Abstract—the progress in digital communication systems has been mainly due to software algorithms instead of dedicated hardware [21]. GNU Radio is a Software development toolkit to implement software radios – for its purpose it provides signal processing blocks. GNU Radio is free and open source [20]. It can be used for simulation as well as with external RF components for the development of software defined radio. In GNU Radio programs/simulations are basically implemented using Python programming language on higher layer whereas on the lower layer the performance-critical components are implemented using C++.

Primarily GNU Radio is not a simulation tool, it is a Software Defined Radio (SDR). It supports development of signal processing algorithms either by creating new or by using the provided blocks. Thus the actual RF environment is not always needed. In GNU radio the transmitted waveform is defined by the software, similarly the received waveform is also demodulated by it. Software radio is a technique of making a code as close to the antenna as possible [22]. The hardware problems are thus converted into software problems.

Universal software radio peripheral (usrp)

USRPs act as bridge between the “soft” and the “hard” world; they allow computers to work as software radios. Produced by Ettus Research [23] each USRP contains a motherboard containing up to four ADCs, four DACs, a million gate-field programmable gate arrays (FPGA) and a programmable USB 2.0 controller [22].

Each USRP motherboard supports four daughter boards, two for reception and two for transmission. RF front ends are implemented on the daughter boards. A number of daughter boards are available to help utilize different functions and frequency bands – for amateur use daughter boards with low power are available.

I. INTRODUCTION

GNU Radio provides a number of built-in blocks for development of a radio; however for creating new blocks C++ and Python languages are used, both the languages play different roles. Python works for creating flow graphs gluing together different blocks and thus GNU Radio helps in data flow abstraction [24]. Performance critical and Signal-Processing blocks are written in C++. For the purpose of development, template blocks are also available which can be edited according to the need and then added to GNU Radio.

Square Block

For the purpose of development of new blocks in GNU Radio, the default block provided by the developers of GNU radio is a Square block. This block takes a data stream as input and by squaring it gives the output stream. The C++ code working at the back end of this block simply runs a ‘for’ loop which multiplies the input with itself and gives the output [27].

The block serves as the basic example for any newbie to start adding new blocks on GNU Radio. The working of the block can be changed by editing the C++ code working at the back end. A block once added can be modified by changing C++ code, naming specification etc. and other files provided in the root directory and then again running the commands used for adding the block. This would eventually replace the old block with the modified one.

Adding the block in GNU Radio requires downloading and unzipping the folder provided on GNU Radio official website. By opening the terminal of the
operating system we go to the directory of downloaded block, the commands [6] for adding the blocks need to be run.

LTE-advanced eNodeB physical layer

The physical layer of LTE Advanced eNodeB transmits (Downlink) is shown in following diagram:

![LTE eNodeB Physical layer Transmit Diagram]

Cyclic Redundancy Check (CRC) Calculation

In LTE-Advanced Physical layer CRC is attached twice to the data blocks, this mechanism is known as two layer CRC attachment. The medium access (MAC) layer works on transport blocks. For the purpose of forward error correction a CRC sequence is appended to a transport block. This sequence is computed from the bits in the transport block. A 24-bit CRC generator polynomial is used for this purpose. A transport block can be up to hundreds of thousands of bits for high data rate services [27]. Such large blocks are divided into N small segments. These segments are known as Code Block with a maximum length of 6144 bits [28].

Algorithm for CRC calculation in C++

For the purpose of implementation a polynomial is made from the code block bits and divided by CRC generator polynomial using binary polynomial division. For implementing polynomial division in C++ we have used spreadsheet algorithm [28]. We discuss the spreadsheet algorithm using the following example.

Columns header depicts powers of x in the polynomials. Polynomial with a double line border is the numerator polynomial or dividend. The denominator polynomial or divisor is in single border. Under the heading of Division Process we first write the numerator. The denominator is then multiplied by xN-K where N is the order of numerator polynomial and K is the order of denominator polynomial. In this example N=16 and K=10 so we multiply the denominator by x6. The multiplication simply shifts the denominator bits so that the most significant bits of denominator fall under the most significant bits of numerator. The decision to execute at any iteration is done based on the most significant bit of numerator. This bit is copied to the YES/NO column. The most significant bit of the numerator is also copied to the most significant bit of Quotient column at the bottom, in this case at x6 position. We add the denominator to xN-K-i in each iteration using binary arithmetic. If the most significant bit of denominator which put in the YES/NO column is zero, the corresponding entries for the bits of the denominator are made zero by multiplying them by this bit.

Tail Biting Convolutional Coding

Tail biting convolutional coding is done on control channels in the physical layer of LTE Advanced. The following diagram specifies the encoder used for this purpose.

![Convolutional Encoder for LTE Advanced Diagram]

Trellis Encoder Block

The convolutional coder is implemented in GNU Radio using the Trellis Encoder block. This is a generic block and any convolutional encoder can be implemented over it by developing an FSM file. Convolutional encoders being a type of Finite State Machines have a definite number of possible memory states and based on memory states have a definite set
Turbo encoder used in LTE Advanced physical layer is a Parallel concatenated convolutional code as shown in following figure. For its implementation we have developed an FSM file which specifies the two convolutional encoders and the interleaver is separately developed as a block. The components are then joined together to work as turbo encoder.

**FSM File**

The following FSM file specifies the convolutional encoder of LTE Advanced. The first number of first row '2' represents the number of possible inputs of the encoder which are 0 and 1. The second number '64' represents the number of possible memory states which is 26, where 6 are number of memory units in the encoder. Similarly the third digit '8' represents the number of possible outputs i.e. 000,001,010,011,100,101,110,111. The first set of two column numbers represents the next states and the second set represents the next outputs. In the first set the first row (0, 32) represents the next state when the initial state is zero; when 0 is given at the input of the encoder the next state will be again zero and if 1 is given at input the next state will be binary 32, so the left column represents the 0 input and the right column represents 1 input. Similarly the second row represents the current state 1 and so on. The second set represents next output in similar fashion.

**Testing in Python**

The following diagram shows the procedure to verify the output of the block in numerical form, the commands run in Python and the output.

![Figure3 Testing in Python](image)

A random input vector [1, 0, 1, 1, 1, and 0] was taken for the purpose of testing the fsm file and passed through the trellis encoder to be encoded using the fsm file. The output (7, 3, 9, 2, 2, and 6) matched exactly with the value produced through manual dry run.

II. TURBO CODING

**Turbo Encoder**

Turbo encoder used in LTE Advanced physical layer is a Parallel concatenated convolutional code as shown in following figure. For its implementation we have developed an FSM file which specifies the two convolutional encoders and the interleaver is separately developed as a block. The components are then joined together to work as turbo encoder.

**Interleaver**

The interleaver changes the order of the bits in which they are coming and ‘re-distribute’ them, the algorithm used is:

\[ I(i) = (f_1i + f_2i^2) \mod K \]

Where K is the code block size and f1 and f2 depend on K i.e. the code block size used [26]. We have used the code block size of 64. Therefore f1 = 7 and f2 = 16.

**Testing in Python**

III. OFDM
CONCLUSION

In this project, the implementation of the physical layer of LTE-Advanced was done from eNodeB side using the GNU Radio software tool. This implementation includes all the signal processing blocks that are employed in the 3GPP specifications for the LTE-Advanced physical layer. Uplink and Downlink for eNodeB were implemented in the project. LTE-Advanced features two different transmission schemes for transport and control channels. Transport channels, which contain higher traffic for transmission as compared to control channels, employ Turbo Coding Schemes in channel coding block while control channels use Convolutional coder. Both channels were implemented in this project. This project focuses on the implementation and subsequent transmission of bits. Simulations made in GNU Radio successfully transmitted and received the bits. Later on, USRPs were used to ensure the real-time performance of the designed system and it was a successful test.

BER error rates for different isolated channel coding schemes and different channel conditions were also analyzed. The effect of channel coding schemes on BER was imperative. OFDM block was also modified with different modulation schemes and different cyclic prefix lengths. GNU Radio is a free software toolkit, which is made for designing software defined radio systems. All signal processing blocks were made in GNU Radio using C++ and Python programming languages. GNU Radio provided a very friendly environment in this project.

REFERENCES