexed in a serial fashion. At this stage, the data are OFDM modulated and ready to be transmitted. A USRP is used to transform the time-domain digital data, up-convert it to transmission frequency and to send it across a wireless channel.


Random Source
Minimum: 0
Maximum: 255
Num Samples: 1k
Repeat: Yes

Stream to Tagged Stream
Packet Length: 96
Length Tag Key: packet_len

Stream CRC32
Mode: Generate CRC
Length tag name: packet_len

Packet Header Generator
Formatter Object: <p...fault>

Virtual Sink
Stream ID: Header Bits

Repack Bits
Bits per input byte: 8
Bits per output byte: 2
Length Tag Key: packet_len
Packet Alignment: Input

Virtual Sink
Stream ID: Payload Bits

Virtual Source
Stream ID: Header Bits

Chunks to Symbols
Symbol Table: -1, 1
Dimension: 1

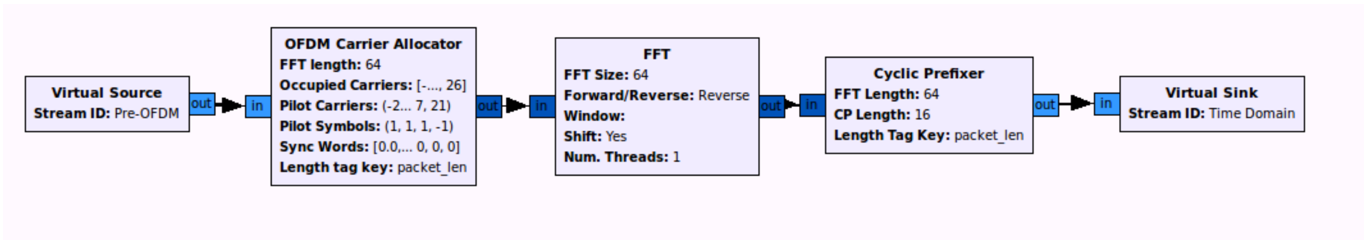
Virtual Source
Stream ID: Payload Bits

Chunks to Symbols
Symbol Table: -707...07.107mj
Dimension: 1

Tagged Stream Mux
Length tag names: packet_len

Virtual Sink
Stream ID: Pre-OFDM

Figure 1. Mapping the data in packets for transmission


Virtual Source
Stream ID: Pre-OFDM

OFDM Carrier Allocator
FFT length: 64
Occupied Carriers: [-..., 26]
Pilot Carriers: (-2... 7, 21)
Pilot Symbols: (1, 1, 1, -1)
Sync Words: [0.0,... 0, 0, 0]
Length tag key: packet_len

FFT
FFT Size: 64
Forward/Reverse: Reverse
Window:
Shift: Yes
Num. Threads: 1

Cyclic Prefixer
FFT Length: 64
CP Length: 16
Length Tag Key: packet_len

Virtual Sink
Stream ID: Time Domain

Figure 2. Implementing OFDM and transmission

Reception

Figures 3 and 4 describe the flowgraphs for receiving the transmitted OFDM signal, converting it back to the constellation scheme, and extracting the payload bits out of the received packets.

After the transmission of OFDM signal from the transmitter antenna, the signals go through all the anomaly and hostility of wireless channel. After receiving the signal, the receiver USRP down-converts the signal and converts it to digital domain. While down-converting the received signal, carrier frequency synchronization is performed. After this conversion, symbol timing synchronization is achieved. An FFT block is then used to demodulate the OFDM signal. After that, channel estimation is performed using the demodulated pilots. Using the estimations, the complex data are obtained which are demapped according to the transmission constellation diagram.

Details of in-built OFDM Blocks

Carrier Indexing and FFT-Shifting

In all cases where OFDM symbols are passed between blocks, the default behaviour is to FFT-Shift these symbols, i.e. that the DC carrier is in the middle (to be precise, it is on carrier $\lfloor N/2 \rfloor$ where N is the FFT length and carrier indexing starts at 0). The reason for this convention is that some blocks require FFT-shifted ordering of the symbols to function, and for consistency's sake, this was chosen as a default for all blocks that pass OFDM symbols. Also, when viewing OFDM symbols, FFT-shifted symbols are in their natural order, i.e. as they appear in the pass band.

Carriers are always index starting at the DC carrier, which has the index 0 (you usually don't want to occupy this carrier). The carriers right of the DC carrier (the ones at higher frequencies) are indexed with 1 through N/2-1 (N being the FFT length again). The carriers left of the DC carrier (with lower frequencies) can be indexed -N/2 through -1 or N/2 through N-1. Carrier indices N-1 and -1 are thus equivalent. The advantage of using negative carrier indices is that the FFT length can be changed without changing the carrier indexing.

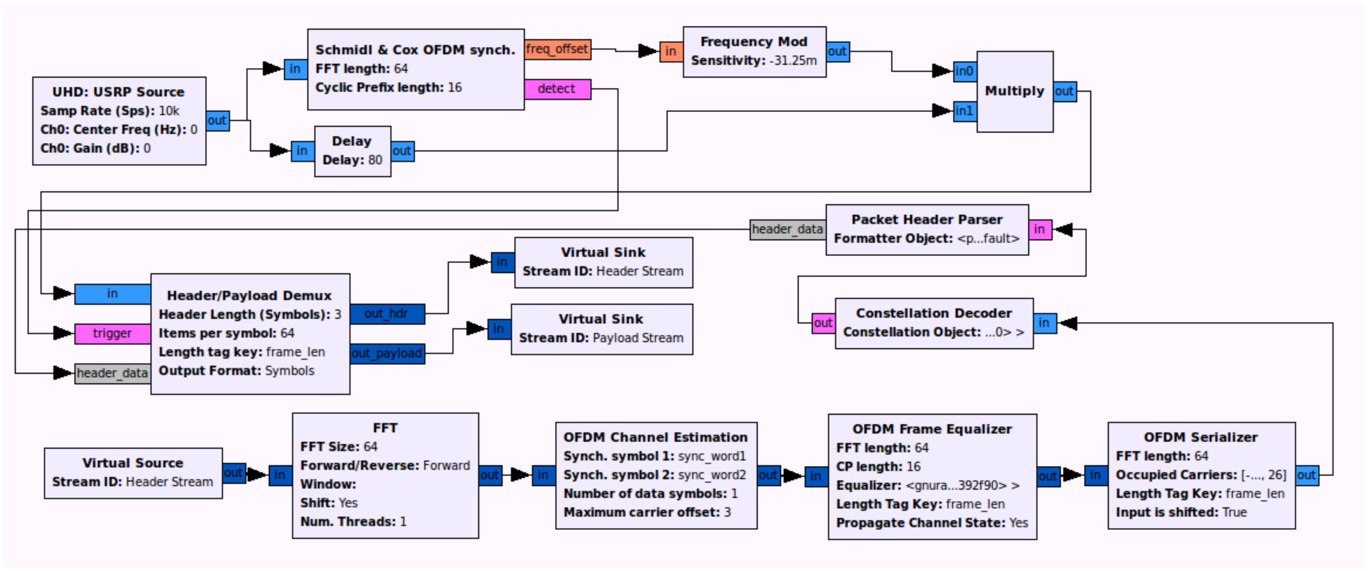


Figure 3. Reception and equalization through frames

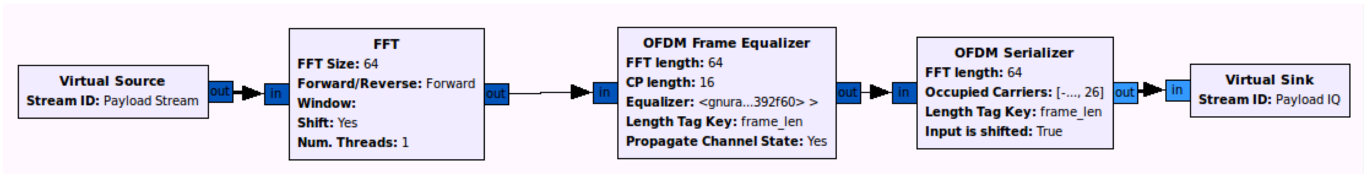


Figure 4. Fetching data back after demodulation

Carrier and Symbol Allocation

Many blocks require knowledge of which carriers are allocated, and whether they carry data or pilot symbols. GNU Radio blocks uses three objects for this, typically called occupied_carriers (for the data symbols), pilot_carriers and pilot_symbols (for the pilot symbols). Pilot carriers and symbols are used for synchronization, channel estimation and equalization and the choice of their values is an important design issue. Here, we used the pilot symbols and carrier scheme used in the IEEE 802.11 standard.

Every one of these objects is a vector of vectors. occupied_carriers and pilot_carriers identify the position within a frame where data and pilot symbols are stored, respectively. The allocation of input sequence of constellation points to the occupied carriers and pilot symbols to pilot symbols is done by the block OFDM-carrier-allocator.

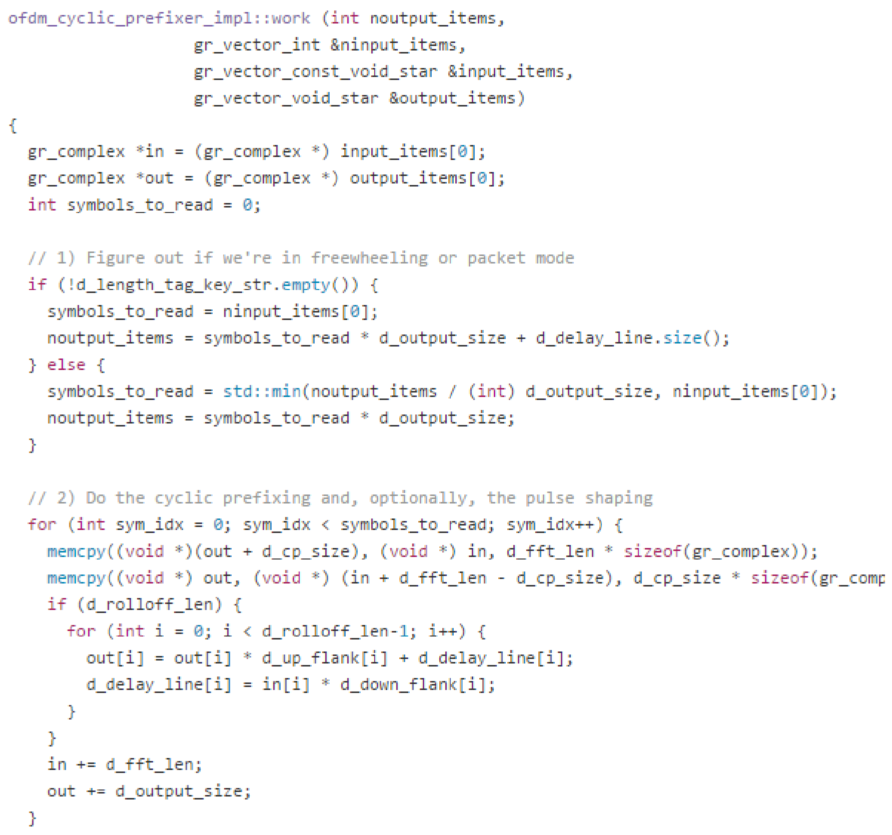
Our Blocks

A major aim of this project, apart from understanding how OFDM works, was to ”look under the hood” of GNU-Radio - understand how to make new signal processing blocks in GNU-radio and add them to the module tree for use in flowgraphs. For this, we coded two blocks for our project - OFDM cyclic prefixer and OFDM power allocator (water-pouring algorithm).

Creating a new block (an out-of-tree-module) involves using the gr-modtool utility in conjunction with knowledge of python (for testing), shell scripting (for compiling and building), C/C++ (for actual signal processing) and XML (for the interface hand-shaking). This also involved understanding the GRC feature of GNU-Radio which generates the python code from the flowgraph and decides how various modules interact with one another to get the desired result.

Cyclic Prefixer

In telecommunications, the term cyclic prefix refers to the prefixing of a symbol with a repetition of the end. Although the receiver is typically configured to discard the cyclic prefix samples, the cyclic prefix serves two purposes.



```
ofdm_cyclic_prefixer_impl::work (int noutput_items,
                   gr_vector_int &ninput_items,
                   gr_vector_const_void_star &input_items,
                   gr_vector_void_star &output_items)
{
  gr_complex *in = (gr_complex *) input_items[0];
  gr_complex *out = (gr_complex *) output_items[0];
  int symbols_to_read = 0;

  // 1) Figure out if we're in freewheeling or packet mode
  if (!d_length_tag_key_str.empty()) {
    symbols_to_read = ninput_items[0];
    noutput_items = symbols_to_read * d_output_size + d_delay_line.size();
  } else {
    symbols_to_read = std::min(noutput_items / (int) d_output_size, ninput_items[0]);
    noutput_items = symbols_to_read * d_output_size;
  }

  // 2) Do the cyclic prefixing and, optionally, the pulse shaping
  for (int sym_idx = 0; sym_idx < symbols_to_read; sym_idx++) {
    memcpy((void *)(out + d_cp_size), (void *) in, d_fft_len * sizeof(gr_complex));
    memcpy((void *) out, (void *) (in + d_fft_len - d_cp_size), d_cp_size * sizeof(gr_comp
    if (d_rolloff_len) {
      for (int i = 0; i < d_rolloff_len-1; i++) {
        out[i] = out[i] * d_up_flank[i] + d_delay_line[i];
        d_delay_line[i] = in[i] * d_down_flank[i];
      }
    }
    in += d_fft_len;
    out += d_output_size;
  }
```

Figure 5. Cyclic Prefixer

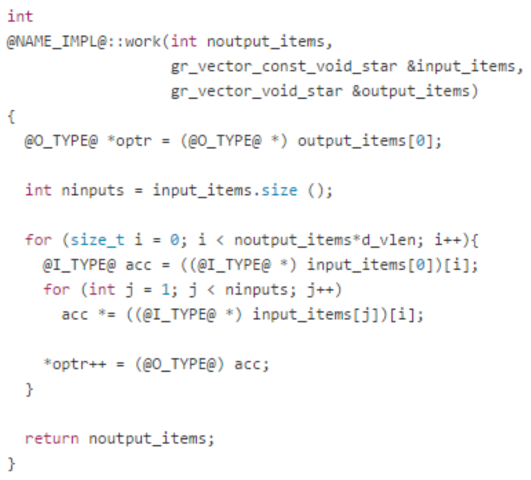
- As a guard interval, it eliminates the intersymbol interference from the previous symbol.
- As a repetition of the end of the symbol, it allows the linear convolution of a frequency-selective multipath channel to be modelled as circular convolution, which in turn may be transformed to the frequency domain using a discrete Fourier transform.

This approach allows for simple frequency-domain processing, such as channel estimation and equalization. For cyclic prefix to be effective (i.e. to serve its aforementioned objectives), the length of the cyclic prefix must be at least equal to the length of the multipath channel.

In GNU-Radio, the cyclic prefixer adds the necessary prefix to each symbol, while also optionally implementing a root raised cosine filter for further reducing ISI. Thus, we needed to implement two functionalities in the signal processing code. Code snippets for the actual signal processing are present in Figure 5. A look at the GNU-Radio source code hosted at github helped us in developing the algorithm of the blocks and test them.

Power Allocator (Water Pouring Algorithm)

The water-pouring algorithm is a technique used in digital communications systems for allocating power among different channels in multi-carrier schemes. It was described by R. C. Gallager in 1968[1] along with the water-pouring theorem which proves its optimality for channels having Additive White Gaussian Noise (AWGN) and intersymbol interference (ISI). For this reason, it is a standard baseline algorithm for various digital communications systems. The intuition that gives the algorithm its name is to think of the communication medium as if it was some



```
int
@NAME_IMPL@::work(int noutput_items,
                  gr_vector_const_void_star &input_items,
                  gr_vector_void_star &output_items)
{
  @O_TYPE@ *optr = (@O_TYPE@ *) output_items[0];

  int ninputs = input_items.size ();

  for (size_t i = 0; i < noutput_items*d_vlen; i++){
    @I_TYPE@ acc = ((@I_TYPE@ *) input_items[0])[i];
    for (int j = 1; j < ninputs; j++)
      acc *= ((@I_TYPE@ *) input_items[j])[i];

    *optr++ = (@O_TYPE@) acc;
  }

  return noutput_items;
}
```

Figure 6. Water pouring

kind of water container with an uneven bottom. Each of the available channels is then a section of the container having its own depth, given by the reciprocal of the frequency-dependent SNR for the channel. To allocate power, imagine pouring water into this container (the amount depends on the desired maximum average transmit power). After the water level settles, the largest amount of water is in the deepest sections of the container. This implies allocating more power to the channels with the most favourable SNR.

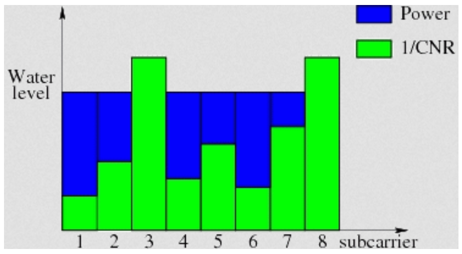


Figure 7. Water pouring algorithm

To actually implement the algorithm, we need to know the SNR/channel frequency response prior to OFDM transmission. This can be done by first performing a channel estimation to determine the SNR of each sub-carrier. Here we implemented this block by simply allocating power to each sub-carrier such that it is inversely proportional to the magnitude of frequency response of the channel, which has already been determined by some other method previously. The block can be used only if the channel response has been estimated by sending pilot symbols at pilot carrier frequencies. Figure 6 shows signal processing code snippets and block diagrams for the power allocator.

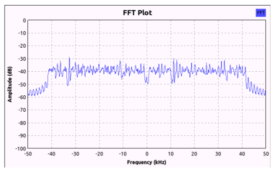


Figure 8. OFDM without power allocation

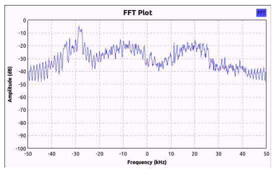


Figure 9. OFDM with power allocation

Advantages of OFDM

OFDM has several advantages over other other wide-band digital modulation techniques. We were able to see some of these through our implementation -

- Reduced ISI - As discussed above, the ”orthogonal” nature of frequency multiplexing, followed by cyclic prefixing effectively eliminate ISI in single-path channels and reduce it considerably in multi-path channels.
- Spectrum Utilisation - Figure 7 describes how OFDM saves spectrum compared to regular frequency division multiplexing, due to overlapping but orthogonal carriers sharing the same spectrum.
- Channel Estimation & Equalization - The cyclic prefixing scheme and DFT allow for fast and easy signal processing, both in hardware and software, thus allowing easy channel estimation and equalization, through known fixed symbols (pilot symbols).
- Other - OFDM can support dynamic packet access. MIMO systems and space–time coding can be realized on OFDM and all the benefits of MIMO systems can be obtained easily. Adaptive modulation and tone/power allocation are also realizable on OFDM.

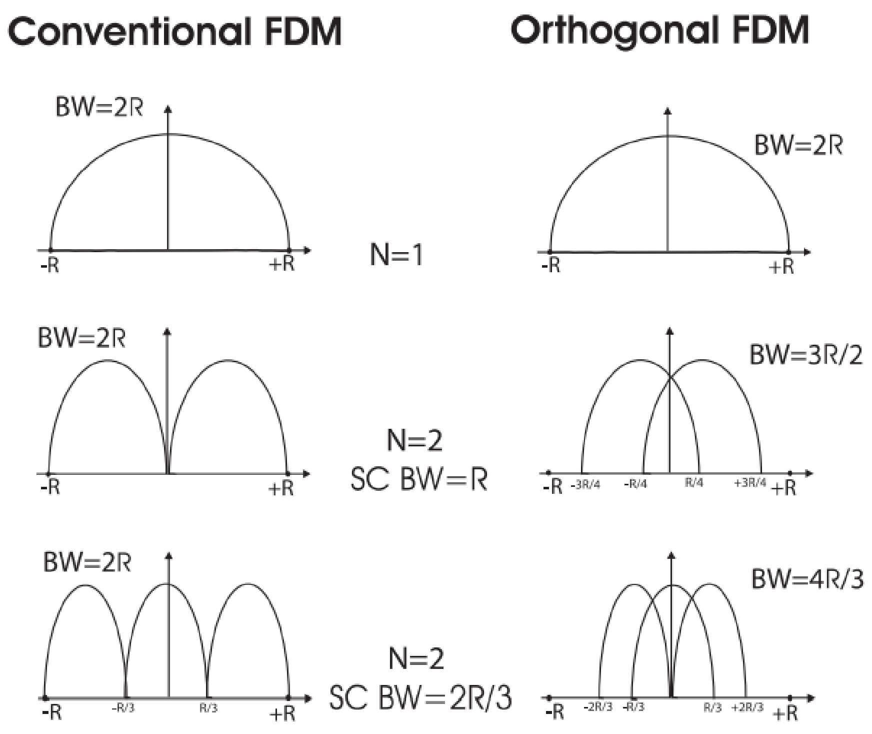


Figure 10. Frame detection

Disadvantages of OFDM

Despite all its advantages, OFDM does have certain disadvantages, which we observed during the procedure.

- Strict Synchronization Requirements - OFDM is highly sensitive to time and frequency synchronization errors, and especially with frequency synchronization errors, everything can go wrong. Indeed, demodulation of an OFDM signal with an offset in the frequency can lead to a high bit error rate. We were able to observe this effect clearly when receiving the signal through an RTL-SDR dongle, when small frequency errors resulted into dropping of most of the packets.

IEEE 802.11 standard

A large variety of digital communication happens using the IEEE 802.11 standard, especially wi-fi. We also studied ways to build an OFDM receiver for this standard. The first part of building an OFDM receiver is to detect the presence of an OFDMD frame. This is done by detecting certain specific bits/properties of the received signal.

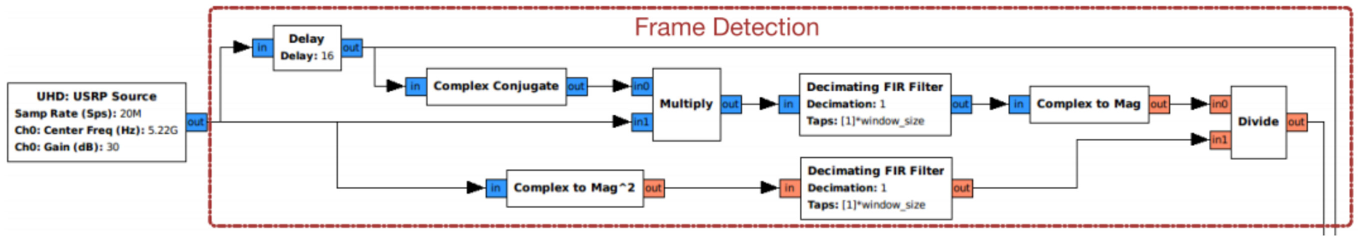


Figure 11

In the IEEE 802.11 standard, the frame begins with a preamble that has very high autocorrelation compared to regular signal, thus giving us a way to detect frames. We found an implementation to detect OFDM frames using this scheme in [8], which we implemented in GNU-Radio, to build an OFDM Frame Detector. (Figure 8)

Acknowledgements

We thank Prof. Shalabh Gupta and the TAs of the lab for their help throughout the project.

Challenges Faced

While information about OFDM was easily accessible on the internet, the actual coding of blocks in GNU-Radio was the hardest part of the project, mainly because of the lack of material available for guidance. Understanding the software stack of GNU-Radio, and how the OFDM related blocks and modules worked was also a challenging task.

Configuring our laptops for using USRP and running codes in itself was a pretty daunting task, with many dependencies, libraries and drivers requirements needing to be resolved.

References

1. *GNU Radio Manual and C++ API Reference 3.7.8*
2. *New Directions in Wireless Communications Research, Chapter-2, OFDM-Principles and Challenges* - N. Marchetti, M. I. Rahman, S. Kumar, R. Prasad
3. *Say, OFDM – You're looking fantastic these days*, GNU Radio Conference, 2013, Boston Martin Braun 2.10.2013
4. *http://github.com/gnuradio/gnuradio-wg-grc* - the source code repository for GNU-Radio
5. *Information Theory and Reliable Communication. Wiley, New York, 1968* - R.C. Gallager, water-pouring algo.
6. *Implementation of Dynamic Resource Allocation for LTE System Using GNU Radio*, Hanke Cheng, 2015-03-24
7. *Wikipedia articles on OFDM, Cyclic Prefixing and Water-Pouring Algorithm*
8. *An IEEE 802.11a/g/p OFDM Receiver for GNU Radio* - B. Bloessl, M. Segata, C. Sommer and F. Dressler