LTE Scanner

Submitted By:

Ola Fathy

Hager Mohammed

Aya Khaled

Mai Mohammed

Seif Eldeen Khaled

Mostafa Mohammed

Mostafa Akhlak

Ahmed Adel

Mahmoud Ramzy

communication

Graduation Project Report

Dr. Diaa Eldin Khalil

Dr. Hatem Yousry

27/8/2020

DECLARATION

We hereby certify this material, which we now submit for assessment on the programme of study leading to the award of Bachelor of Science in *(communication systems and circuits)* is entirely our own work, that we have exercised reasonable care to ensure that the work is original, and does not to the best of our knowledge breach any law of copyright, and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our work.

Signed: by all students

Ola Fathy Abdelazim

Aya Khaled

Hager Mohammed

Mai Mohammed

Seif Eldeen Khaled

Moustafa Mohamed

Mostafa Akhlak

Mahmoud Ramzy

Ahmed Adel

Date: 27/8/2020.

ABSTRACT

The LTE Scanner is a device used to measure the coverage of an LTE cell by decoding the broadcast Frame sent continuously by the EnodeB (LTE base station) and processing its data, the data decoding and processing is done at different places to get useful information that will make us able to evaluate the signal quality.

This project shows how the Matlab LTE Toolbox & FPGA can be used to make a simulation of LTE downlink signal. And how it can be used to extract information form it.
At last, we work on hardware devices based on SDR and use their channels to receive the same LTE downlink signal, then compare the results.

ACKNOWLEDGMENT

We have taken efforts in this project and it has been a great opportunity to gain a lot of experience throughout this project.

We hereby acknowledge the contribution of all my team members, our respected supervisors: Dr. Diaa, and Dr. Hatem that truly guided us throughout this year, and all the communication systems and circuits professors that helped us throughout our years in the faculty.

Table of Contents

1 Introduction 12
1.1 Introduction 12
1.2 Technical Overview 12
2 LTE Physical Layer 13
2.1 LTE Main features 13
2.2 Time and Frequency domain resources 13
2.2.1 Frequency domain resources 14
2.2.2 Time domain resources 16
2.2.3 Physical Resource Block and Resource element 19
2.2.4 Downlink Resource allocation 20
3 LTE Scanner 21
4 SDR overview 22
4.1 SDR "software defined radio": 22
4.2 C700 WIRELESS Modular Test & Development Platform: 23
4.2.1 Features: 23
4.2.2 SYSTEM ARCHITECTURE AND PORTS: 23
4.2.3 SPECIFICATIONS: 24
4.3 USRP N210: 27
4.3.1 N210 product overview: 27
4.3.2 N210 Features: 27
4.3.3 N210 Block diagram: 28
4.3.4 N210 Ports: 28
4.4 RTL-SDR: 29
29
Specifications 30
4.5 Evrytania software: 31
4.5.1 Evrytania LTE Tools: 31
5 FPGA 32
5.1 Introduction 32
5.2 Programming Technologies 32
5.2.1 SRAM-Based FPGAs 32
5.2.2 Antifuse-Based FPGAs 33
5.2.3 Technologies Comparisons 34
5.2.4 FINE-, MEDIUM-, AND COARSE-GRAINED ARCHITECTURES 35
5.3 Logic Blocks 36
5.3.1 MUX-based 36
5.4 CLBs VERSUS LABs VERSUS SLICES 40
5.4.1 Logic Cells/Logic Elements 40
5.4.2 Slicing and Dicing 41
5.4.3 CLBs and LABs 42
5.4.4 Distributed RAMs and Shift Registers 42
5.5 Embedded elements inside the FPGA 43
5.5.1 Embedded RAMS 43
5.5.2 Embedded MULTIPLIERS, ADDERS, ETC. 44
5.5.3 Embedded Processor Cores 45
5.5.4 General Purpose I/O 48
5.5.5 GIGABIT Transceivers 50
5.6 5.6 INTELLECTUAL PROPERTY (IP) 51
5.6.1 Handcrafted IP 52
5.6.2 IP Core Generator 53
5.7 FPGA Design Flow 54
5.7.1 Overview 55
5.7.2 Design Entry 55
5.7.3 Synthesis 56
5.7.4 Implementation 56
5.7.5 Device Programming 59
5.7.6 Design Verification 59
6 Project idea and planning 60
6.1 Project description 60

6.2 Project objective 60
6.3 Project planning 60
7 MATLab LTE Scanner Simulation 61
7.1 Introduction 61
7.2 Generating LTE signals 61
7.2.1 Standard-compliant waveforms 61
7.2.2 Custom Signals 63
7.3 Performing cell search and MIB recovery 63
7.3.1 Simulation results 64
7.4 Conclusion 66
8 C700 LTE Scanner 67
8.1 Main target: 67
8.2 C700 Operation: 67
8.3 Generated/Captured Waveform: 69
8.4 Send waveform to VSM: 70
8.5 VSD acquires the samples: 70
8.6 Setup of C700: 71
8.7 Results: 72
9 Evryrania-RTL LTE scanner 73
9.1 Required libraries : 73
9.1.1 Boost library : 73
9.1.2 ITPP library : 73
9.1.3 rtlsdr library : 74
9.1.4 fftw library : 75
9.2 Evrytania Download source code and build instructions : 75
9.2.1 Obtaining the code : 75
9.2.2 Compile : 75
9.2.3 Install : 76
9.3 Setup: 76
9.4 Running Cellsearch tool: 76
9.5 Results: 76
10 FREQUENCY SYNCHRONIZATION 78
10.1 Carrier frequency offset (CFO) 78
The baseband transmit signal is converted up to the passband by a carrier modulation and then converted down to the baseband at receiver by using a local carrier signal of the same carrier frequency at the transmitter. Well, that is not always the case, there may be a mismatch between transmitter and receiver local oscillators due to non-idealities of the manufacturing process, also a Doppler shift may be exist due to the relative motion between transmitter and receiver. The existence of any of the two previous main causes of CFO will result in inter-carrier interference (ICI) between different subcarriers which result in loss of orthogonality. 78
10.2 the cause for Carrier Frequency Offset Error 78
10.2.1 Mismatch between transmitter and receiver 78
10.2.2 The Doppler effect (or the Doppler shift) 79
10.3 Effects of CFO 80
10.3.1 Integer Frequency Offset (IFO) 80
10.3.2 Fractional Frequency Offset (FFO) 80
10.4 Estimation Techniques for CFO 81
10.4.1 CP based Estimation Method 81
10.4.2 Symbol based Estimation Method 82
10.4.3 Pilot Tone based Estimation Method 82
10.5 Simulation Results 82
11 Time synchronization and Cell ID Detection 85
11.1 Cell search 85
11.1.1 step I- Primary synchronization signals 86
11.1.2 Step II- Secondary synchronization signal (SSS) 90
11.1.3 Timing synchronization and Cell ID 93
11.1.4 Differences between FDD & TDD 94
11.2 Master Information Block 95
11.2.1 Master Information Block Importance 96
11.2.2 Master information Block positioning 96

11.3 Digital Team contribution 98
11.3.1 Cross Correlation 98
11.3.2 Cross correlator code for real inputs 101
11.3.3 Cross correlator code for complex inputs 105
11.3.4 Max Finder Circuit 110
11.3.5 Top Module 111
11.3.6 Conclusion 116
12 OFDM 118
12.1 Introduction 118
12.1.1 Why OFDM? 118
12.1.2 Intersymbol Interference (ISI) 119
12.2 Cyclic Prefix Insertion 122
12.3 OFDM (Orthogonal Frequency Division Multiplexing) 124
12.4 Orthogonality 125
12.5 OFDM Block Diagram 126
12.5.1 On the transmitter side 126
12.5.2 On the receiving side 126
12.6 LTE in downlink and uplink 126
12.6.1 LTE OFDMA in the downlink 126
Downlink carriers and resource blocks 127
12.6.2 LTE SC-FDMA in the uplink 127
12.7 OFDM Advantages & Disadvantages 129
12.7.1 OFDM Advantages 129
12.7.2 OFDM Disadvantages 129
12.8 Brief of the code steps 130
12.9 Simulation results 130
12.10 Digital part 131
12.10.1 FFT in OFDM system 131
12.10.2 FFT algorithms 132
12.10.3 Comparison between different FFT algorithms 137
12.10.4 HDL implementation of radix-2 FFT 138
13 Channel estimation 152
13.1 Introduction 152
13.2 Channel Estimation Overview 152
13.3 Process of channel estimation Block Diagram 153
13.4 Pilot Estimation Techniques 153
13.4.1 Least Square 153
13.5 Pilot Average Subsystem 155
13.6 Interpolation 156
13.6.1 Linear interpolation method: 156
13.7 The Digital Perspective: 157
13.7.1 The Rx grid (input): 158
13.7.2 The limitations in Verilog HDL: 159
13.7.3 Real values: 159
13.7.4 Arithmetic operations of Fixed point: 160
13.8 Implementation of Estimation: 162
13.8.1 Extract the Pilot from Rx Grid: 162
13.8.2 Averaging and Interpolation: 164
13.9 Equalization: 165
13.10 Simulation results: 165
13.10.1 Communication results: 165
13.10.2 Digital results: 166
14 conclusion And Future work 171
14.1 communication team 171
Frequency synchronizations: 171
Cell Search and Time Synchronization: 171
OFDM: 171
Channel Estimation: 172
14.2 SDR team 172
14.1 Digital team 172
References 173

List of figures

Figure 2-1 time & freq. domain resources demonstration 14
Figure 2-2 frequency domain resources 14
Figure 2-3 Subcarriers types 15
Figure 2-4 data subcarriers for different channel bandwidths 16
Figure 2-5 Type 1 frame for FDD 17
Figure 2-6 Differences between normal and extended CP 17
Figure 2-7 Frame type 2 for TDD 18
Figure 2-8 Resource Block and Resource Element 19
Figure 2-9 transmitted resource grid for 1.4 MHz channel 20
Figure 4-1 SDR Block diagram 22
Figure 4-2 C700 Architecture 23
Figure 4-3 C700-BU: system base unit 24
Figure 4-4 c700-sync: synchronization module 25
Figure 4-5 controller/host interface 25
Figure 4-6 system local oscillator modules 26
Figure 4-7 C700-vsm1 and vsd 26
Figure 4-8 N210 product overview 27
Figure 4-9 N210 Block digram 28
Figure 4-10 N210 interfaces 28
Figure 4-11 RTL-SDR 29
Figure 5-1 comparison of Programming technologies 35
Figure 5-2 Underlying FPGA Fabric 36
Figure 5-3 MUX-based Logic Block 37
Figure 5-4 A transmission gate based LUT (programming chain omitted for purposes of clarity). 38
Figure 5-5 Configuration cells linked in a chain 39
Figure 5-6 A multifaced LUT 40
Figure 5-7 A simplified view of Xilinx LC 41
Figure 5-8 A slice containing two logic cells 41
Figure 5-9 A CLB containing four slices (the number of slices depends on the FPGA family) 42
Figure 5-10 embedded RAM Blocks. 44
Figure 5-11 columns of embedded multipliers and RAM blocks 44
Figure 5-12 The functions forming a MAC. 45
Figure 5-13 chip with embedded core outside of the main fabric 47
Figure 5-14 chips with embedded cores inside the main fabric. 47
Figure 5-15 chip showing general-purpose I/O banks. 49
Figure 5-16 Using high-speed transceivers to communicate between devices 50
Figure 5-17 Alternative potential IP acquisition points 53
Figure 5-18 IP block/core generator 54
Figure 5-19 FPGA Design Flow. 55
Figure 5-20 Synthesis 56
Figure 5-21 Translate process 57
Figure 5-22 Mapping process 58
Figure 5-23 Placement and Routing 58
Figure 7-1 real waveform for reference channel R2 generated by MATlab 61
Figure 7-2 spectrum for the refrence channel R2 generated by MATLab 62
Figure 7-3 Cell search procedure flow 63
Figure 7-4 generated signal parameters 64
Figure 7-5 Simulation results 64
Figure 7-6 Timing offset position 65
Figure 7-7 Start of frame position in case of 0 delay for 21 subframes 65
Figure 7-8delayed signal 65
Figure 7-9 timing offset for delayed signal 66
Figure 8-1Generated waveform is processed in MATLAB to obtain MIB bits 69
Figure 8-2 C700 Setup 71
Figure 8-3Matlab results after decoding 72
Figure 9-1RTL-SDR Setup 76
Figure 9-2 Evrytania results 77

Figure 10-1The effect of CFO in frequency domain, Inter-carrier interference (ICI) 78
Figure 10-2 The Doppler effect of moving source 79
Figure 10-3 Effect of integer CFO on the received signal 80
Figure 10-4 Constellation of received symbols with different CFO(FFO) 83
Figure 10-5 SNR degradation due to CFO 83
Figure 10-6 Bit Error Rate vs Eb/N0 (dB) with/without estimation 84
Figure 10-7 Bit Error Rate vs Eb/N0 (dB) with estimation methods 84
Figure 11-1 PSS in case of different cyclic prefix lengths for frame structure 1 (FDD) 86
Figure 11-2 PSS in frequency domain 86
Figure 11-3 PSS generation by MATLab 88
Figure 11-4 Correlation between different PSS sequences 88
Figure 11-5 Auto correlation for pss sequences 89
Figure 11-6 SSS signal mapping in subframe 0 90
Figure 11-7 SSS Sequence for NID(1)=120 , NID(2)=1 IN Subframe 5 92
Figure 11-8 constellation diagram for SSS 92
Figure 11-9 cross and auto correlation for SSS sequences 92
Figure 11-10 PSS detection using correlator 93
Figure 11-11 primary/secondary synchronization signals for LTE TDD 94
Figure 11-12 Differences between all possible transmission schemes 95
Figure 11-13 MIB periodicity and repetition 96
Figure 11-14 Position of the PBCH,PSS and SSS for both frame struc 97
Figure 11-15 Cross correlation Formula 99
Figure 11-16 Cross correlation calculation using graphical method. 100
Figure 11-17 Simulation Results of behavioural code 102
Figure 11-18 Simulation Results of Behavioural code with 62 input value 102
Figure 11-19 Simulation Results of structural code 103
Figure 11-20 programming file generated and ready to be put on FPGA 104
Figure 11-21 schematic of the correlator 104
Figure 11-22 part of the Technology schematic of the correlator 105
Figure 11-23 simulation results of behavioural complex Correlator 106
Figure 11-24 simulation results of behavioural complex Correlator with 62 input values 107
Figure 11-25 simulation results of structural complex Correlator 107
Figure 11-26 complex correlator programming file generated and ready to be put on FPGA 108
Figure 11-27 RTL schematic of complex correlator. 108
Figure 11-28 RTL schematic of complex multiplier. 109
Figure 11-29 Technology schematic of complex correlator. 109
Figure 11-30 simulation results of real input max finder. 110
Figure 11-31 simulation results of complex input max finder 111
Figure 11-32 simulation results of real valued input top module. 112
Figure 11-33 Top module Programming file generated and ready to be put on FPGA 112
Figure 11-34 RTL Schematic of real valued input top module 113
Figure 11-35 RTL schematic of Max finder 113
Figure 11-36 Technology schematic of real valued input top module. 114
Figure 11-37 simulation results of complex valued input top module 114
Figure 11-38 Complex Top module Programming file generated and ready to be put on FPGA. 115
Figure 11-39 RTL Schematic of complex valued input top module 115
Figure 11-40 Technology Schematic of complex valued input top module 116
Figure 12-1Multipath propagation from the Transmitter and the Receiver 119
Figure 12-2Power Delay Profile 119
Figure 12-3Multipath-Induced Time Delays Result in ISI 120
Figure 12-4Longer Delay Spreads Result in Frequency Selective Fading 120
Figure 12-5 delay spread 121
Figure 12-6 Symbols with time delay spread 121
Figure 12-7 Symbols with time delay spread 122
Figure 12-8 Symbol Rate 122
Figure 12-9 Cyclic Prefix Insertion 122
Figure 12-10 Normal and Extended Cyclic Prefix 123
Figure 12-11 multi-carrier transmission 124
Figure 12-12 time and frequency domains 125
Figure 12-13 Orthogonally 125
Figure 12-14 Block diagram of OFDM system 126

Figure 12-15 downlink in OFDM system 127
Figure 12-16 uplink in OFDM system 127
Figure 12-17 downlink and uplink in OFDM system 128
Figure 12-18 simulation results 131
Figure 12-19 Length-4, DIT radix-2 FFT 133
Figure 12-20 Radix-4 Burst i/o 133
Figure 12-21 Radix-4 FFT butterfly 135
Figure 12-22 Radix-4, Burst I/O implementation 135
Figure 12-23 Split-radix FFT 136
Figure 12-24 Resource VS Throughput for Architecture Options 138
Figure 12-25 Radix-2 latency 138
Figure 12-26 Radix-4 latency 138
Figure 12-27 page1 in the GUI 139
Figure 12-28 page2 in the GUI 140
Figure 12-29 Page 3 of the GUI 141
Figure 12-30 Resource Estimates tab in the GUI 143
Figure 12-31 Latency tab in the GUI 144
Figure 12-32 Technology schematic 144
Figure 12-33 RTL schematic 145
Figure 12-34 Simulation(1) 145
Figure 12-35 Simulation(2) 146
Figure 12-36 Simulation(3) 146
Figure 12-37 Device Utilization Summary 148
Figure 13-1 Reference signal allocation 152
Figure 13-2 Averaging methods 155
Figure 13-3 grid after doing time average 156
Figure 13-4 Linear interpolation example 157
Figure 13-5 Rx Grid 158
Figure 13-6 (32 bit) single floating point format 160
Figure 13-7 Q2.8 fixed point format 160
Figure 13-8 Locations of Reference signal 163
Figure 13-9 Time averaging and Frequency averaging 164
Figure 13-10 Comparison between LS and MMSE vs MSE 165
Figure 13-11 Comparison between LS and MMSE vs BER 166
Figure 13-12 The generated random binary data from matlab 166
Figure 13-13 Multiplier Block calculation vs matlab calculation over 30 trials 167
Figure 13-14 Divider block calculation vs matlab calculation over 30 trials 167
Figure 13-15 Rtl Schematic of LSE Estimation block 168
Figure 13-16 Summary of timing report 168
Figure 13-17 Detailed delay in gates in timing report 169
Figure 13-18 the summary of utilization report 169
Figure 13-19 the simulation of the recovered data in Equalization block 170

List of tables

Table 2-1 main features of LTE systems 13
Table 2-2 LTE Downlink physical layer parameters 16
Table 2-3 Uplink-downlink configurations for TDD frame 18
Table 2-4 Available RBs for different Channel BWs 20
Table 4-1 C700 Ports 24
Table 4-2 RTL-SDR features and specs 30
Table 4-3 RTL-SDR types of tuners 30
Table 7-1 Parameters of the reference channels used to generate LTE signals 62
Table 10-1 Doppler frequency and normalized CFO: an example 79
Table 11-1 Root indices for the PSS 87
Table 11-2 mapping between NID (1) and indices m0 & m1 91
Table 12-1 LTE DOWNLINK CARRIERS AND RESOURCE BLOCKS 127
Table 12-2 Number of real multiplications to compute a length-N complex DFT 137
Table 12-3 Number of real additions to compute a length-N complex DFT 137

1 INTRODUCTION

1.1 INTRODUCTION

The network scanner is a critical test equipment for mobile communication network planning and evaluation. It receives and decodes broadcast base-stations signaling to evaluate signal quality and network information. The speed and portability of the network scanner are very critical performance metrics.

The basic idea of the LTEscanner is to receive the broadcast channel message and process it to extract important information like:

1-System Bandwidth.

2- System Frame Number (SFN).

3- cell identity.

4- duplexing techniques (FDD/TDD)

To implement the scanner, we need a receiver and an FPGA to do the signal processing, one hardware with receiver functionality is an SDR (software defined Radio), we will use it as our receiver.

1.2 TECHNICAL OVERVIEW

The scanner performs cell search procedures to obtain system information. This involves acquiring the primary synchronization signal (PSS) and the secondary synchronization signal (SSS) transmitted by the eNodeB, finding out the cell identity and decoding the Master Information Block (MIB) to get the signal information from them. All of these steps will be described in details in the following book chapters.

2 LTE PHYSICAL LAYER

2.1 LTE MAIN FEATURES

Band	GSM & UMTS Bands, 2600 MHz, ...…
Duplex Techniques	FDD / TDD
Multiple Access	OFDMA / TDMA (Downlink) SC-FDMA / TDMA (Uplink)
Modulation	Adaptive Modulation BPSK – 64 QAM
Carrier BW	1.4MHz – 20MHz flexible but not adaptive
Data rate	Up to 100Mbps (downlink) / 50Mbps (uplink)

Table 2-1 main features of LTE systems

2.2 TIME AND FREQUENCY DOMAIN RESOURCES

For the downlink in LTE systems orthogonal frequency division multiplex (OFDM) was selected as the basis for the PHY layer while SC-FDMA was selected for uplink schemes.
OFDMA is an extension of the OFDM transmission scheme by allowing multiple users. That is allowing for simultaneous frequency separated transmissions to / from multiple mobile terminals.
In OFDM the user data is transmitted in parallel across multiple orthogonal narrowband subcarriers. Each subcarrier only transports a part of the whole transmission.
The orthogonal subcarriers are generated with IFFT (Inverse Fast Fourier Transform) processing, the number of the subcarriers depends on the available bandwidth. In LTE they range from less than one hundred to more than one thousand.

The spacing between subcarriers is fixed in LTE and equivalent to 15 KHz in the frequency domain.
There are two factors that constrain the selection of the OFDM subcarrier spacing:

- The OFDM subcarrier should be as small as possible (Tb as large as possible)
- A too small subcarrier spacing increases the sensitivity of the OFDM transmission to doppler shift.

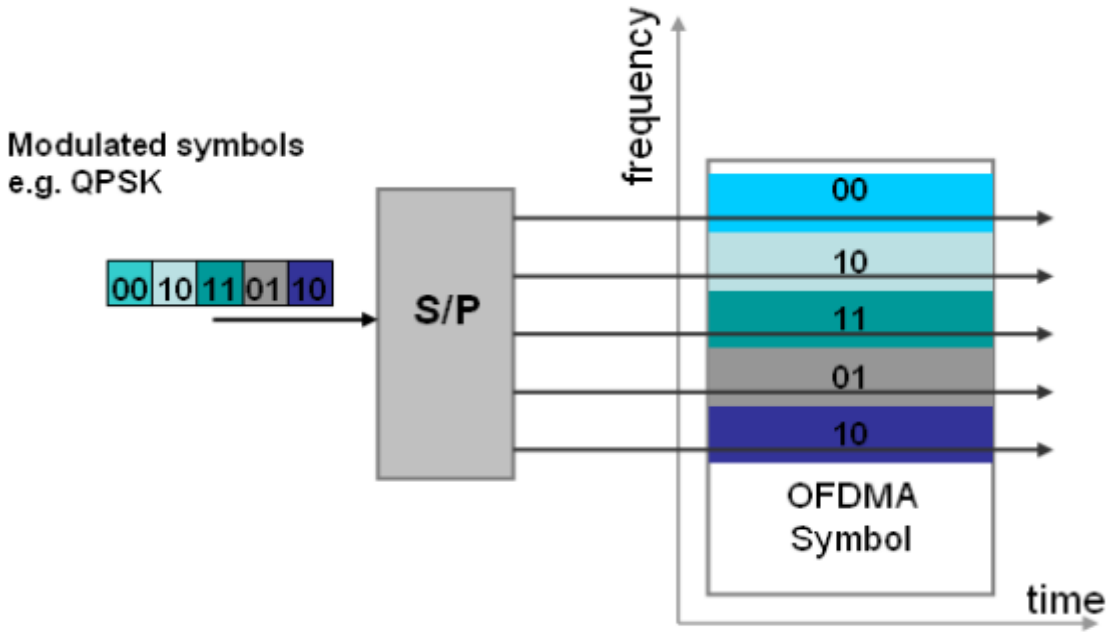


Figure 2-1 time & freq. domain resources demonstration

2.2.1 Frequency domain resources

Sub-Carrier spacing = 15 KHz
Sub-channel = 12 subcarrier
Sub-chanel = 180 KHz

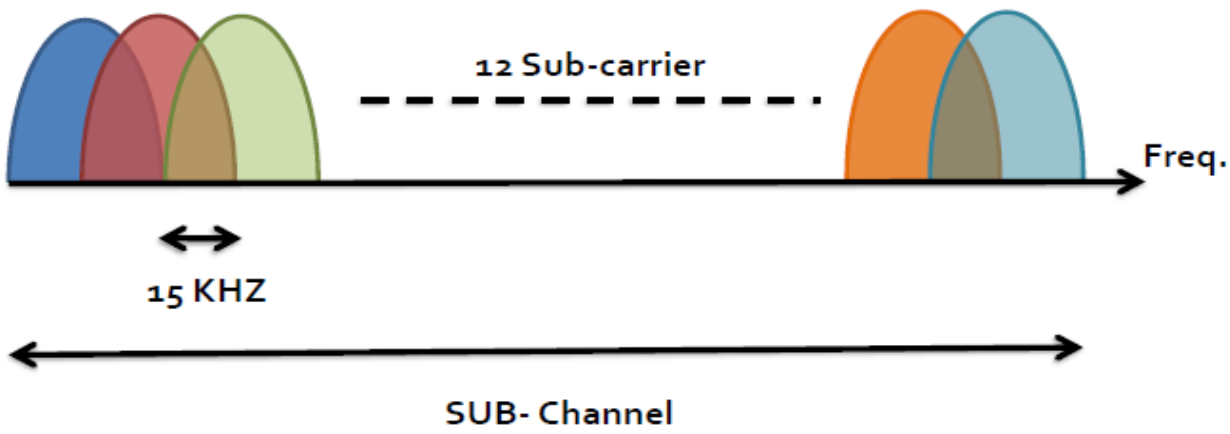


Figure 2-2 frequency domain resources

Subcarriers types

There are different types of subcarriers:

- **Data subcarriers:** Represent the majority of the subcarriers. They carry the modulated user data signals. The data rate of each subcarrier depends on the symbol rate and the modulation scheme used.

- **Null subcarriers:** As the name indicates, nothing is transmitted in these subcarriers. There are two types:

  a) **Guard subcarriers:** they are located at the bottom and the top of the channel. Their function is to limit the amount of the interference caused by the channel and also to limit the adjacent channel interference from neighboring channels. The more guard subcarriers the less the interference but this also reduces the data throughput of the channel.

  b) **DC-subcarrier or null subcarrier:** It is the center subcarrier of the downlink band (0 Hz offset from the channel's center frequency). This subcarrier is not used because it may suffer from high interference due to local oscillator leakage.

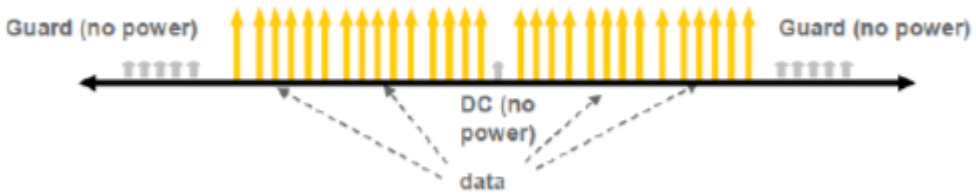


Figure 2-3 Subcarriers types

Channel bandwidth

Downlink bandwidth ranges from 1.4 MHz to 20 MHz and take one of the following values:
- 1.4 MHz
- 3.0 MHz
- 5.0 MHz
- 10.0 MHz
- 15 MHz
- 20 MHz

Data subcarriers

The number of data subcarriers varies with the bandwidth of the channel as shown in the following figure

- 72 for 1.4 MHz to 1200 for 20 MHz

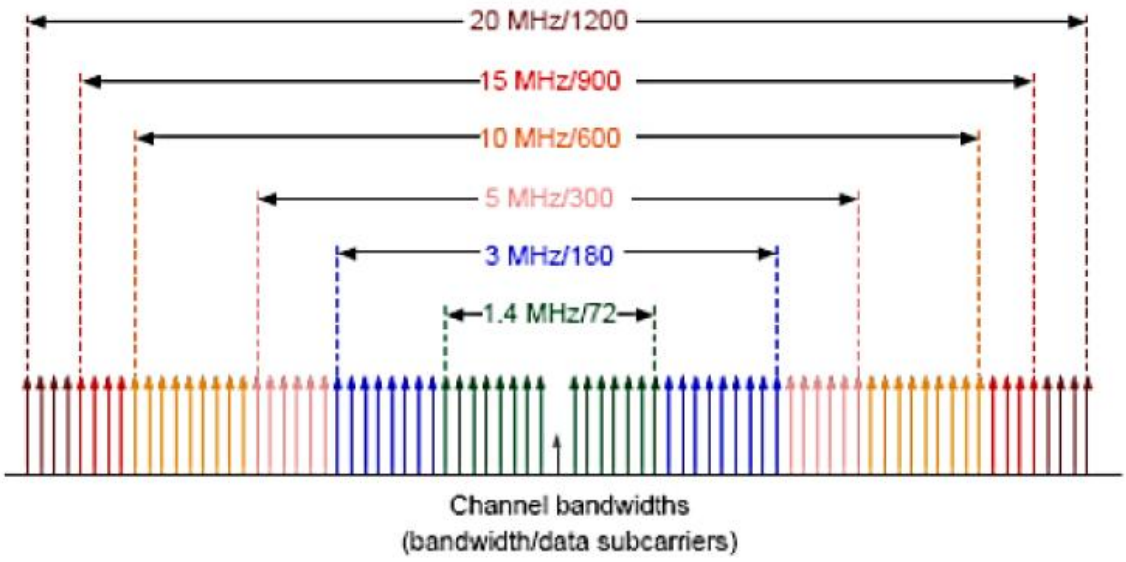


Figure 2-4 data subcarriers for different channel bandwidths

Carrier Bandwidth (MHZ)	1.4	3.0	5.0	10	15	20
No. of sub-channels	6	15	25	50	75	100
No. of subcarriers	72	180	300	600	900	1200
FFT Size	128	256	512	1024	1536	2048
Sampling rate (MHz)	1.92	3.84	7.68	15.36	23.04	30.72

Table 2-2 LTE Downlink physical layer parameters

2.2.2 Time domain resources

Downlink and uplink transmissions are organized into radio frames with duration = 10 ms and There are two types of frames defined for LTE:

- Type 1 frame (for FDD)
- Type 2 frame (for TDD)

Type 1 frame

LTE type 1 Frame (FDD) is common to both uplink and downlink. For FDD, 10 subframes are available for downlink transmission and 10 subframes are available for uplink transmissions in each 10 ms interval. Uplink and downlink transmissions are separated in the frequency domain. Components and durations are illustrated in figures

- Frame length: 10 ms.
- 1 Frame= 20 slots each of 0.5 ms each.
- 1 slot = 7 symbols in the case of normal CP or 6 symbols in case of extended CP
- 1 Frame = 10 subframes of 1 ms each

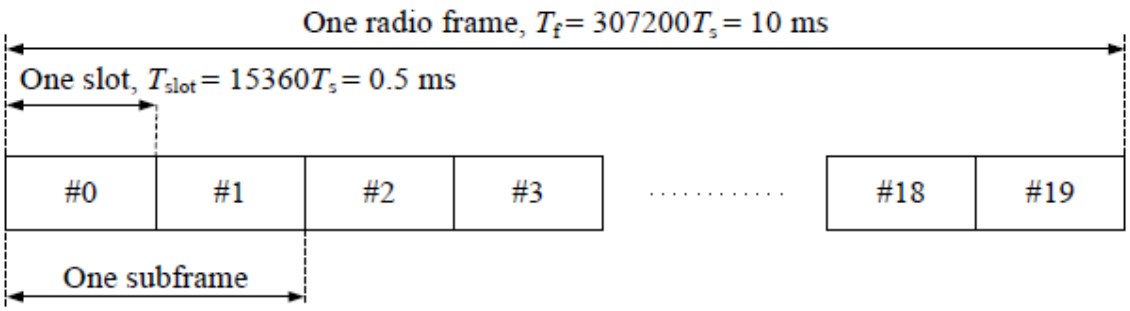


Figure 2-5 Type 1 frame for FDD

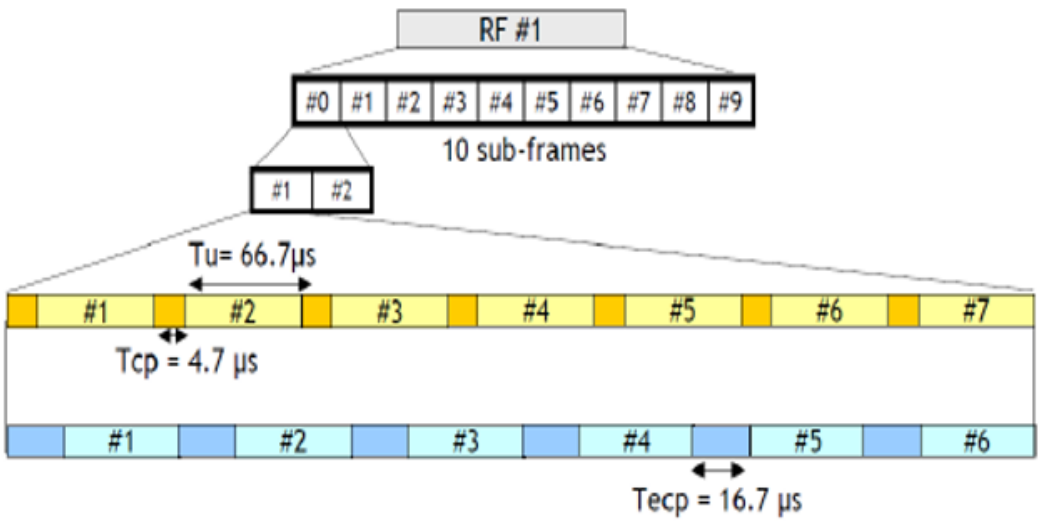


Figure 2-6 Differences between normal and extended CP

The total symbol duration (Ts) includes the cyclic prefix duration. The extended cyclic prefix duration is equal for all the symbols in the slot (16.67 µs). Normal cyclic prefix duration is (4.7 µs)

The reason why there is fewer symbols per slot in the case where extended CP is used, is that the total symbol duration (with extended CP) is longer and this leaves room for fewer symbols within the 0.5 ms slot.

Type 2 frame

Frame structure type 2 is applicable to TDD only. The uplink-downlink configuration in a cell may vary between frames and controls in which subframes uplink or downlink transmissions may take place in the current frame.

The supported uplink-downlink configurations are listed in Table 2.3 where, for each subframe in a radio frame, "D" denotes a downlink subframe reserved or downlink transmissions, "U" denotes an uplink subframe reserved for uplink transmissions and "S" denotes a special subframe with the three fields DwPTS, GP and UpPTS.

Special subframes are used for switching from downlink to uplink and contain three sections: DWPTS, GP, and UpPTS.
DwPTS is the “Downlink Pilot Time Slot”. UpPTS is the “Uplink Pilot Time Slot” and GP is a “guard period” between DwPTS and UpPTS.

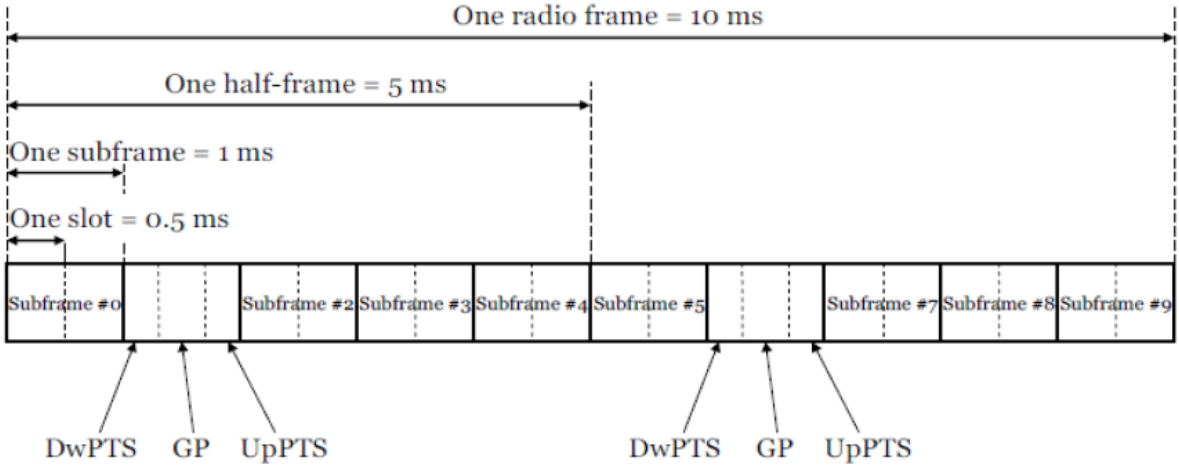


Figure 2-7 Frame type 2 for TDD

Uplink-downlink configuration with both 5 ms and 10 ms downlink-to-uplink switch point periodicity are supported.

- In case of 5 ms downlink-to-uplink switch-point periodicity, the special subframe exists in both half-frames.

- In case of 10 ms downlink-to-uplink switch-point periodicity, the special subframe exists in the first half-frame only.

Uplink-downlink configuration	Downlink-to-Uplink Switch-point periodicity	Subframe number									
		0	1	2	3	4	5	6	7	8	9
0	5 ms	D	S	U	U	U	D	S	U	U	U
1	5 ms	D	S	U	U	D	D	S	U	U	D
2	5 ms	D	S	U	D	D	D	S	U	D	D
3	10 ms	D	S	U	U	U	D	D	D	D	D
4	10 ms	D	S	U	U	D	D	D	D	D	D
5	10 ms	D	S	U	D	D	D	D	D	D	D
6	5 ms	D	S	U	U	U	D	S	U	U	D

Table 2-3 Uplink-downlink configurations for TDD frame

2.2.3 Physical Resource Block and Resource element

A Physical Resource Block (PRB) or Resource Block (RB) is the physical resource used for transmission. Capacity allocation in LTE is based on physical Resource Blocks.
A PRB is composed of 12 subcarriers (1 sub-channel) in the frequency domain per 1 slot period (0.5 ms) in time domain. Since each subcarrier occupies 15 KHz, a PRB occupies 180 KHz (12 x 15 KHz) in the frequency domain.

A resource element (RE) is the theoretical minimum capacity allocation unit. It is formed by 1 subcarrier per 1 symbol.
The following figure represents the resource element concept and physical resource block concept in the case of normal CP being used (7 symbols per slot). There are 84(7 x 12) resource elements per resource block.
In case of extended CP being used each slot contains 6 symbols and a PRB is composed of 72 (6 x 12) resource elements.

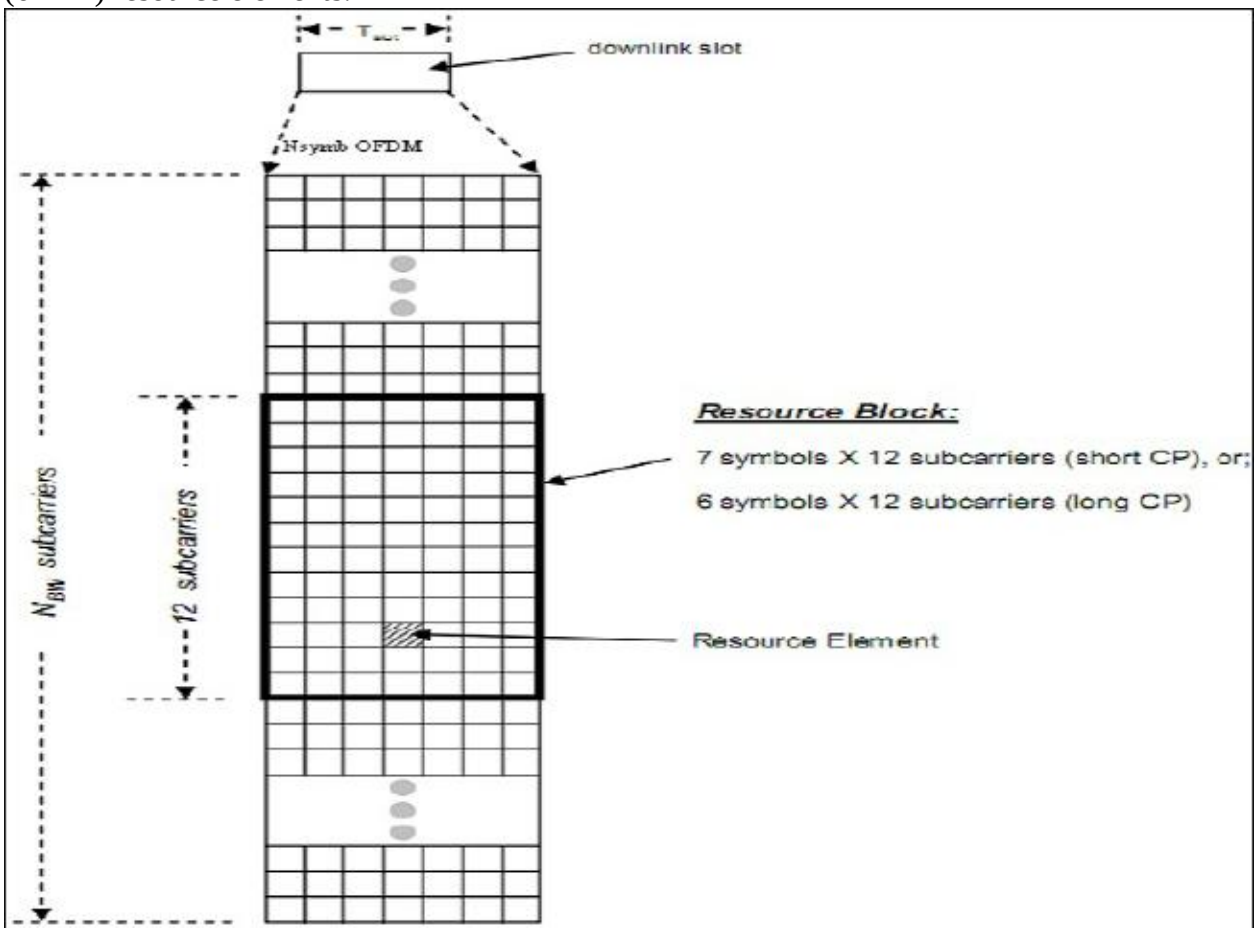


Figure 2-8 Resource Block and Resource Element

2.2.4 Downlink Resource allocation

The physical layer specification allows for a downlink carrier to consist of any number of resource blocks, ranging from a minimum of 6 resource blocks up to a maximum of 100 (equivalent to a downlink transmission bandwidth ranging from around 1.4 MHz up to 20 Mhz). However only certain bandwidths are specified for LTE systems as mentioned before in 'Table 2.2'.
Once the bandwidth is known also the number of available resource blocks for scheduling is known as described in 'table 2.4'.

Channel Bandwidth (MHz)	1.4	3	5	10	15	20
Number of resource blocks	6	15	25	50	75	100

Table 2-4 Available RBs for different Channel BWs

Downlink transmitted resource grid

As mentioned, the number of resource blocks depend on the channel BW available.
In 'figure 2.9' an illustration for the transmitted resource grid for a 1.4 MHz Channel (6 RB) available in frequency domain and a 1 frame (10 ms) representing 140 symbols (for normal CP) in time domain generated by MATLAB.

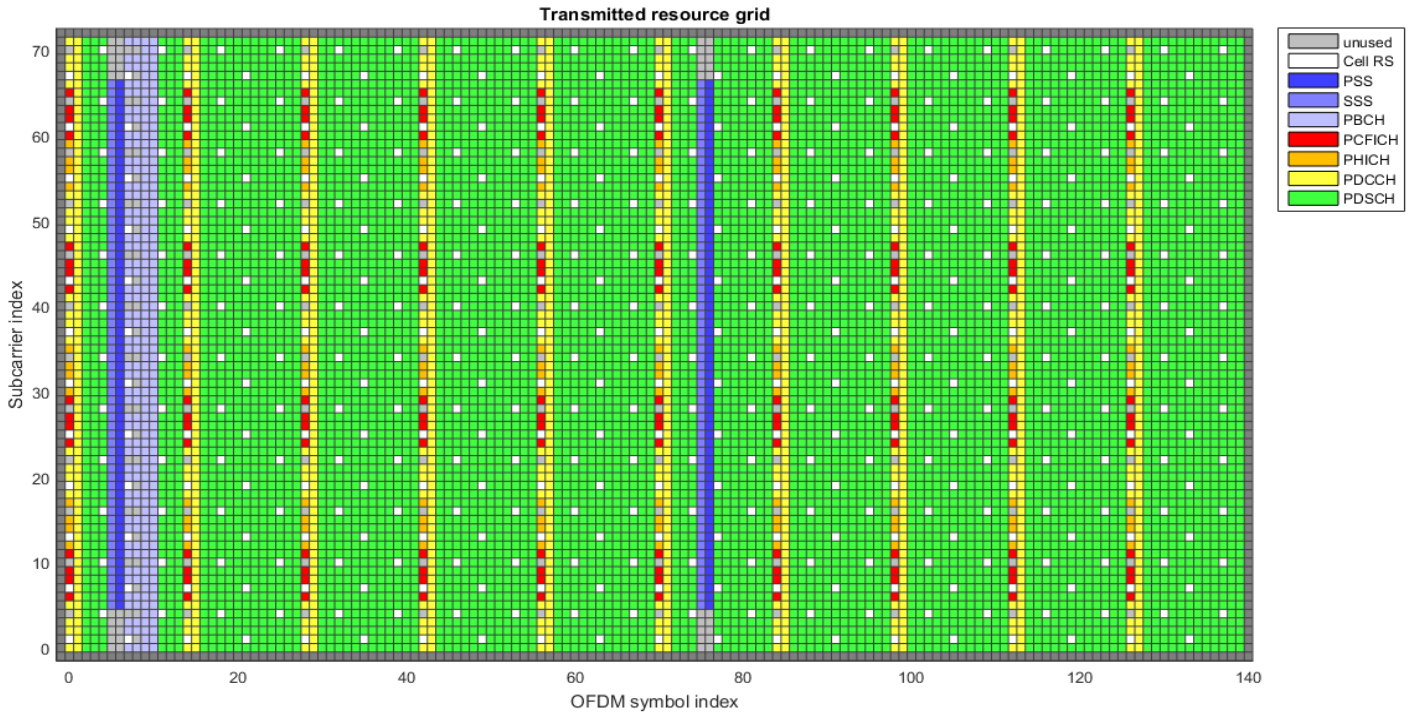


Figure 2-9 transmitted resource grid for 1.4 MHz channel

3 LTE SCANNER

The LTE scanner is a critical test equipment for mobile communication, network planning, and evaluation. It receives and decodes broadcast base-stations signaling to evaluate signal quality and network information. The data decoding and processing is done at different places to get useful information that will make us able to know where the coverage is good and where is not.

The scanner has many uses:

1- Check signal availability for a potential or existing installation
2- Determine the best operator for a particular location
3- Optimize Antenna types
4- Optimize Antenna location

Scanners must be portable to record the signal quality at different locations. An added functionality in a scanner is a scanner equipped with a GPS, this can be used to record the signal quality and record the location at the same time so that it can plot the coverage. Scanners has USB for recharging, programming and connecting to PC to save the results.

The term Test-drive when mentioned with the scanner refers to recording the quality of the signal and its location while moving with a car and to plot coverage.

4 SDR OVERVIEW

With the exponential growth in the ways and means by which people need to communicate - data communications, voice communications, video communications, broadcast messaging, command and control communications, emergency response communications, etc. – modifying radio devices easily and cost-effectively has become business critical. Software defined radio (SDR) technology brings the flexibility, cost efficiency and power to drive communications forward, with wide-reaching benefits realized by service providers and product developers through to end users.

4.1 SDR "SOFTWARE DEFINED RADIO":

A Software Defined Radio (SDR) is a radio that can be reconfigured and reprogrammed through a software (or firmware). Unlike traditional radios which are designed for specific frequency ranges and applications, a software defined radio can be programmed to operate at a required frequency. SDRs usually have a broad frequency range by default and can be programmed to operate at a specific frequency within that range.

So an SDR that has a frequency range from DC to 6 GHz can be used for Wi-Fi (2.4 GHz and 5 GHz), ISM Band Applications (400 MHz, 900 MHz, 2.5 GHz) and various other applications within the DC to 6 GHz frequency range.

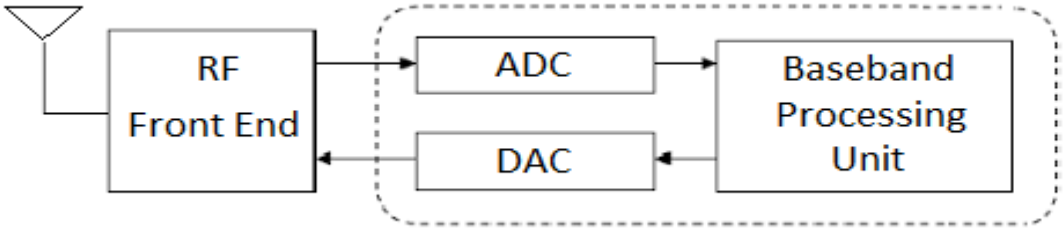


Figure 4-1 SDR Block diagram

Traditional radios are implemented using hardware like mixers, filters, amplifiers, modulators, demodulators etc. SDR's use an ADC and DAC to convert a signal from Analog to Digital and then from Digital to Analog without the need for the various hardware components.

The concept of software-defined radios has been around for some time, however, their application has been limited. With higher frequency and higher resolution ADCs and DACs available, these radios are starting to become more popular and we are seeing them being used more and more.

4.2 C700 WIRELESS MODULAR TEST & DEVELOPMENT PLATFORM:

- The C700 system constitutes an architecture of the primary blocks building any radio system. Users build their systems on the C700 Base Unit which is responsible for synchronization, power management, data communication and mechanical fixation of other blocks. With a bus speed of 700Mbps, communication takes place between modules and the Base Unit.
- A truly modular architecture replaces conventional boards with the versatility required for innovation in SDR (Software Defined Radio), Cognitive Radio, 2x2MIMO, 4x4MIMO, 8x8MIMO, Massive MIMO and Multi-Channel applications. Modules or even entire units can be stacked and aggregated for large scale designs and easily connected to your PC for control, data I/O or as hardware in the loop for simulation.

4.2.1 Features:

- Direct FPGA access - Custom user Code
- Supports Altera & Xilinx FPGAs & tools
- Frequency range: 400 MHz to 6 GHz
- Wide IQ Bandwidth up to 40 MHz
- Integrates with your design environment of choice like MATLAB, LabVIEW, GNU Radio & more…
- Digital I & Q with 16-bit resolution

4.2.2 SYSTEM ARCHITECTURE AND PORTS:

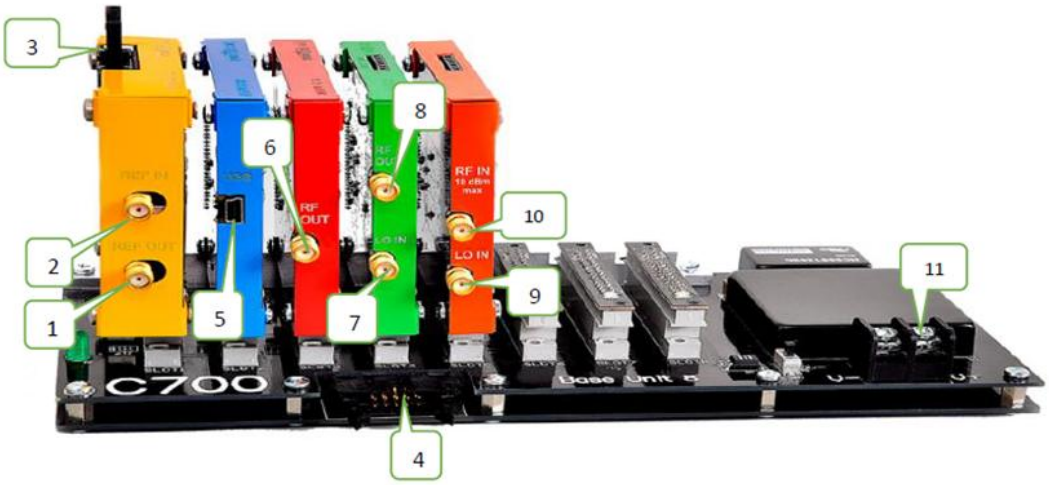


Figure 4-2 C700 Architecture

Port	Description
1	10 MHz Reference O/P
2	External Reference I/P (10 MHz)
3	Clock (50 MHz) & Timing O/P
4	Clock (50 MHz) & Timing I/P
5	USB2 Port (Communication to host)
6	Local Oscillator O/P (PLL)
7	VSM Module; Local Oscillator I/P
8	VSM Module; RF Vector (Digital) Signal O/P
9	VSD Module; Local Oscillator I/P
10	VSD module; RF Vector (Digital) Signal I/P
11	DC Input

Table 4-1 C700 Ports

4.2.3 SPECIFICATIONS:

- C700 Modules are user accessible to allow easy debugging and to enable user insertion of hardware in the loop. Innovation modules are programmable to redefine its functionality and involve user code as a block in the system. In any configuration, C700 can be used as a standalone system (using integrated controller), driven by an external host or employed as a subsystem in more complex systems.
- The firmware natively supports programming the FPGA with custom HDL code through direct JTAG access.

C700-BU: SYSTEM BASE UNIT

- The system base Unit is responsible for synchronization, power management, data communication and mechanical fixation of other blocks.

- The base unit connects the system modules utilizing a ring topology (so mount a bypass cap in any non-used slots) with a proprietary addressing and communication protocol of speed of 700 Mbps.

Figure 4-3 C700-BU: system base unit

C700-SYNC: SYNCHRONIZATION MODULE

Figure 4-4 c700-sync: synchronization module

- The C700-Sync module provides synchronization and timing information to the whole system, and generates a reference clock for all modules (system clock = 50 MHz)

CONTROLLER/HOST INTERFACE

Figure 4-5 controller/host interface

- C700 can be controlled using integrated controller C700-AVR1 for standalone applications, or to use the C700-USB2 Communication module to control the system by an external host; our Case in this project.

C700-PLL: SYSTEM LOCAL OSCILLATOR MODULES

Figure 4-6 system local oscillator modules

- PLL modules provide highly stable RF local oscillator to other system modules.

C700-VSM1 and VSD:

Figure 4-7 C700-vsm1 and vsd

- VSM; Vector (digital) signal generator and modulator for TX path
- VSD; Vector (digital) signal demodulation & digitization for RX Path

4.3 USRP N210:

4.3.1 N210 product overview:

Figure 4-8 N210 product overview

4.3.2 N210 Features:

- Use with GNU Radio, LabVIEW™ and Simulink™
- Modular Architecture: DC-6 GHz
- Dual 100 MS/s, 14-bit ADC
- Dual 400 MS/s, 16-bit DAC
- DDC/DUC with 25 MHz Resolution
- Up to 50 MS/s Gigabit Ethernet Streaming
- Fully-Coherent MIMO Capability
- Gigabit Ethernet Interface to Host
- 2 Gbps Expansion Interface
- Spartan 3A-DSP 3400 FPGA
- 1 MB High-Speed SRAM
- Auxiliary Analog and Digital I/O
- 2.5 ppm TCXO Frequency Reference
- ppm w/ GPSDO Option

4.3.3 N210 Block diagram:

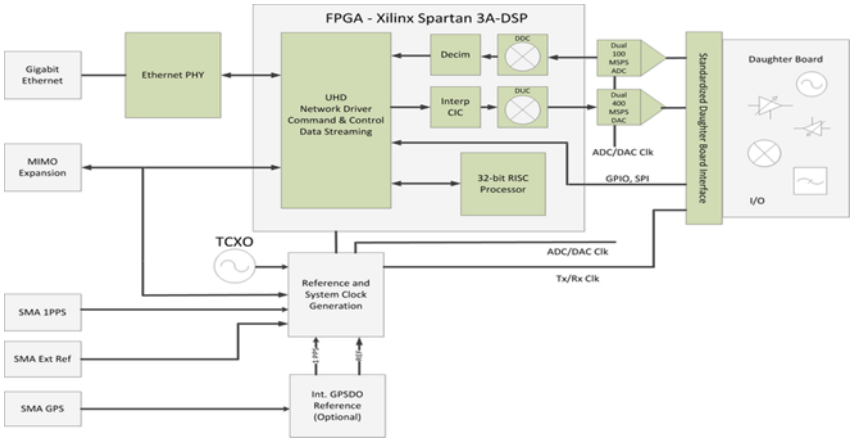


Figure 4-9 N210 Block digram

4.3.4 N210

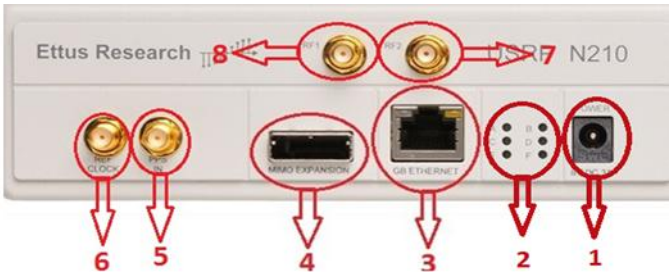


Figure 4-10 N210 interfaces

1)Power:

- DC Input 6 V
- Current Consumption 1.3 A

2)Front panel LEDs:

- LED A: transmitting
- LED B: MIMO cable link
- LED C: receiving
- LED D: firmware loaded
- LED E: reference lock
- LED F: CPLD loaded

3) Gigabit Ethernet.

4)MIMO expansion.

5) PPS in - Pulse Per Second.

6) Ref Clock - 10 MHz, 7&8) TX & RX

4.4 RTL-SDR:

- RTL-SDR is a very cheap USP dongle that can be used as a computer-based radio scanner for receiving live radio signals in your area (no internet required). Depending on the particular model it could receive frequencies from 500 kHz up to 1.75 GHz.
- Osmocom they are the ones who developed the drivers and brought RTL-SDR to life.
- we can use RTL-SDR dongle with different softwares as matlab, GNU Radio & Evrytania.
- In our project we work on Mini Portable Digital USB 2.0 TV Stick DVB-T + DAB + FM RTL2832U + R820T2 Support SDR Tuner Receiver.

Figure 4-11 RTL-SDR

Mini Portable Digital USB 2.0 TV Stick DVB-T + DAB + FM RTL2832U + R820T2 Support SDR Tuner Receiver

price	About 15$
Features	-Supports Digital terrestrial video and radio program recording (records digital terrestrial TV on PC or Laptop). -The included Input Terrestrial Antenna will offer stable TV signal. -Supports Real Time digital video recording and broadcasting. -Supports Watching DVB-T digital TV and listen to DAB and FM radio. -Full DVB-T bandwidth reception (at 6 / 7 / 8MHz) -Supports Still image snapshots and PIP (Picture in picture) function. -Can Schedule recording using the built-in Scheduler and Time-shifting function -Digital TV recording and playback as DVD quality -Support both MPEG-2, MPEG-4(H.264) encoding -Support multi-picture display -Support EPG (Electronic Program Guide); Support Teletext -Support Microsoft BDA drivers -Supports Remote control -Support Systems: Windows 2000/XP/Vista/WIN7/WIN8
Specifications	The RTL2832U outputs 8-bit I/Q-samples, and the highest theoretically possible sample-rate is 3.2 MS/s, however, the highest sample-rate without lost samples that has been tested wit regular USB controllers so far is 2.4 MS/s. A stable sample-rate of 3.2 MS/s without lost samples is only possible with the Etron EJ168/EJ188/EJ198 series of host controllers due to their specific maximum latency. The frequency range is highly dependent of the used tuner

Table 4-2 RTL-SDR features and specs

- **Types of tuners:**

Tuner	Frequency range
Elonics E4000	52 - 2200 MHz with a gap from 1100 MHz to 1250 MHz (varies)
Rafael Micro R820T	24 - 1766 MHz
Rafael Micro R828D	24 - 1766 MHz
Fitipower FC0013	22 - 1100 MHz (FC0013B/C, FC0013G has a separate L-band input, which is unconnected on most sticks)
Fitipower FC0012	22 - 948.6 MHz
FCI FC2580	146 - 308 MHz and 438 - 924 MHz (gap in between)

Table 4-3 RTL-SDR types of tuners

Note: The type of tuner in our RTL-SDR is" Rafael Micro R820T".

4.5 EVRYTANIA SOFTWARE:

- This is a collection of open source software (OSS) tools to locate and track LTE basestation cells using very low performance RF front ends. For example, these tools work with RTL2832 based dongles (E4000, R820T, etc.) which have a noise figure of 20dB, only 8 bits in the A/D, and a crystal with a frequency error of about 100 PPM. In fact, since the LTE downlink center frequencies are accurate to within 0.050 PPM, these tools can be used to calibrate the receiver's oscillator.

4.5.1 Evrytania LTE Tools:

- **Evrytania software consists of two tools:**
  - CellSearch tool: CellSearch is open source software that searches a set of carrier frequencies for LTE basestation signals and returns a list of all cells found. A cell is considered identified if the master information block (MIB) can be successfully decoded
  - LTE-Tracker tool: LTE-Tracker is open source software that continuously searchers for LTE cells on a particular frequency and then tracks, in realtime, all found cells. With the addition of a GPS receiver, this program can be used to obtain basic cellular coverage maps.
    Note: In our project we will focus on “ CellSearch tool ”.

5 FPGA

Since the digital team main purpose is to implement the scanner on the FPGA a detailed research on FPGAs is needed. In this chapter we will talk about FPGA architecture, FPGA programming, embedded elements in the FPGA, and other FPGA related topics .

5.1 INTRODUCTION

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC).

FPGAs contain an array of programmable logic blocks, and a hierarchy of "reconfigurable interconnects" that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. Many FPGAs can be reprogrammed to implement different logic functions, allowing flexible reconfigurable computing as performed in computer software.

5.2 PROGRAMMING TECHNOLOGIES

There are Multiple Technologies used in programming the FPGA, but we will talk about only two which are widely used by most companies ‘SRAM’ and ‘Antifuse’.

5.2.1 SRAM-Based FPGAs

The majority of FPGAs are based on the use of SRAM configuration cells, which means that they can be configured over and over again. The main advantages of this technique are that new design ideas can be quickly implemented and tested, while evolving standards and protocols can be accommodated relatively easily. Furthermore, when the system is first powered up, the FPGA can initially be programmed to perform one function such as a self-

test or board/system test, and it can then be reprogrammed to perform its main task. Another big advantage of the SRAM-based approach is that these devices are at the forefront of technology. FPGA vendors can leverage the fact that many other companies specializing in memory devices expend tremendous resources on research and development (R&D) in this area. Furthermore, the SRAM cells are created using exactly the same CMOS technologies as the rest of the device, so no special processing steps are required in order to create these components. One downside of SRAM-based devices is that they have to be reconfigured every time the system is powered up. This either requires the use of a special external memory device (which has an associated cost and consumes real estate on the board), or of an on-board microprocessor (or some variation of these techniques).

Another consideration with regard to SRAM-based devices is that it can be difficult to protect your intellectual property, or IP, in the form of your design. This is because the configuration file used to program the device is stored in some form of external memory.

On the bright side, some of today's SRAM-based FPGAs support the concept of bitstream encryption. In this case, the final configuration data is encrypted before being stored in the external memory device. The encryption key itself is loaded into a special SRAM-based register in the FPGA via its JTAG port. The main downside to the encrypted bitstream scheme is that you require a battery backup on the circuit board to maintain the contents of the encryption key register in the FPGA when power is removed from the system.
This battery will have a lifetime of years or decades because it need only maintain a single register in the device, but it does add to the size, weight,
complexity, and cost of the board.

5.2.2 Antifuse-Based FPGAs

On Unlike SRAM-based devices, which are programmed while resident in the system, antifuse-based devices are programmed off-line using a special device programmer. The proponents of antifuse-based FPGAs are proud to point to an assortment of (not-insignificant) advantages:
1. these devices are **nonvolatile** (their configuration data remains when
the system is powered down), which means that they are immediately available as soon as power is applied to the system, and they don't require an external memory chip to store their configuration data, which saves the cost of an additional component and saves real estate on

the board.

2. Another noteworthy advantage of antifuse-based FPGAs is the fact that their interconnect structure is naturally **rad hard**, which means they are relatively immune to the effects of radiation. This is of particular interest in the case of military and aerospace applications because the state of a configuration cell in an SRAM-based component can be " flipped " if that cell is hit by radiation (of which there is a lot in space). By comparison, once an antifuse has been programmed, it cannot be altered in this way.

3.Perhaps the most significant advantage of antifuse-based FPGAs is that their **configuration data is buried deep inside them, making it almost impossible to reverse-engineer the design**. By default, it is possible for the device programmer to read this data out because this is actually how the programmer works. As each antifuse is being processed, the device programmer keeps on testing it to determine when that element has been fully programmed; then it moves on to the next antifuse. Furthermore, the device programmer can be used to automatically verify that the configuration was
performed successfully (this is well worth doing when you're talking about devices containing 50 million plus programmable elements). Once the device has been programmed, however, it is possible to set (grow) a special security antifuse that subsequently prevents any programming data (in the form of the presence or absence of antifuses) from being read out of the device. Even if the device is decapped (its top is removed), programmed and unprogrammed antifuses appear to be identical, and the fact that all of the antifuses are buried in the internal metallization layers makes reverse engineering close to impossible.

Of course, the main disadvantage associated with antifuse-based devices is that they are OTP, so once you've programmed one, its function is set in stone. This makes these components a poor choice for use in a development or prototyping environment.

In our case it's very hard to use antifuse devices because of the continuous modifications made, so we will stick with SRAM-Based FPGAs.

5.2.3 Technologies Comparisons

Table 5.1 briefly summarizes the key points associated with the various programming technologies described above:

Feature	SRAM	Antifuse
Technology node	State-of-the-art	One or more generations behind
Reprogrammable	Yes (in system)	No
Reprogramming speed (inc. erasing)	Fast	–
Volatile (must be programmed on power-up)	Yes	No
Requires external configuration file	Yes	No
Good for prototyping	Yes (very good)	No
Instant-on	No	Yes
IP Security	Acceptable (especially when using bitstream encryption)	Very Good
Size of configuration cell	Large (six transistors)	Very small
Power consumption	Medium	Low
Rad Hard	No	Yes

Figure 5-1 comparison of Programming technologies

5.2.4 FINE-, MEDIUM-, AND COARSE-GRAINED ARCHITECTURES

It is common to categorize FPGA offerings as being either fine grained or coarse grained. In order to understand what this means, we first need to remind ourselves that the main feature that distinguishes FPGAs from other devices is that their underlying fabric predominantly consists of large numbers of relatively simple programmable logic block “islands” embedded in a “sea” of programmable interconnect as shown in figure 5.1.

In the case of a fine-grained architecture, each logic block can be used to

implement only a very simple function. For example, it might be possible to configure the block to act as any 3-input function, such as a primitive logic gate (AND,

OR, NAND, etc.) or a storage element (D-type flip-flop, D-type latch, etc.).
In the case of a coarse-grained architecture, each logic block contains a relatively large amount of logic compared to their fine-grained counterparts.
For example, a logic block might contain four 4-input LUTs, four multiplexers, four D-type flip-flops, and some fast carry logic .

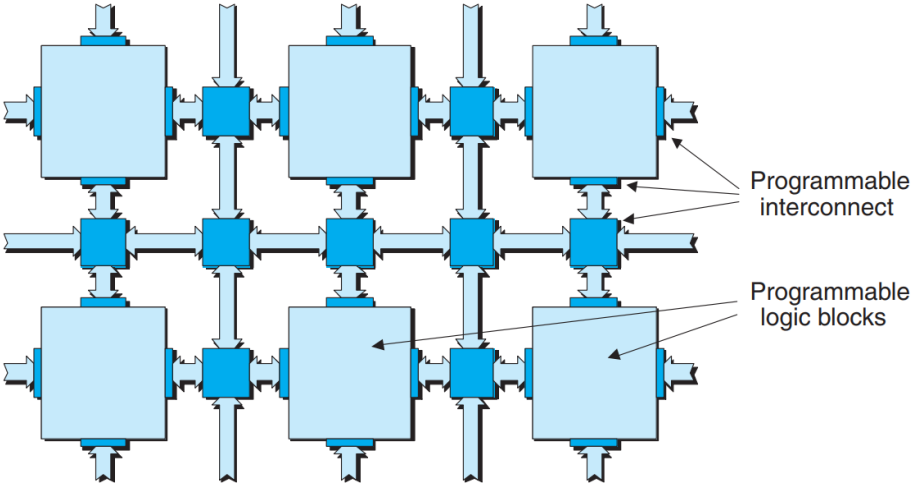


Figure 5-2 Underlying FPGA Fabric

An important consideration with regard to architectural granularity is that fine-grained implementations require a relatively large number of connections into and out of each block compared to the amount of functionality that can be supported by those blocks. As the granularity of the blocks increases to medium grained and higher, the amount of connections into the blocks decreases compared to the amount of functionality they can support. This is important because the programmable interblock interconnect accounts for the vast majority of the delays associated with signals as they propagate through an FPGA.

5.3 LOGIC BLOCKS

5.3.1 MUX-based

As an example of a MUX-based approach, consider one way in which the 3-input function y= (a & b) | c could be implemented using a block containing only multiplexers (Figure 5.2). The device can be programmed such that each input to the block is presented with a logic 0, a logic 1, or the true or inverse version of a signal (a, b, or c in this case) coming from another

block or from a primary input to the device. (The x shown on the input to the central multiplexer in Figure 5.2 indicates that we don't care whether this input is connected to a 0 or a 1.)

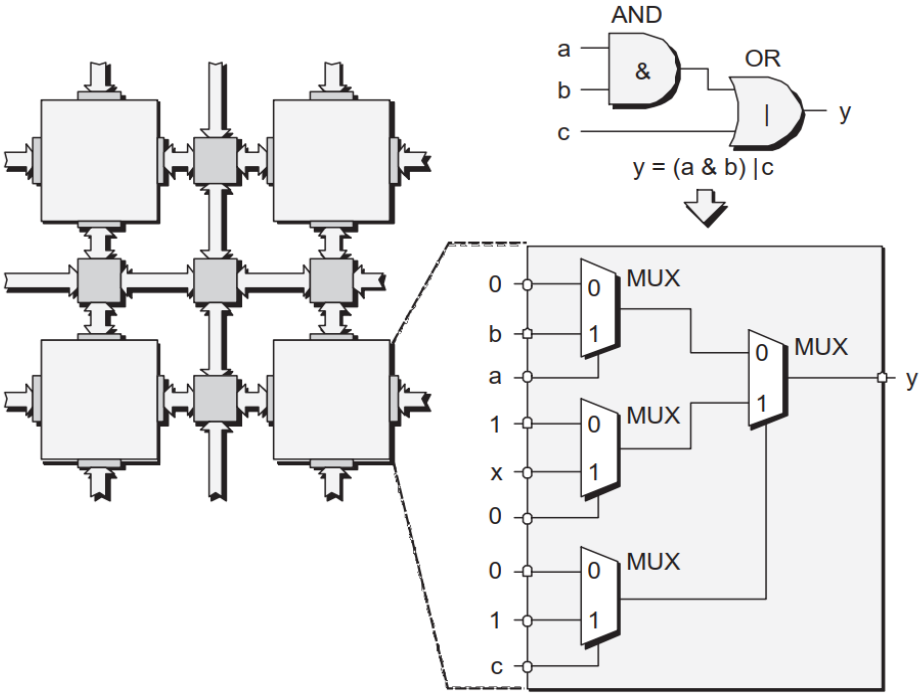


Figure 5-3 MUX-based Logic Block

5.3.2 LUT-BASED

The underlying concept behind a LUT is relatively simple. a group of input signals is used as an index (pointer) to a lookup table. the contents of this table are arranged such that the cell pointed to by each input combination contains the
desired value. for example, let's assume that we wish to implement the function:

$$y= (a \& b) \mid c$$

This can be achieved by loading a 3-input LUT with the appropriate values.
For the purposes of the following examples, we shall assume that the LUT is formed from SRAM cells (but it could be formed using antifuses).A commonly used technique is to use the inputs to select the desired SRAM cell using a cascade of transmission gates as shown in Figure 5.3 . (Note that the SRAM cells will also be connected together in a chain for configuration purposes—that is, to load them with the required values—but these connections have been omitted from this illustration to keep things simple). If a transmission gate is enabled (active), it passes the signal seen on its input through to its output. If the gate is disabled, its output is electrically disconnected from the wire it is driving. The transmission gate symbols shown with a small circle (called a " bobble " or a " bubble ") indicate that

these gates will be activated by a logic 0 on their control input. By comparison, symbols without bobbles indicate that these gates will be activated by a logic 1. Based on this understanding, it's easy to see how different input combinations can be used to select the contents of the various SRAM cel

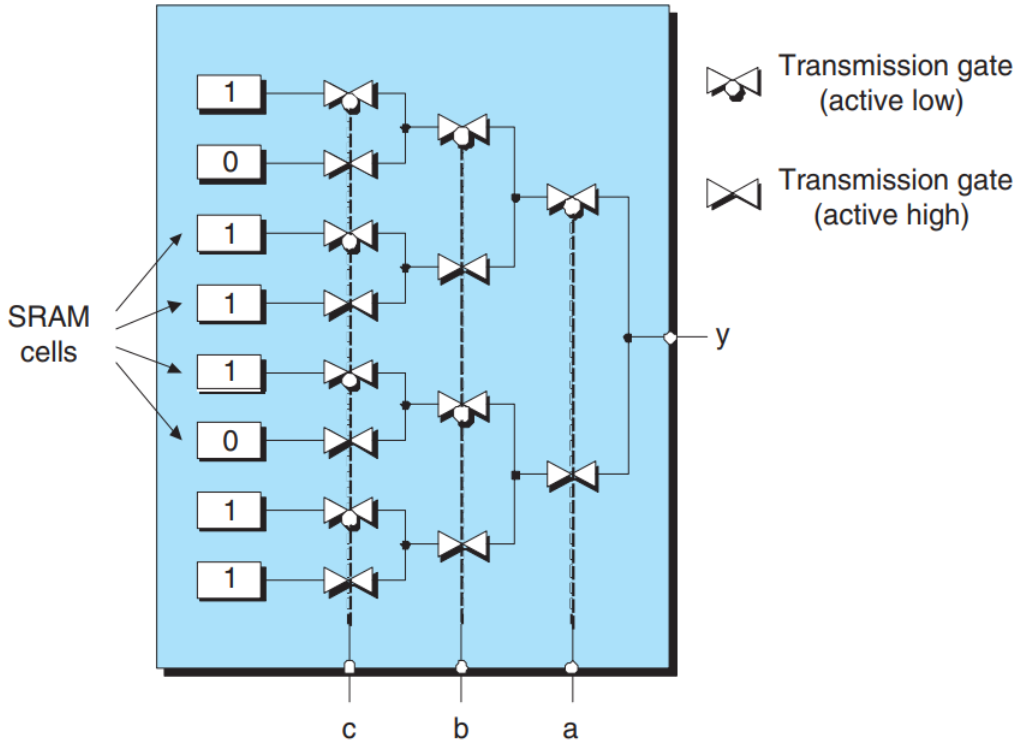


Figure 5-4 A transmission gate based LUT (programming chain omitted for purposes of clarity).

The great thing about an n-input LUT is that it can implement any possible n-input combinational (or combinatorial) logic function. Adding more inputs allows you to represent more complex functions, but every time you add an input, you double the number of SRAM cells. The first FPGAs were based on 3-input LUTs. FPGA vendors subsequently researched the relative merits of 3-, 4-, 5-, and even 6-input LUTs into the ground . The current consensus is that 4-input LUTs offer the optimal balance of pros and cons.

In the past, some devices were created using a mixture of different LUT sizes, such as 3-input and 4-input LUTs, because this offered the promise of optimal device utilization. However, one of the main tools in the design engineer's treasure chest is logic synthesis, and uniformity and regularity are what a synthesis tool likes best. Thus, all of the really successful architectures are currently based only on the use of 4-input LUTs. (This is not to say that mixed-size LUT architectures won't reemerge in the future as design software continues to increase in sophistication.).

5.3.2.1 LUT versus Distributed RAM versus SR

The fact that the core of a LUT in an SRAM-based device comprises a number of SRAM cells offers some interesting possibilities. In addition to its primary role as a lookup table, some vendors allow the cells forming the LUT
to be used as a small block of RAM (the 16 cells forming a 4-input LUT, for example, could be cast in the role of a 16 1 RAM). This is referred to as distributed RAM because (a) the LUTs are strewn (distributed) across the surface of the chip, and (b) this differentiates it from the larger chunks of block RAM (introduced later in this chapter). Yet another possibility devolves from the fact that all of the FPGA's configuration cells—including those forming the LUT—are effectively strung
together in a long chain (Figure 5.4) , some vendors allow the SRAM cells forming a LUT to be treated independently of the main body of the chain and to be used in the form of a shift register. Thus, each LUT may be considered multifaceted (Figure 5.5).

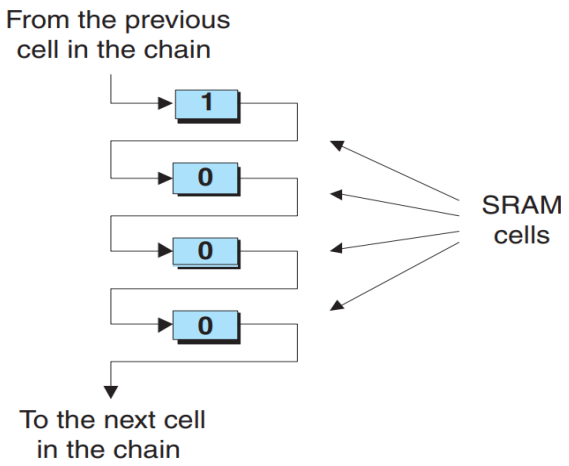


Figure 5-5 Configuration cells linked in a chain

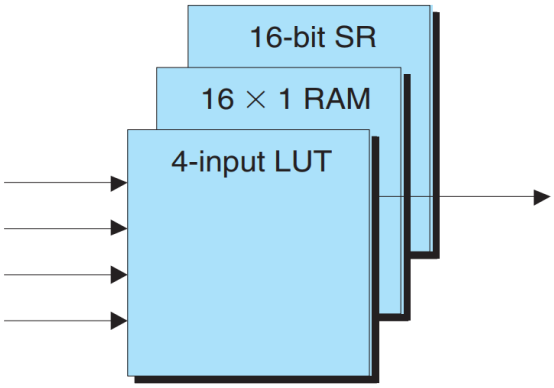


Figure 5-6 A multifaced LUT

5.4 CLBS VERSUS LABS VERSUS SLICES

In addition to one or more LUTs, a programmable logic block will contain other elements, such as multiplexers and registers. But before we delve into this topic, we first need to wrap our brains around some terminology.

5.4.1 Logic Cells/Logic Elements

Each FPGA vendor has its own names for things. For example, the core building block in a modern FPGA from Xilinx is called a logic cell (LC). Among other things, an LC comprises a 4-input LUT (which can also act as a 16 *1 RAM or a 16-bit shift register), a multiplexer, and a register (Figure 5.6).

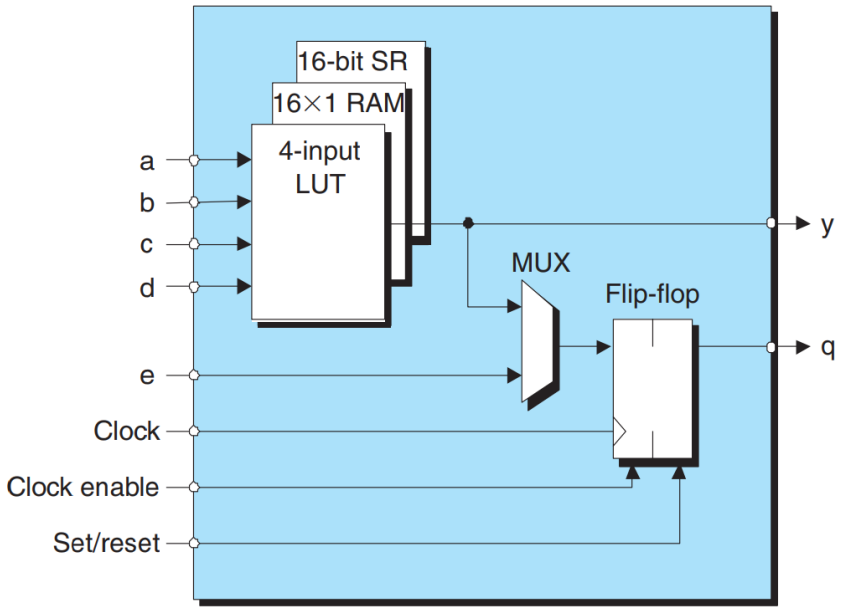


Figure 5-7 A simplified view of Xilinx LC

5.4.2 Slicing and Dicing

The next step up the hierarchy is what Xilinx calls a slice (Altera and the other vendors have their own equivalent names). A slice contains two logic cells (Figure 5.7). The internal wires have been omitted from this illustration to keep things simple; it should be noted, however, that although each logic cell's LUT, MUX, and register have their own data inputs and outputs, the slice has one set of clock, clock enable, and set/reset signals common to both logic cells.

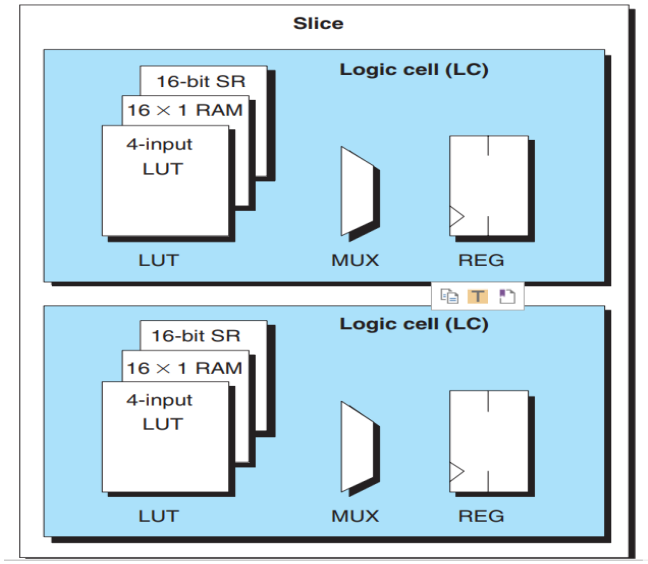


Figure 5-8 A slice containing two logic cells

5.4.3 CLBs and LABs

And moving one more level up the hierarchy, we come to what Xilinx calls a configurable logic block (CLB) and what Altera refers to as a logic array block (LAB). Using CLBs as an example, some Xilinx FPGAs have two slices in each CLB, while others have four. A CLB equates to a single logic block in our original visualization of “ islands ” of programmable logic in a “ sea ” of programmable interconnect (Figure 5.8).There is also some fast-programmable interconnect within the CLB. This interconnect (not shown in Figure 5.8 for reasons of clarity) is used to connect neighboring slices.

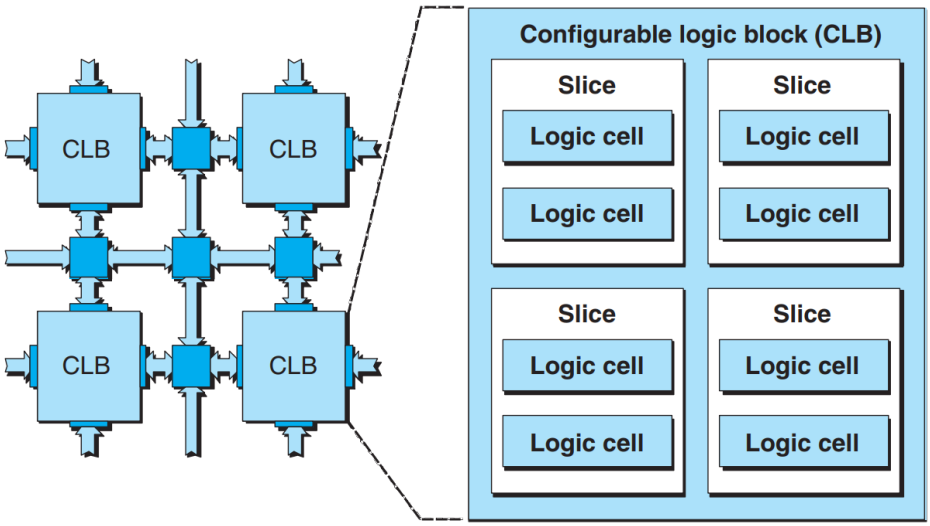


Figure 5-9 A CLB containing four slices (the number of slices depends on the FPGA family)

5.4.4 Distributed RAMs and Shift Registers

This We previously noted that each 4-bit LUT can be used as a 16 1 RAM. And things just keep on getting better because, assuming the four-slices-per-CLB configuration illustrated in Figure 2-9 , all of the LUTs within a CLB can be configured together to implement the following:

- Single-port 16 8 bit RAM
- Single-port 32 4 bit RAM
- Single-port 64 2 bit RAM
- Single-port 128 1 bit RAM

● Dual-port 16 4 bit RAM

● Dual-port 32 2 bit RAM

● Dual-port 64 1 bit RAM

Alternatively, each 4-bit LUT can be used as a 16-bit shift register. In this case, there are special dedicated connections between the logic cells within a slice and between the slices themselves that allow the last bit of one shift register to be connected to the first bit of another without using the ordinary LUT output (which can be used to view the contents of a selected bit within that 16-bit register). This allows the LUTs within a single CLB to be configured together to implement a shift register containing up to 128 bits as required.

5.5 EMBEDDED ELEMENTS INSIDE THE FPGA

FPGA vendors add embedded frequently used functional blocks in their products to make it easier for users, embedded elements vary in functionality and size including elements like: RAMs, DSP Blocks, adders, Multipliers , …etc. In this chapter we'll talk about the most used elements.

5.5.1 Embedded RAMS

Configured Many applications require the use of memory, so FPGAs now include relatively large chunks of embedded RAM called e-RAM or block RAM. Depending on the architecture of the component, these blocks might be positioned around the periphery of the device, scattered across the face of the chip in relative isolation, or organized in columns, as shown in Figure 5.9 .

Depending on the device, such a RAM might be able to hold anywhere from a few thousand to tens of thousands of bits. Furthermore, a device might contain anywhere from tens to hundreds of these RAM blocks, thereby providing a total storage capacity of a few hundred thousand bits all the way up to several million bits.

Each block of RAM can be used independently, or multiple blocks can be combined together to implement larger blocks. These blocks can be used for a variety of purposes, such as implementing standard single- or dual-port RAMs, first-in first-out (FIFO) functions, state machines, and so forth.

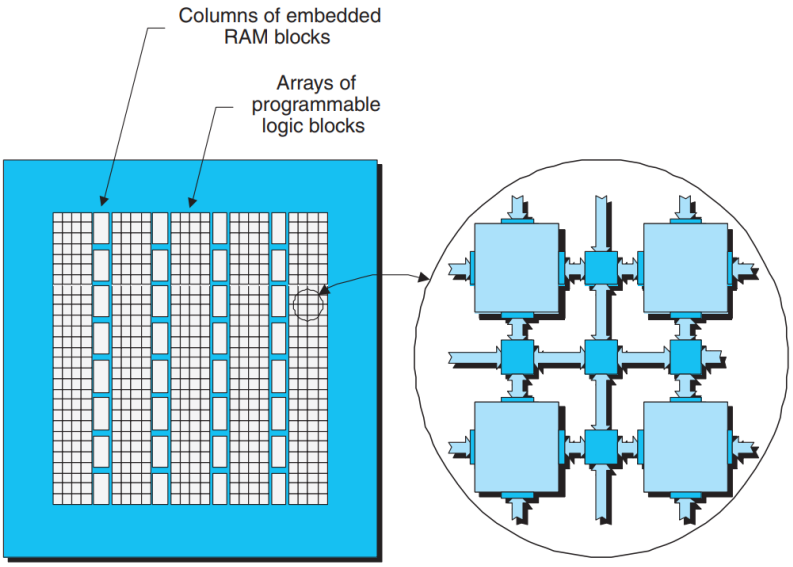


Figure 5-10 embedded RAM Blocks.

5.5.2 Embedded MULTIPLIERS, ADDERS, ETC.

Some functions, like multipliers, are inherently slow if they are implemented by connecting a large number of programmable logic blocks together.

Since many applications require these functions, many FPGAs incorporate special hardwired multiplier blocks. These are typically located in close proximity to the embedded RAM blocks introduced in the previous point because these functions are often used in conjunction with each other(Figure 5.10).

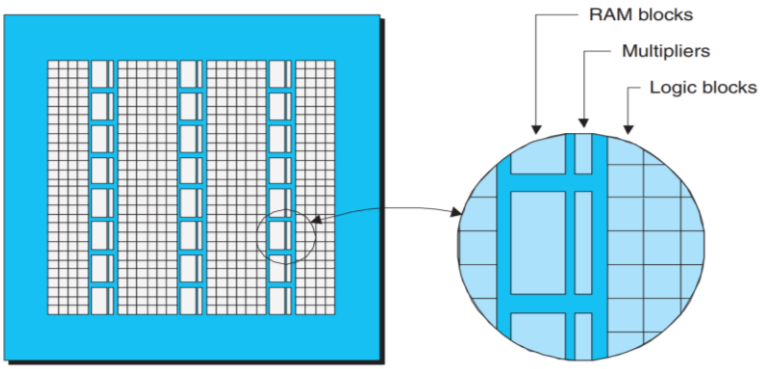


Figure 5-11 columns of embedded multipliers and RAM blocks

Similarly, some FPGAs offer dedicated adder blocks. One operation very common in DSP-type applications is called a multiply-and-accumulate (MAC) (Figure 5.11). As its name would suggest, this function multiplies two numbers together and adds the result to a running total stored in an accumulator.

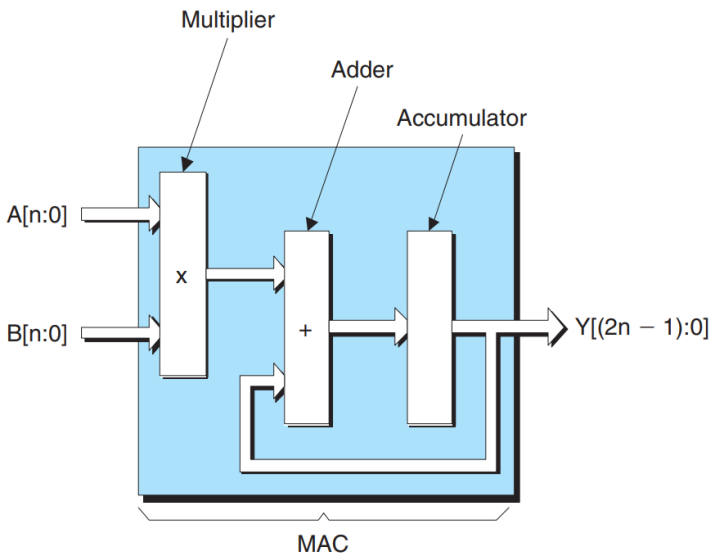


Figure 5-12 The functions forming a MAC.

5.5.3 Embedded Processor Cores

Almost any portion of an electronic design can be realized in hardware (using logic gates and registers, etc.) or software (as instructions to be executed on a microprocessor). One of the main partitioning criteria is how fast you wish the various functions to perform their tasks:

● Picosecond and nanosecond logic: This has to run insanely fast, which mandates that it be implemented in hardware (in the FPGA fabric).

● Microsecond logic: This is reasonably fast and can be implemented either in hardware or software (this type of logic is where you spend the bulk of your time deciding which way to go).

● Millisecond logic: This is the logic used to implement interfaces such as reading switch positions and flashing light-emitting diodes (LEDs). It's a pain slowing the hardware down to implement this sort of function (using huge counters to generate delays, for example). Thus, it's often better to

implement these tasks as microprocessor code (because processors give you lousy speed—compared to dedicated hardware—but fantastic complexity). The fact is that the majority of designs make use of microprocessors in one form or another. Until recently, these appeared as discrete devices on the circuit board. Of late, high-end FPGAs have become available that contain one or more embedded microprocessors, which are typically referred to as microprocessor cores. In this case, it often makes sense to move all of the tasks that used to be performed by the external microprocessor into the internal core. This provides a number of advantages, not the least being that it saves the cost of having two devices; it eliminates large numbers of tracks, pads, and pins on the circuit board; and it makes the board smaller and lighter.

5.5.3.1 Hard Microprocessor Cores

A hard microprocessor core is implemented as a dedicated, predefined block. There are two main approaches for integrating such a core into the FPGA:

1. Locate it in a strip (actually called “ The Stripe ”) to the side of the main FPGA fabric (Figure 5.12). In this scenario, all of the components are typically formed on the same silicon chip, although they could also be formed on two chips and packaged as a multichip module (MCM). The main FPGA fabric would also include the embedded RAM blocks, multipliers, and so on, but these have been omitted from this illustration to keep things simpler. One advantage of this implementation is that the main FPGA fabric is identical for devices with and without the embedded microprocessor core, which can help make things easier for the design tools used by the engineers. The other advantage is that the FPGA vendor can bundle a whole load of additional functions in the strip to complement the microprocessor core, such as memory, special peripherals, and so forth.

2. An alternative is to embed one or more microprocessor cores directly into the main FPGA fabric. One-, two-, and even four-core implementations are currently available (Figure 5.13). In this case, the design tools have to

be able to take account of the presence of these blocks in the fabric; any memory used by the core is formed from embedded RAM blocks, and any peripheral functions are formed from groups of general-purpose programmable logic blocks. Proponents of this scheme will argue that there are inherent speed advantages to be gained from having the microprocessor core in intimate proximity to the main FPGA fabric.

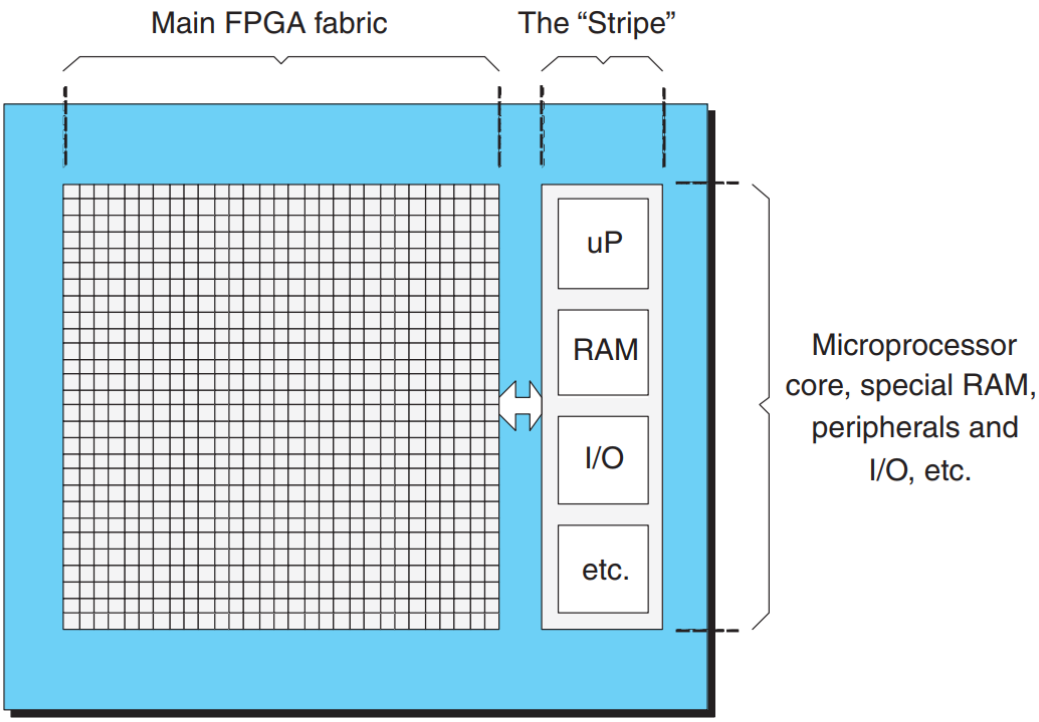


Figure 5-13 chip with embedded core outside of the main fabric

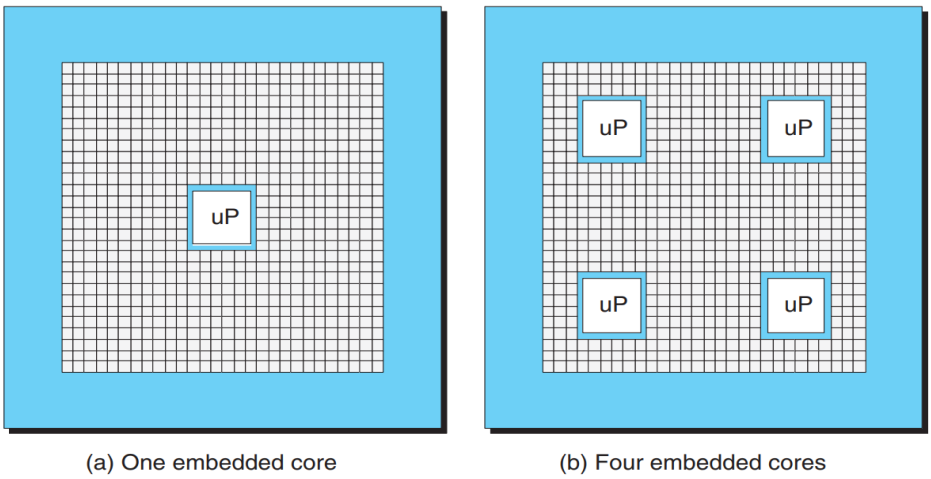


Figure 5-14 chips with embedded cores inside the main fabric.

5.5.3.2 Soft Microprocessor Cores

As opposed to embedding a microprocessor physically into the fabric of the chip, it is possible to configure a group of programmable logic blocks to act as a microprocessor. These are typically called soft cores, but they may be more precisely categorized as either “ soft ” or “ firm ” depending on the way in which the microprocessor’s functionality is mapped onto the logic blocks. Soft cores are simpler (more primitive) and slower than their hard-core counterparts.

A soft core typically runs at 30 to 50 percent of the speed of a hard core, However, they have the advantage that you only need to implement a core if you need it and that you can instantiate as many cores as you require until you run out of resources in the form of programmable logic blocks.

5.5.4 General Purpose I/O

Today’s FPGA packages can have a thousand or more pins, which are arranged as an array across the base of the package. Similarly, when it comes to the silicon chip inside the package, flip-chip packaging strategies allow the power, ground, clock, and I/O pins to be presented across the surface of the chip. Purely for the purposes of these discussions (and illustrations), however, it makes things simpler if we assume that all of the connections to the chip are presented in a ring around the circumference of the device, as indeed they were for many years.

5.5.4.1 Configurable I/O Standards

Let’s consider for a moment an electronic product from the perspective of the architects and engineers designing the circuit board. Depending on what they are trying to do, the devices they are using, the environment the board will operate in, and so on, these designers will select a particular standard to be used to transfer data signals. (In this context, “ standard ” refers to electrical aspects of the signals, such as their logic 0 and logic 1 voltage levels.) The problem is that there is a wide variety of such standards, and it would be painful to have to create special FPGAs to accommodate each variation. For this reason, an FPGA’s general-purpose I/O can be configured to accept and generate signals conforming to whichever

standard is required. These general purpose I/O signals will be split into a number of banks—we'll assume eight such banks numbered from 0 to 7 (Figure 5.13).The interesting point is that each bank can be configured individually to support a particular I/O standard. Thus, in addition to allowing the FPGA to work with devices using multiple I/O standards, this allows the FPGA to actually be used to interface between different I/O standards (and to translate between
different protocols that may be based on particular electrical standards).

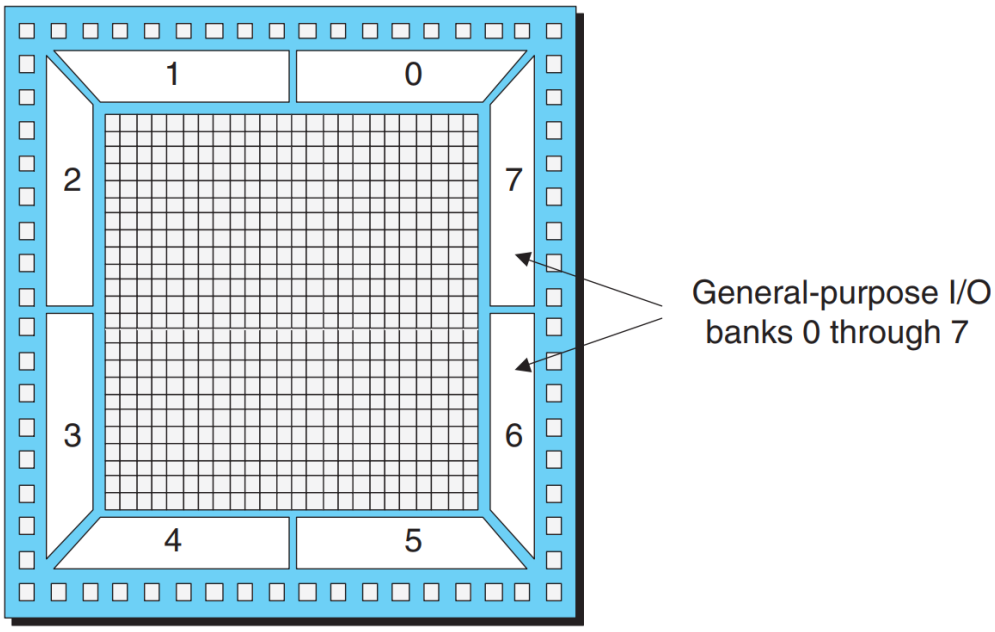


Figure 5-15 chip showing general-purpose I/O banks.

5.5.4.2 Configurable I/O Impedance

The signals used to connect devices on today's circuit board often have fast
edge rates (this refers to the time it takes the signal to switch between one logic value and another). In order to prevent signals reflecting back (bouncing around), it is necessary to apply appropriate terminating resistors to the FPGA's input or output pins. In the past, these resistors were applied as discrete components that were attached to the circuit board outside the FPGA.
However, this technique became increasingly problematic as the number of pins started to increase and their pitch (the distance between them) shrank. For this reason, today's FPGAs allow the use of internal terminating resistors whose values can be configured by the user to accommodate different circuit board environments and I/O standards.

5.5.5 GIGABIT Transceivers

The traditional way to move large amounts of data between devices is to use a bus, a collection of signals that carry similar data and perform a common function (Figure 5.14). Early microprocessor-based systems circa 1975 used 8-bit buses to pass data around. As the need to push more data around and to move it faster grew, buses grew to 16 bits in width, then 32 bits, then 64 bits, and so forth. The problem is that this requires a lot of pins on the device and a lot of tracks connecting the devices together. Routing these tracks so that they all have the same length and impedance becomes increasingly painful as boards grow in complexity. Furthermore, it becomes increasingly difficult to manage signal integrity issues (such as susceptibility to noise) when you are dealing with large numbers of bus-based tracks. For this reason, today's high-end FPGAs include special hard-wired gigabit transceiver blocks. These blocks use one pair of *differential signals* (which means a pair of signals that always carry opposite logical values) to transmit (TX) data and another pair to receive (RX) data (Figure 5.15).These transceivers operate at incredibly high speeds, allowing them to transmit and receive billions of bits of data per second. Furthermore, each block actually supports a number (say four) of such transceivers, and an FPGA may contain a number of these transceiver blocks.

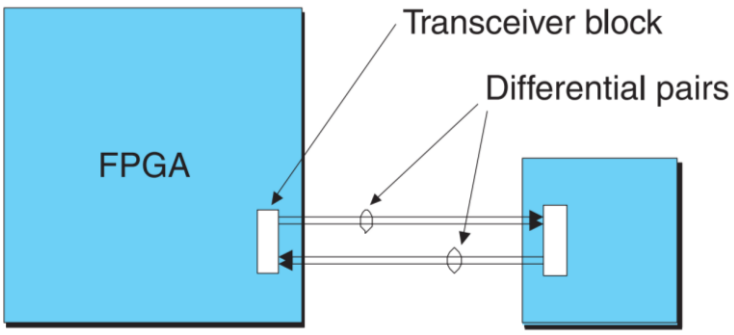


Figure 5-16 Using high-speed transceivers to communicate between devices

5.6 5.6 INTELLECTUAL PROPERTY (IP)

Today's FPGA designs are so big and complex that it would be impractical to create every portion of the design from scratch. One solution is to reuse existing functional blocks for the boring stuff and spend the bulk of your time and resources creating the new portions of the design that will differentiate your design from any competing offerings. Any existing functional blocks are typically referred to as intellectual
property (IP). The three main sources of such IP are:

1. internally created blocks reused from previous designs,

2. FPGA vendors, and

3. third-party IP providers.

For the purposes of these discussions, we shall concentrate on the latter two
categories.

Each FPGA vendor offers its own selection of hard, firm, and soft IP.

Hard IP comes in the form of pre-implemented blocks such as microprocessor cores, gigabit interfaces, multipliers, adders, MAC functions, and the like.

These blocks are designed to be as efficient as possible in terms of power consumption, silicon real estate, and performance. Each FPGA family will feature different combinations of such blocks, together with various quantities of programmable logic blocks.

At the other end of the spectrum, soft IP refers to a source-level library of
high-level functions that can be included to the users ' designs. These functions
are typically represented using a hardware description language, or HDL, such s Verilog or VHDL at the register transfer level (RTL) of abstraction. Any soft IP functions the design engineers decide to use are incorporated into the main body of the design—which is also specified in RTL—and subsequently synthesized down into a group of programmable logic blocks (possibly combined with some hard IP blocks like multipliers, etc.).

Holding somewhat of a middle ground is *firm IP*, which also comes in the form of a library of high-level functions. Unlike their soft IP equivalents, however, these functions have already been optimally mapped, placed, and routed into a group of programmable logic blocks (possibly combined with some hard IP blocks like multipliers, etc.). One or more copies of each predefined firm IP block can be instantiated (called up) into the design as required.

Some IP that used to be " soft " is now becoming " hard. " For example, the
most current generation of FPGAs contains hard processor, clock manager,

Ethernet, and gigabit I/O blocks, among others. These helps bring high-end ASIC functionality into standard FPGAs. Over time, it is likely that additional functions of this ilk will be incorporated into the FPGA device.

5.6.1 Handcrafted IP

One scenario is that the IP provider has handcrafted an IP block starting with an RTL description (the provider might also have used an IP block/core generator application, as discussed later in this chapter). In this case, there are several ways in which the end user might purchase and use such a block (Figure 5.16):

1. Blocks of unencrypted source code. These blocks can then be integrated into the RTL code for the body of the design (Figure 5.16a). (Note that the IP provider would already have simulated, synthesized, and verified the IP blocks before handing over the RTL source code.)

2. Encrypted RTL level. Unfortunately, at the time of this writing, there is no industry-standard encryption technique for RTL that has popular tool support. This has led companies like Altera and Xilinx to develop their own encryption schemes and tools. RTL encrypted by a particular FPGA vendor's tools can only be processed by that vendor's own synthesis tools (or sometimes by a third-party synthesis tool that has been OEM'd by the FPGA vendor).

3. Unplaced-and-unrouted netlist level. Perhaps the most common scenario is for FPGA designers to purchase IP at the unplaced-and-unrouted LUT/CLB netlist level (Figure 5.16b). Such netlists are typically provided in encrypted form, either as encrypted EDIF or using some FPGA vendor-specific format. In this case, the IP vendor may also provide a compiled cycle accurate C/C++ model to be used for functional verification because such a model will simulate much faster than the LUT/CLB netlist-level model.

4. Placed-and-routed netlist level. In certain cases, the FPGA designer may purchase IP at the placed-and-routed LUT/CLB netlist level (Figure 5.16c). Once again, such netlists are typically provided in encrypted form, either as encrypted EDIF or using some FPGA vendor-specific format. The reason for having placed-and-routed representations is to obtain the highest levels of performance. In some cases, the placements will be relative, which means that the locations of all of the LUT, CLB, and other elements forming the block are fixed with respect to each other, but the block as a

whole may be positioned anywhere (suitable) within the FPGA. Alternatively, in the case of IP blocks such as communications or bus protocol functions
with specific I/O pin requirements, the placements of the elements forming the block may be absolute, which means that they cannot be changed in any way. Once again, the IP vendor may also provide a compiled cycle accurate C/C++ model to be used for functional verification because such a model will simulate much faster than the LUT/CLB netlist-level model.

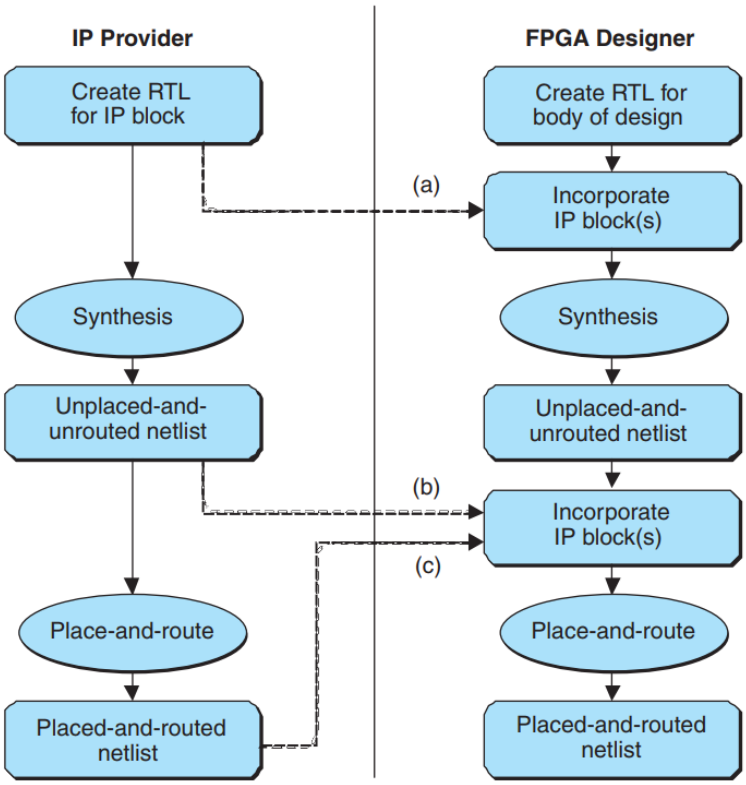


Figure 5-17 Alternative potential IP acquisition points

5.6.2 IP Core Generator

Another very common practice is for FPGA vendors (sometimes EDA vendors,
IP providers, and even small, independent design houses) to provide special tools that act as

IP block/core generators. These generator applications are almost invariably parameterized, thereby allowing you to specify the widths and depths, or both of buses and functional elements.

First, you get to select from a list of different blocks/cores, and then you get to specify the parameters to be associated with each. Furthermore, in the case of some blocks/cores, the generator application may allow you to select from a list of functional elements that you wish to be included or excluded from the final representation. In the case of a communications block, for example, it might be possible to include or exclude certain error-checking logic. Or in the case of a CPU core, it might be possible to omit certain instructions or addressing modes. This allows the generator application to create the most efficient IP block/core in terms of its resource requirements and performance.

Depending on the origin of the generator application (or sometimes the licensing option you've signed up for), its output may be in the form of encrypted or unencrypted RTL source code, an unplaced-and-unrouted netlist, or a placed-and-routed netlist. In some cases, the generator may also output a cycle-accurate C/C++ model for use in simulation (Figure 5.17).

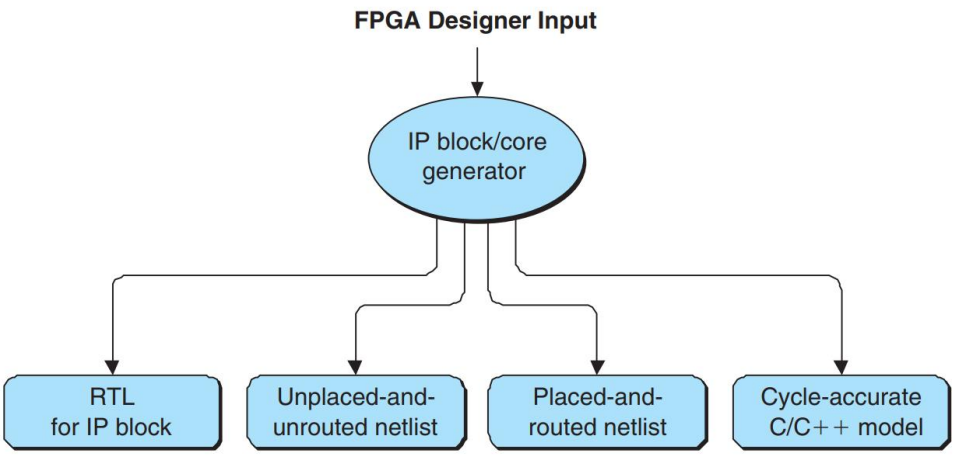


Figure 5-18 IP block/core generator

5.7 FPGA DESIGN FLOW

As a digital team in addition to learning about the LTE basics and FPGA structure we had to acquire the ability to create a configuration bit stream file a ; starting from the RTL code, going through the multiple steps of the flow , and finally creating the bit file, this knowledge is must to be able to decode the lte frames and extract the wanted information. In this section we will talk about the flow of processes to create the bit file.

5.7.1 Overview

In this part we are going to have a short intro on FPGA design flow. A simplified version of design flow is given in figure 5.1.

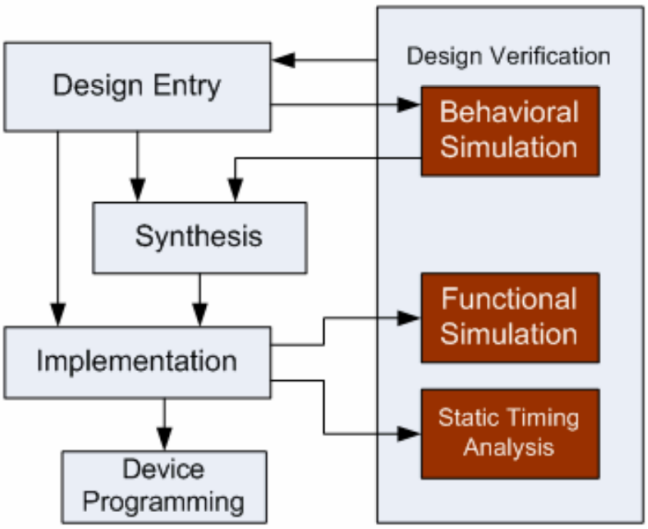


Figure 5-19 FPGA Design Flow.

5.7.2 Design Entry

There are different techniques for design entry. Schematic based, Hardware Description Language and combination of both etc. . Selection of a method depends on the design and designer. If the designer wants to deal more with Hardware, then Schematic entry is the better choice. When the design is complex, or the designer thinks the design in an algorithmic way then HDL is the better choice. Language based entry is faster but lag in performance and density. HDLs represent a level of abstraction that can isolate the designers from the details of the hardware implementation. Schematic based entry gives designers much more visibility into the hardware. It is the better choice for those who are hardware oriented. Another method but rarely used is state

5.7.3 Synthesis

The process which translates VHDL or Verilog code into a device netlist format. i.e. a complete circuit with logical elements(gates, flip flops, etc.…) for the design. If the design contains more than one sub designs, ex. to implement a processor, we need a CPU as one design element and RAM as another and so on, then the synthesis process generates netlist for each design element.

Synthesis process will check code syntax and analyze the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected. The resulting netlist(s) is saved to an NGC(Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).

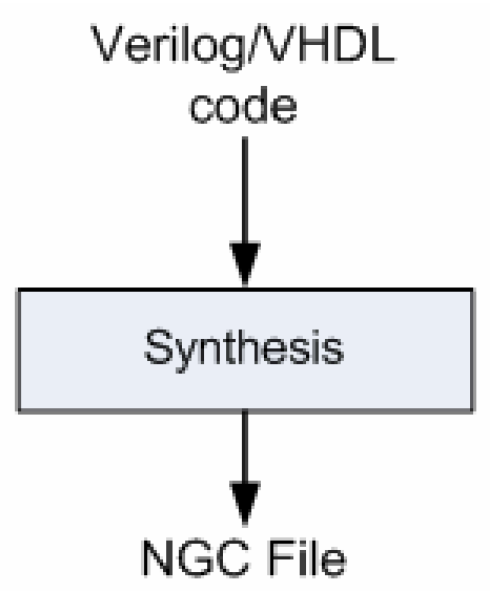


Figure 5-20 Synthesis

5.7.4 Implementation

This process consists a sequence of three steps:

1. Translate
2. Map
3. Place and Route

5.7.4.1 Translate

Translate process combines all the input netlists and constraints to a logic design file. This information is saved as a NGD (Native Generic Database) file. This can be done using NGD Build program. Here, defining constraints is nothing but, assigning the ports in the design to the physical elements (ex. pins, switches, buttons etc) of the targeted device and specifying time requirements of the design. This information is stored in a file named UCF (User Constraints File).

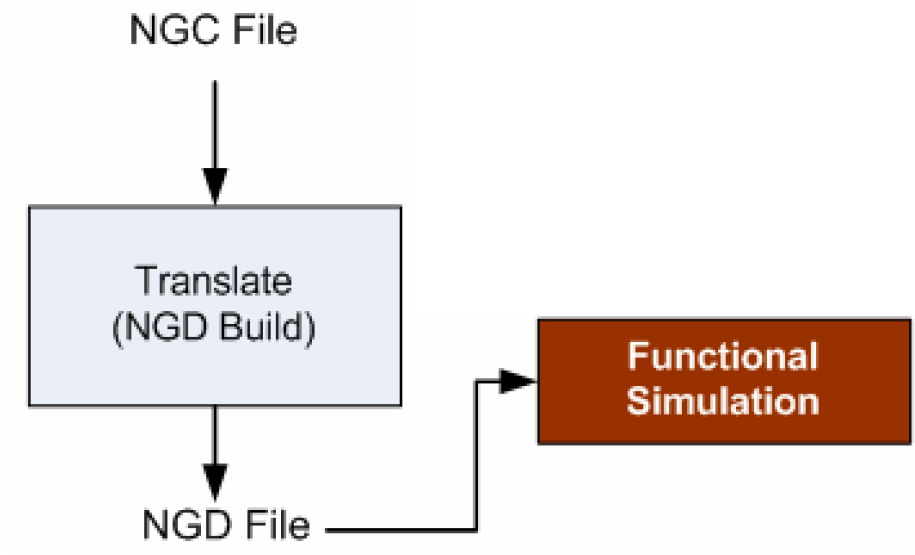


Figure 5-21 Translate process

5.7.4.2 Map

Map process divides the whole circuit with logical elements into sub blocks such that they can be fit into the FPGA logic blocks. That means map process fits the logic defined by the NGD file into the targeted FPGA elements (Combinational Logic Blocks (CLB), Input Output Blocks (IOB)) and generates an NCD (Native Circuit Description) file which physically represents the design mapped to the components of FPGA.

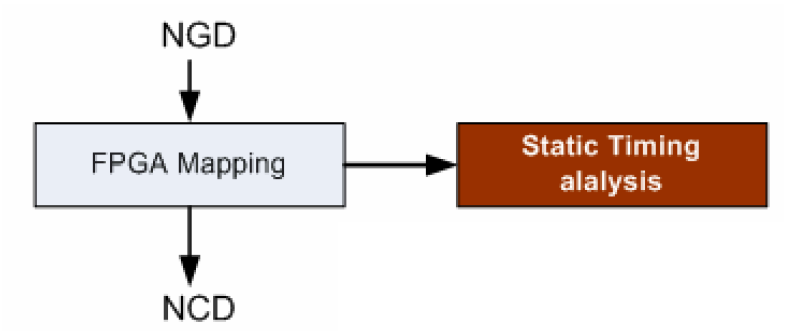


Figure 5-22 Mapping process

5.7.4.3 Placement and Routing:

Place and Route PAR program is used for this process. The place and route process place the sub blocks from the map process into logic blocks according to the constraints and routes the connections between the logic blocks. Ex. if a sub block is placed in a logic block which is very near to IO pin, then it may save time, but it may affect some other constraint. So, a tradeoff between all the constraints is taken account by the place and route process.
The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. Output NCD file contains the routing information.

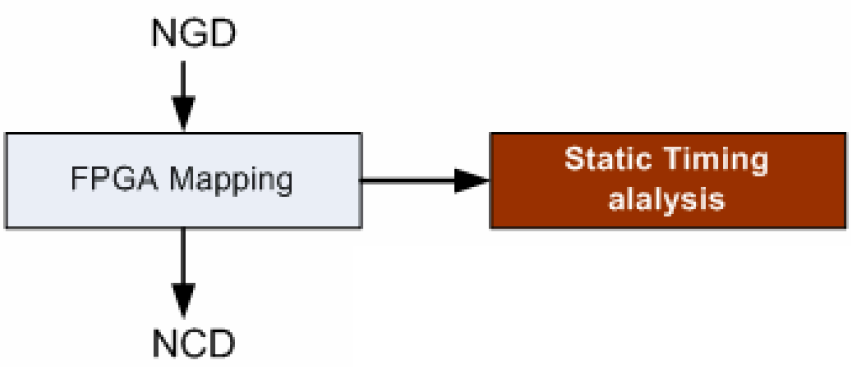


Figure 5-23 Placement and Routing

5.7.5 Device Programming

Now the design must be loaded on the FPGA. But the design must be converted to a format so that the FPGA can accept it. BITGEN program deals with the conversion. The routed NCD file is then given to the BITGEN program to generate a bit stream (a .BIT file) which can be used to configure the target FPGA device. This can be done using a cable. Selection of cable depends on the design.

5.7.6 Design Verification

Verification can be done at different stages of the process steps, to ensure that the functionality of system remains the same and to check the circuit timing, this can be divided in the following process.

5.7.6.1 Behavioral Simulation (RTL Simulation)

This is the first of all simulation steps; those are encountered throughout the hierarchy of the design flow. This simulation is performed before synthesis process to
verify RTL (behavioral) code and to confirm that the design is functioning as intended. Behavioral simulation can be performed on either VHDL or Verilog designs. In this process, signals and variables are observed, procedures and functions are traced, and breakpoints are set. This is a very fast simulation and so allows the designer to change the HDL code if the required functionality is not met with in a short time period. Since the design is not yet synthesized to gate level, timing and resource usage properties are still unknown.

5.7.6.2 Functional Simulation (Post Translate Simulation)

Functional simulation gives information about the logic operation of the circuit. Designer can verify the functionality of the design using this process after the Translate process. If the functionality is not as expected, then the designer has to made changes in the code and again follow the design flow steps.

5.7.6.3 Static Timing Analysis

This can be done after Mapping or placement and routing processes Post MAP timing report lists signal path delays of the design derived from the design logic. Post Place and Route timing report incorporates timing delay information to provide a comprehensive timing summary of the design.

6 PROJECT IDEA AND PLANNING

6.1 PROJECT DESCRIPTION

The aim of this project is to design an LTE network scanner which is an equipment used to test the network planning and evaluation. It receives and decodes all base-stations signalling to evaluate signal quality and network information. The project will be sponsored by Consultix, a leading Egyptian communication equipment developer.

6.2 PROJECT OBJECTIVE

In This Project we are aiming to study LTE system based on 3GPP standards and design LTE scanner, using MATLAB as a simulation tool and implement it on software(C-700).

6.3 PROJECT PLANNING

The project will include 3 groups.

1- One group will be oriented towards mobile communication and will study the 4G standards and simulate all signalling reception and decoding using MATLab.

2- Another group will be oriented towards software-defined-radio SDR and will study the USRP & C700 SDR platforms and use them to implement the network scanner.

3- The last group will be oriented towards digital design and will study the complete digital design FPGA flow, study the architecture of the target FPGA/board, and implement DSP blocks of the network scanner on the target FPGA.

7 MATLAB LTE SCANNER SIMULATION

7.1 INTRODUCTION

In this project we successfully achieved the simulation of cell search procedure (which are mentioned in chapter (11) with the usage of the LTE toolbox system which is instructed to help simulating the physical layer of LTE communication system.

In this part we used MATLab to generate LTE waveforms similar to real LTE waveforms generated from the e-NodeB in real life to justify and test our design.

7.2 GENERATING LTE SIGNALS

The generation of LTE waveforms in matlab can be done by two methods as it will be illustrated in 7.1.1 and 7.1.2

7.2.1 Standard-compliant waveforms

Generating standard-compliant waveforms using LTE Downlink RMC (reference measurement channel) Generator Application Tool with fixed parameters like the system bandwidth and the no. of transmit antenna ports, those waveforms enable thorough testing of LTE components and systems on realistic LTE signals. The parameters of those reference channels are mentioned in table 7.1

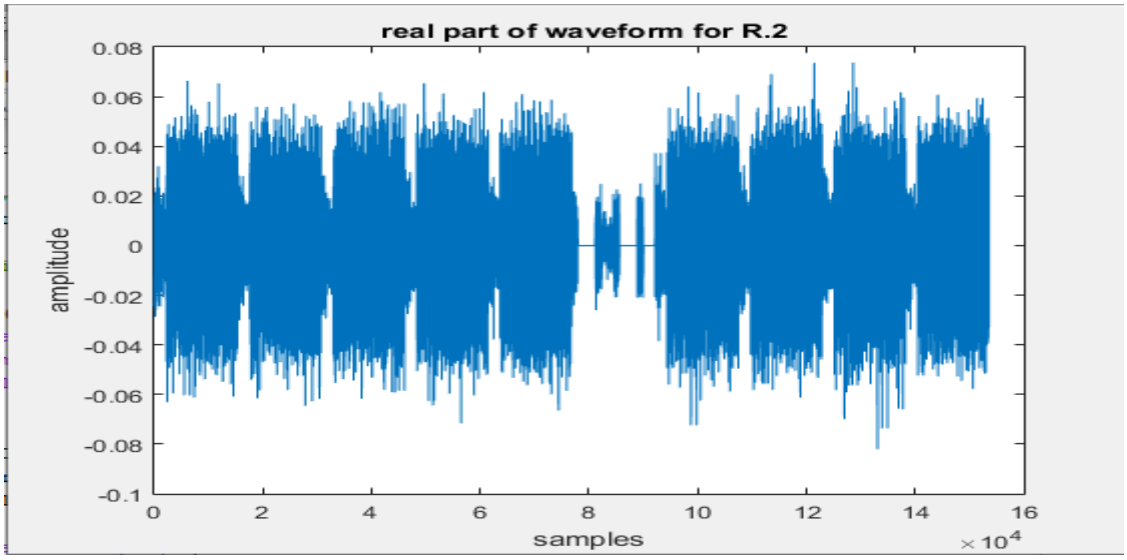


Figure 7-1 real waveform for reference channel R2 generated by MATlab

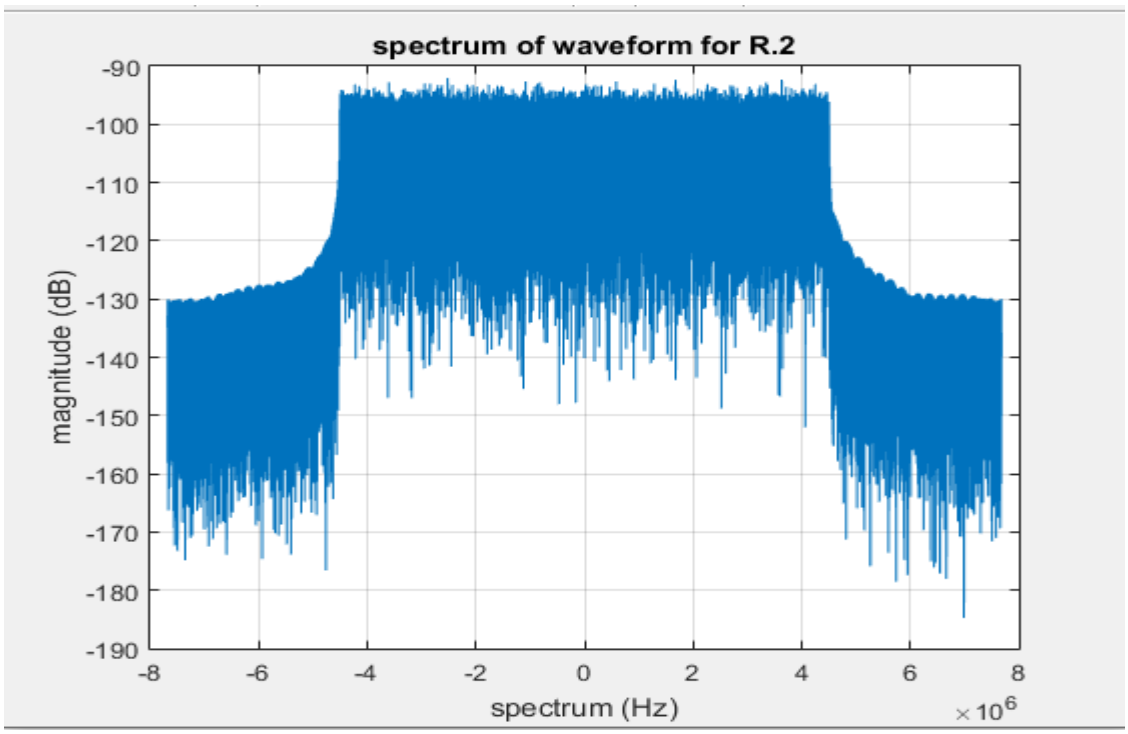


Figure 7-2 spectrum for the refrence channel R2 generated by MATLab

As we can see from the spectrum diagram the bandwidth of the signal is 10 MHz which corresponds to 50 Resource Block (RB) as specified in table 7.1

Reference channels	Reference channels (continued)
R.0 (Port0, 1 RB, 16QAM, CellRefP=1, R=1/2)	R.31-3A FDD (CDD, 50 RB, 64QAM, CellRefP=2, R=0.85-0.90)
R.1 (Port0, 1 RB, 16QAM, CellRefP=1, R=1/2)	R.31-3A TDD (CDD, 68 RB, 64QAM, CellRefP=2, R=0.87-0.90)
R.2 (Port0, 50 RB, QPSK, CellRefP=1, R=1/3)	R.31-4 (CDD, 100 RB, 64QAM, CellRefP=2, R=0.87-0.90)
R.3 (Port0, 50 RB, 16QAM, CellRefP=1, R=1/2)	R.43 FDD (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/3)
R.4 (Port0, 6 RB, QPSK, CellRefP=1, R=1/3)	R.43 TDD (SpatialMux, 100 RB, 16QAM, CellRefP=4, R=1/2)
R.5 (Port0, 15 RB, 64QAM, CellRefP=1, R=3/4)	R.44 FDD (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/3)
R.6 (Port0, 25 RB, 64QAM, CellRefP=1, R=3/4)	R.44 TDD (Port7-14, 50 RB, 64QAM, CellRefP=2, R=1/2)
R.7 (Port0, 50 RB, 64QAM, CellRefP=1, R=3/4)	R.45 (Port7-14, 50 RB, 16QAM, CellRefP=2, R=1/2)
R.8 (Port0, 75 RB, 64QAM, CellRefP=1, R=3/4)	R.45-1 (Port7-14, 39 RB, 16QAM, CellRefP=2, R=1/2)
R.9 (Port0, 100 RB, 64QAM, CellRefP=1, R=3/4)	R.48 (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/2)
R.10 (TxDiversity|SpatialMux, 50 RB, QPSK, CellRefP=2, R=1/3)	R.50 FDD (Port7-14, 50 RB, 64QAM, CellRefP=2, R=1/2)
R.11 (TxDiversity|SpatialMux|CDD, 50 RB, 16QAM, CellRefP=2, R=1/2)	R.50 TDD (Port7-14, 50 RB, QPSK, CellRefP=2, R=1/3)
R.12 (TxDiversity, 6 RB, QPSK, CellRefP=4, R=1/3)	R.51 (Port7-14, 50 RB, 16QAM, CellRefP=2, R=1/2)
R.13 (SpatialMux, 50 RB, QPSK, CellRefP=4, R=1/3)	R.6-27RB (Port0, 27 RB, 64QAM, CellRefP=1, R=3/4)
R.14 (SpatialMux|CDD, 50 RB, 16QAM, CellRefP=4, R=1/2)	R.12-9RB (TxDiversity, 9 RB, QPSK, CellRefP=4, R=1/3)
R.25 (Port5, 50 RB, QPSK, CellRefP=1, R=1/3)	R.11-45RB (CDD, 45 RB, 16QAM, CellRefP=2, R=1/2)
R.26 (Port5, 50 RB, 16QAM, CellRefP=1, R=1/2)	
R.27 (Port5, 50 RB, 64QAM, CellRefP=1, R=3/4)	
R.28 (Port5, 1 RB, 16QAM, CellRefP=1, R=1/2)	

Table 7-1 Parameters of the reference channels used to generate LTE signals

7.2.2 Custom Signals

Generating custom signals by can be achieved by controlling each channel of the LTE physical layer separately then modulating this signal.

7.3 PERFORMING CELL SEARCH AND MIB RECOVERY

After generating the LTE waveform the next step in the simulation is to fully synchronize, demodulate and decode the LTE signal. Before the User Equipment (UE) can communicate with the network it must perform cell search and selection procedures and obtain initial system information. This involves acquiring slot and frame synchronization, finding out the cell identity and decoding the Master Information Block (MIB). The details of this procedure is explained in details in 'chapter 11'. The flow of the procedure done can be shown in 'figure 7.3'.

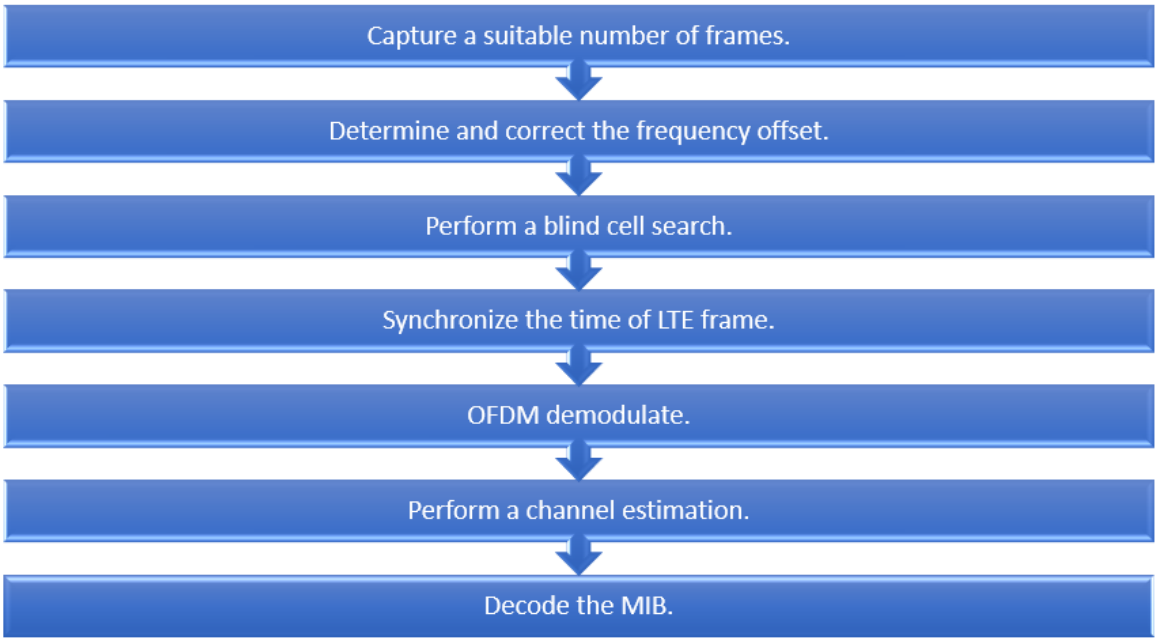


Figure 7-3 Cell search procedure flow

Each of these steps will be described in more details in the next chapters illustrating how they are done.

Prior to decoding the MIB, the UE does not know the full system bandwidth. The primary and secondary synchronization signals (PSS and SSS) and the PBCH (containing the MIB) all lie in the central 72 subcarriers (6 resource blocks) of the system bandwidth, allowing the UE to initially demodulate just this central region.

The MIB is carried on the Broadcast Channel (BCH) mapped into the Physical Broadcast Channel (PBCH). This is transmitted with a fixed coding and modulation scheme which is QPSK and can be decoded after the initial cell search procedure.

7.3.1 Simulation results

For generating R.2 reference channel signal as an example which properties is described in 'table 7.1' and with choosing FDD as a duplex technique, cyclic prefix length normal and cell ID is 283 as shown in figure 7.4

```
%% Load and Process I/Q Waveform

    rmc = lteRMCDL('R.2');

    rmc.NCellID = 283;
    rmc.CyclicPrefix = 'Normal';  % CP length
    rmc.DuplexMode = 'FDD';
    rmc.TotSubframes = 21;
    rmc.PHICHDuration = 'Extended'; % PHICH duration
```

Figure 7-4 generated signal parameters

After doing cell search procedure and the MIB recovery the results are

```
Command Window

Performing frequency offset estimation...
Frequency offset: -3.308Hz
Timing offset to frame start: 0 samples
Performing MIB decoding...
Cell-wide settings after MIB decoding:
        DuplexMode: 'FDD'
      CyclicPrefix: 'Normal'
             NDLRB: 50
            NCellID: 283
          NSubframe: 0
           CellRefP: 1
     PHICHDuration: 'Extended'
                Ng: 'Sixth'
            NFrame: 0

fx >>
```

Figure 7-5 Simulation results

And with plotting the frame time offset which will be known after performing PSS and SSS correlation we get the following result

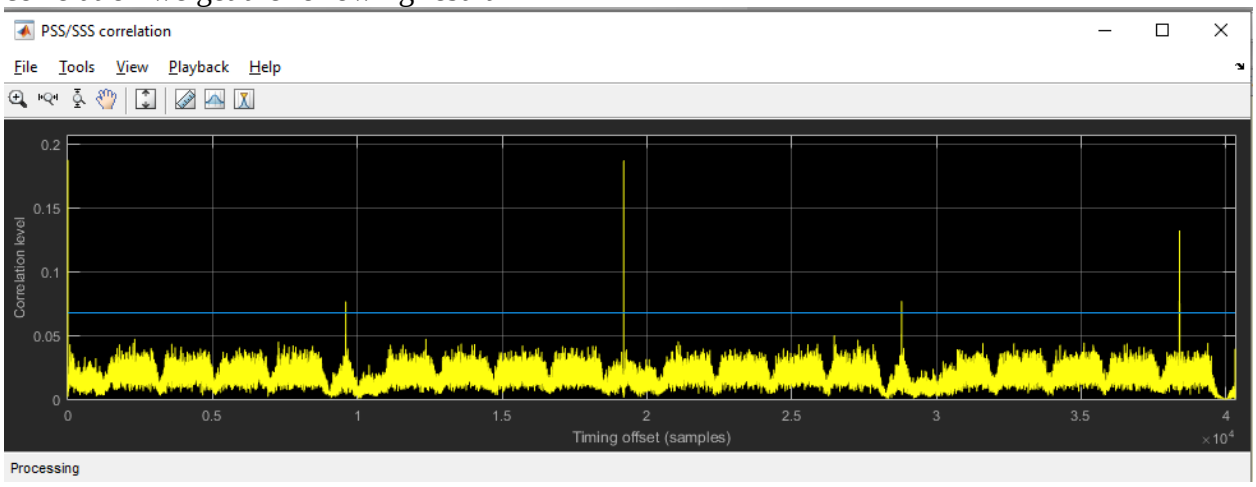


Figure 7-6 Timing offset position

Commenting on the results:

- The frame timing offset to frame start is 0 as shown in 'figure 7.6' because there was no delay in the received waveform as illustrated in ch.11
- From 'figure 7.4' the number of generated subframes is 21 so the timing offset was detected 3 times as in 'figure 7.7'

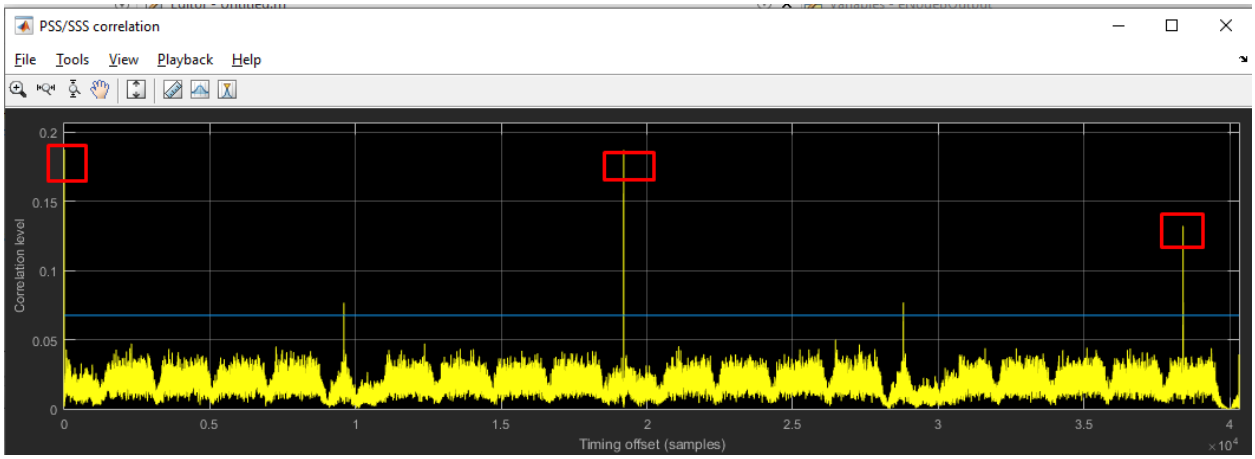


Figure 7-7 Start of frame position in case of 0 delay for 21 subframes

- If we delayed the received waveform as in 'figure 7.8' with some samples we will get the following result as in 'figure 7.9'

```
sampled_signal = [0; 0; 0; 0; 0; 0; 0; 0; 0; sampled_signal];
[cellID, offset] = lteCellSearch(enb, sampled_signal);
```

Figure 7-8delayed signal

```
Performing cell search...
Timing offset to frame start: 9 samples
Cell-wide settings after cell search:
      DuplexMode: 'FDD'
    CyclicPrefix: 'Normal'
            NDLRB: 6
          NCellID: 283
        NSubframe: 0
```

Figure 7-9 timing offset for delayed signal

7.4 CONCLUSION

The final code used for simulation used by can successfully perform cell search procedure and decode the physical broadcast channel which contains the master information block (MIB) for any LTE waveform for all different duplexing techniques (FDD/TDD) and for both cases of cyclic prefix lengths normal or extended.

This code can be implemented on the software define radio (SDR) to have an LTE Network scanner which will load the I-Q received waveform from real based cell's and from it we can deduce the cell system information and detect it's performance at different locations.

8 C700 LTE SCANNER

8.1 MAIN TARGET:.

For using C700 is how to recover MIB which is carried in Physical Broadcast Channel (PBCH) which is contained in central 6 resource blocks of the LTE downlink signal by implementing a communication system on hardware that will be responsible of main functions.

8.2 C700 OPERATION:.

```
% specify USB port index for C700:
sys=0;

% specify modules addresses:
RX = '401'; %VSD2 module address
LO = '115'; %PLL1 module address
TX = '20E'; %VSM1 module address

% specify waveform parameters:
fs=12.5e6; %fs can be 50 MHz, 25 MHz, 12.5 MH, and 6.25 MHz
N_samples=4096;
Fc=1e9; % carrier frequency 1 GHz
frqfst=1*15e6;
SRF=log2(50e6/fs); % Sampling rate factor

%Make system ready to start to initialize its modules
 USB([],'#',2); % import driver in case it is not imported
CloseUSB2(sys); % close USB ports
OpenUSB2(sys); % open USB port
C700_SYSTEM_OFF(sys); % turn system off, in case it was on
pause(.3);
C700_SYSTEM_ON(sys); % turn system ON
pause(1); % wait to boot
 % check connectivity of all modules
C700_CheckModules(sys,1,'000',LO,TX,RX);
```

- First, you must specify modules addresses as they are copied on modules because they are different from version to another.
- Then, specify waveform parameters just to match our limited hardware; here all modules work on fs=12.5MHz to capture BW=10MHz.

- Number of samples is just the capacity of these modules for this version only, for latest versions it is up to 16M samples instead of 4K.
- Then we define frequency offset =15M; is totally different from offset that we will calculate it, here we put it to avoid carrier feedthrough at center frequency; VSM and VSD internally contains 2 mixers and for practical cases these mixers are not identically matched and balanced so it causes leakage which independent on carrier frequency and we know that PLL module provides LO for all modules so for all that we define 15M as an offset.
- SRF, it just a parameter our modules need it to know sampling rate in c700 itself.
- SRF calculated as log2 (50e6/fs), so 50M here is the synchronization signal which the base unit provides to other modules, SRF ranges 0:3 here we work on SRF=2.
- Then make sure that driver is imported correctly and it at the same directory.
  - E:\NetworkScannerC700\C700_USB2_Driver\ftdi-usb-chip-driver

- Close USB2(sys); just to make sure the next line will not give you an error.
- The same for C700_SYSTEM_OFF(sys).
- Then check connectivity for all modules to the system; if an error occurred at that line make sure that you entered each module correctly into the base unit.

```
%% initialize the whole system:

% turn modules ON
PLL1_ON(sys,LO); % turn on LO
VSM1_ON(sys,TX); % turn on the Tx
VSD2_ON(sys,RX); % turn on the Rx
pause(0.3); % wait until modules power up.

% set carrier frequency
PLL1_SET_FREQ(sys,LO, Fc-frqfst);
% turn on RF output of Tx and LOs
PLL1_RF_ON(sys,LO);  % turn on RF of LO
VSM1_RF_ON(sys,TX);    % turn on RF of Tx
pause(0.3);
```

- Start to initialize the whole system by assign each module with its functionality which we already defined it.
- Turn VSM module it means that is correctly connected to the whole ‘sys’ and work as a transmitter.

- Set carrier frequency for all modules using PLL module, this line takes as a parameter (Fc-frqfst) that means carrier frequency 1G we defined it before and subtract frequency offset which caused by the mismatch between each 2 mixers in VSM and VSD modules.
- Then turn RF outputs ports of VSM and PLL.

8.3 GENERATED/CAPTURED WAVEFORM:.

- Now we are ready to use c700 and its modules, we have 3 options: Generated signal using LTE waveform Generator, Captured Signal, or connect directional antenna to c700 to capture a real signal around your area.
- Our code sequence follows that block diagram:

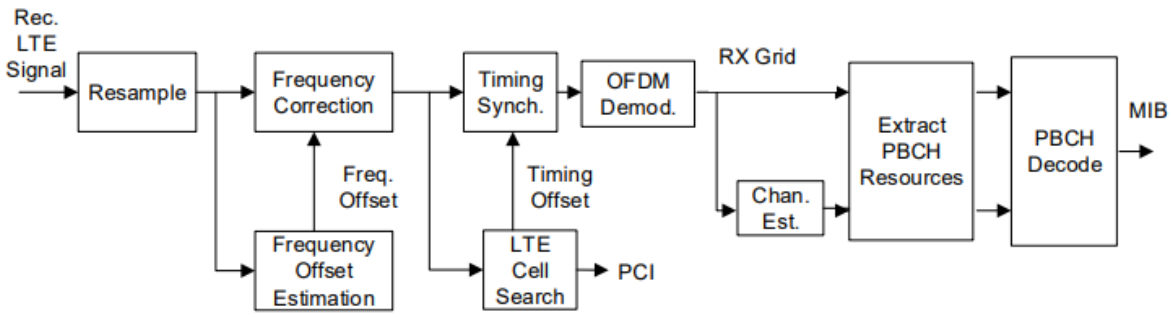


Figure 8-1Generated waveform is processed in MATLAB to obtain MIB bits

- we use lte waveform generator; to pass it through our modules:

```
IQ=eNodeBOutput/abs(max((eNodeBOutput)))*25000;


signal=real(IQ); % separate the real part
Qsignal=imag(IQ); % separate the imaginary part
```

- we scale the waveform, so its maximum is within the max amplitude of the VSM which is (+/-)32000, we choose 25000 not to be too close to the maximum.
- Then we separate the real and imaginary parts of the waveform so that VSM module take the waveform as a separated part; it just a syntax.

8.4 SEND WAVEFORM TO VSM:.

- Then send the waveform to TX to start the signal generation

```
VSM1_ARB3_SendWvaform_MM(sys,TX,signal,Qsignal,SRF,1,frqfst,32000,1);
 % send the waveform to the Tx and start the signal generation
pause(.5);
```

- Our sequence in this code that we generate a specific waveform with defined parameters including cyclic prefix and duplex mode types, but we compare at RX with the 4 modes (Extended/Normal FDD and TDD) that is to make our code be more general and ready to apply any waveform else or even capture signal using connected omnidirectional antenna to the C700.

8.5 VSD ACQUIRES THE SAMPLES:.

- We follow the previous diagram till OFDM demodulation step, we need VSD module to acquire the transmitted samples then get back [I Q]:
-

```
VSD2_CAP2_Configure_MM(sys, RX, N_samples,SRF,-frqfst);

    MM_Synced_Trgs(sys,{TX,RX});
    pause(N_samples/fs); % wait till the samples are captured.
    [I,Q] = VSD2_CAP2_GetWaveform_MM(sys,RX,N_samples,[],[],0);
```

- MM_Synced_Trgs; a function for triggering {TX,RX} to work at same time to prevent receiving undesired zeros at RX; here we use only one VSM and VSD, we can work on multiple ones then {TX1,TX2,RX1,RX2}.
- At VSM module, it calculate the number of samples using the length of signal and Qsignal, but at VSD we must configure the Rx and specify the number of samples it will capture.
- We repeat that sequence 4 times to detect each mode and finally make decoding for PBCH which carried MIB that contains info about system bandwidth, antenna configuration, system frame number.

8.6 SETUP OF C700:.

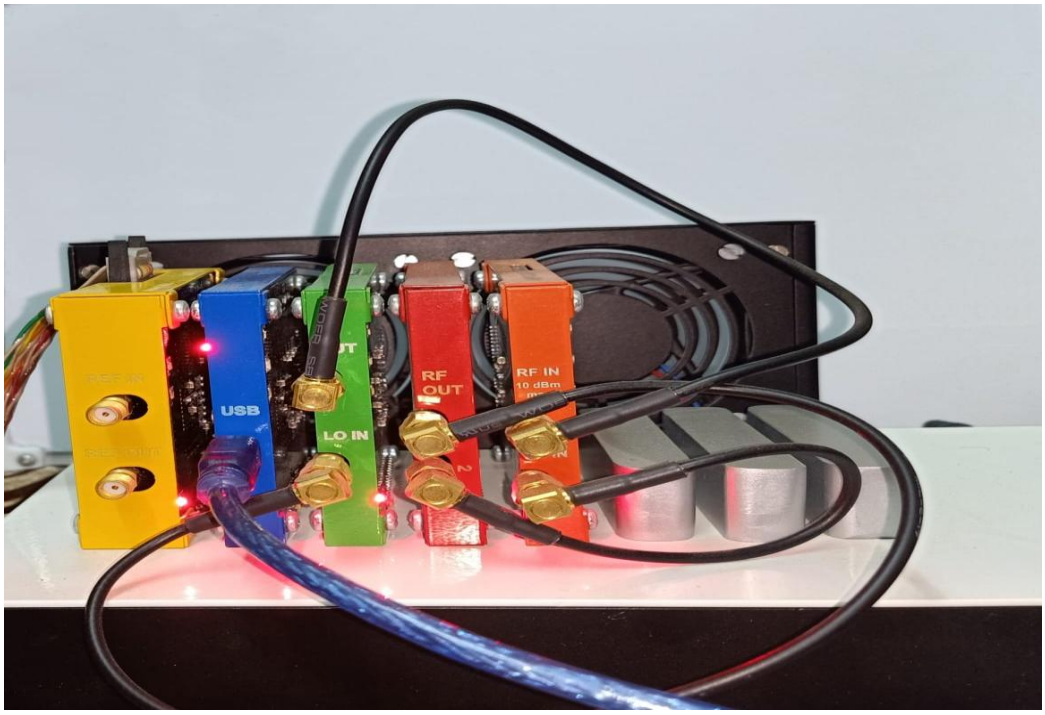


Figure 8-2 C700 Setup

8.7 Results:.

```
Command Window
>> scanner

Performing cell search...

Performing cell search...

Performing cell search...

Performing frequency offset estimation...
Frequency offset: -0.000Hz
Timing offset to frame start: 0 samples
Performing MIB decoding...
Cell-wide settings after MIB decoding:
        DuplexMode: 'TDD'
      CyclicPrefix: 'Normal'
             NDLRB: 6
            NCellID: 283
          NSubframe: 0
           CellRefP: 1
     PHICHDuration: 'Extended'
                Ng: 'Sixth'
            NFrame: 0
```

Figure 8-3Matlab results after decoding

```
%% Load and Process, I/Q Waveform

    rmc = lteRMCDL('R.4');
    rmc.SSC = 0;

    rmc.TDDConfig = 6;

    rmc.NCellID = 283;
    rmc.CyclicPrefix = 'Normal';
    rmc.DuplexMode = 'TDD';
    rmc.TotSubframes = 41;
    rmc.PHICHDuration = 'Extended';
```

- These results depend on “ignoring channel and AWGN” but the sequence of code is flexible to add channel and AWGN and then that result definitely will change.
- We get NDLRB=6; that MIB require 6 RBs (72 subcarriers) in Frequency domain
- Location in Resource Grid – 4 symbols of first subframe second slot symbol 0, 1, 2 and 3.

9 EVRYRANIA-RTL LTE SCANNER

In this chapter we will talk about how to make LTE Scanner by RTL-SDR dongle and Evrtania software

9.1 REQUIRED LIBRARIES :

- We need some libraries to run and compile Evrytania software:

  1- Boost library.

  2- ITPP library.

  3- Rtlsdr library.

  4- Fftw library.

9.1.1 Boost library :

- To install boost library easly you can use this command :

```
sudo apt-get install libboost-all-dev
```

9.1.2 ITPP library :

To install itpp library :

- first you want to download the source code from https://sourceforge.net/projects/itpp/files/ .
- the version that I download is itpp-4.3.1.tar.bz2 but I face a problem that does not compile on Ubuntu 18.04 LTS.

So I search and found solution for this error by using this commands

```
sudo apt-get install autoconf automake libtool build-essential cmake git \
 libblas-dev liblapack-dev libfftw3-dev
 cd ~/Downloads
 git clone https://git.code.sf.net/p/itpp/git itpp-git
 cd itpp-git
 ./autogen.sh
 ./configure
 mkdir build
 cd build
 cmake ..
 make
 sudo make install
```

- I have an error on the itpp library That error is due to a change in c++11. i adding "-std=c++03" to CXX_FLAGS in the CMakeLists.txt file in the application root directory.
  It should look something like this:

```
SET (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++
```

- But CMakeLists.txt file is for read only so I make me as a root user and give me permission on this file by the following commands

  - sudo -i
  - chmod -R 777 //directory

Note: you will replace directory with the location of file you want to have permission on it.

9.1.3 rtlsdr library :

- To install rtlsdr library you will need to run this commands:

```
sudo apt-get install cmake
sudo apt-get install git
sudo git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON -DDETACH_KERNEL_DRIVER=ON
make
sudo make install

sudo ldconfig
```

- I add this “ -DDETACH_KERNEL_DRIVER=ON ” in cmake ../ command as I face an error while I try to do cellsearch on evrytania this error is:

```
Kernel driver is active, or device is claimed by second instance of librtlsdr. In the first case,
please either detach or blacklist the kernel module (dvb_usb_rtl28xxu), or enable automatic
detaching at compile time.
```

9.1.4 fftw library :

- I install it when I install the missing libraries on Ubuntu 18.04 from this command

```
sudo apt-get -y install git swig cmake doxygen build-essential libboost-all-dev libtool libusb-1.0-0 libusb-1.0-0-dev libudev-dev libncurses5-dev libfftw3-bin libfftw3-dev libfftw3-doc libcppunit-1.14-0 libcppunit-dev libcppunit-doc ncurses-bin cpufrequtils python-numpy python-numpy-doc python-numpy-dbg python-scipy python-docutils qt4-bin-dbg qt4-default qt4-doc libqt4-dev libqt4-dev-bin python-qt4 python-qt4-dbg python-qt4-dev python-qt4-doc python-qt4-doc libqwt6abi1 libfftw3-bin libfftw3-dev libfftw3-doc ncurses-bin libncurses5 libncurses5-dev libncurses5-dbg libfontconfig1-dev libxrender-dev libpulse-dev swig g++ automake autoconf libtool python-dev libfftw3-dev libcppunit-dev libboost-all-dev libusb-dev libusb-1.0-0-dev fort77 libsdl1.2-dev python-wxgtk3.0 git libqt4-dev python-numpy ccache python-opengl libgsl-dev python-cheetah python-mako python-lxml doxygen qt4-default qt4-dev-tools libusb-1.0-0-dev libqwtplot3d-qt5-dev pyqt4-dev-tools python-qwt5-qt4 cmake git wget libxi-dev gtk2-engines-pixbuf r-base-dev python-tk liborc-0.4-0 liborc-0.4-dev libasound2-dev python-gtk2 libzmq3-dev libzmq5 python-requests python-sphinx libcomedi-dev python-zmq libqwt-dev libqwt6abi1 python-six libgps-dev libgps23 gpsd gpsd-clients python-gps python-setuptools
```

- I get this command from GNU Radio instullation guide from ettus research site Link: https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux#Install_Linux

9.2 EVRYTANIA DOWNLOAD SOURCE CODE AND BUILD INSTRUCTIONS :

9.2.1 Obtaining the code :

- Using git:

```
git clone git://github.com/Evrytania/LTE-Cell-Scanner.git
```

9.2.2 Compile :

```
cd LTE-Cell-Scanner
mkdir build
cd build
cmake ..
make
```

9.2.3 Install :

```
sudo make install
```

9.3 SETUP:

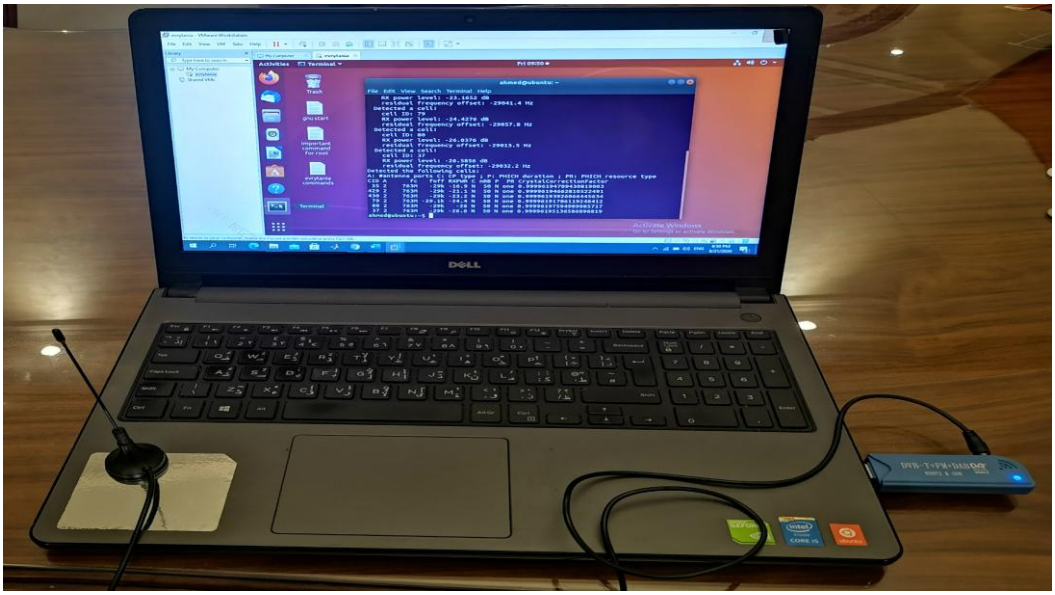


Figure 9-1RTL-SDR Setup

9.4 RUNNING CELLSEARCH TOOL:

1- we open the terminal to open Evrytania software to run cellsearch tool by running this command:

- If you don't know the frequency of the neighbor cells you can use the following command :

```
CellSearch --freq-start <frequency> --freq-end <frequency>
```

We will but the range of frequency we want to search on it.

- If you know the center frequency of the neighbor cell you can use the following command

```
CellSearch --freq-start <frequency>
```

9.5 RES

ULTS:

- In the project I do cell search at center frequency 763MHz and I found the following cells

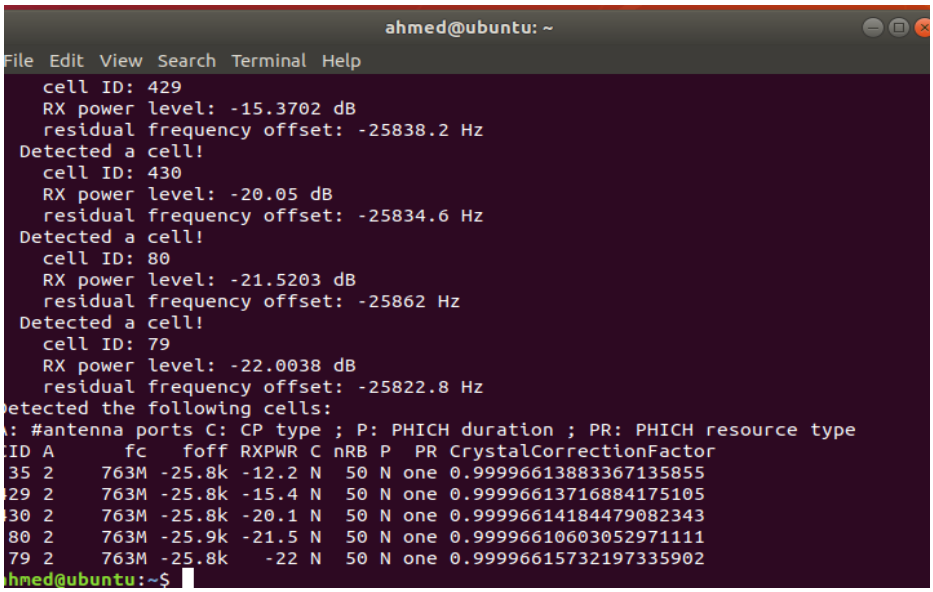


Figure 9-2 Evrytania results

- The cellsearch tool scane the cells that operates at this center frequency and collect the information of this cells and summaries them at the end.

```
Detected the following cells:
A: #antenna ports C: CP type ; P: PHICH duration ; PR: PHICH resource type
CID A      fc    foff RXPWR C nRB P  PR CrystalCorrectionFactor
35  2    763M -25.8k -12.2 N  50 N one 0.99996613883367135855
429 2    763M -25.8k -15.4 N  50 N one 0.99996613716884175105
430 2    763M -25.8k -20.1 N  50 N one 0.99996614184479082343
80  2    763M -25.9k -21.5 N  50 N one 0.99996610603052971111
79  2    763M -25.8k -22   N  50 N one 0.99996615732197335902
```

1. We can get from cellscanner:
   a) CID " cell id " 35
   b) A " number of transmit antennas of this cell " 2
   c) Fc " center frequency " 763M
   d) Foff " frequency offset " -25.8k
   e) RXPWR " received power " -12.2
   f) C " cyclic prefix " N
   g) nRB " how many this cell occupies from the resource block " 50
   h) P " what is the type of PHICH duration " N
   i) PR " what is the type of PHICH resource " one
   j) The last column in the program output is the crystal correction factor which provides information about the frequency error of the crystal in the dongle 0.999966138833671358

10 FREQUENCY SYNCHRONIZATION

10.1 CARRIER FREQUENCY OFFSET (CFO)

The baseband transmit signal is converted up to the passband by a carrier modulation and then converted down to the baseband at receiver by using a local carrier signal of the same carrier frequency at the transmitter. Well, that is not always the case, there may be a mismatch between transmitter and receiver local oscillators due to non-idealities of the manufacturing process, also a Doppler shift may be exist due to the relative motion between transmitter and receiver. The existence of any of the two previous main causes of CFO will result in inter-carrier interference (ICI) between different subcarriers which result in loss of orthogonality.

	Received signal	Effect of CFO ε on the received signal
Time-domain signal	$y[n]$	$e^{j2\pi n\varepsilon/N}x[n]$
Frequency-domain signal	$Y[k]$	$X[k-\varepsilon]$

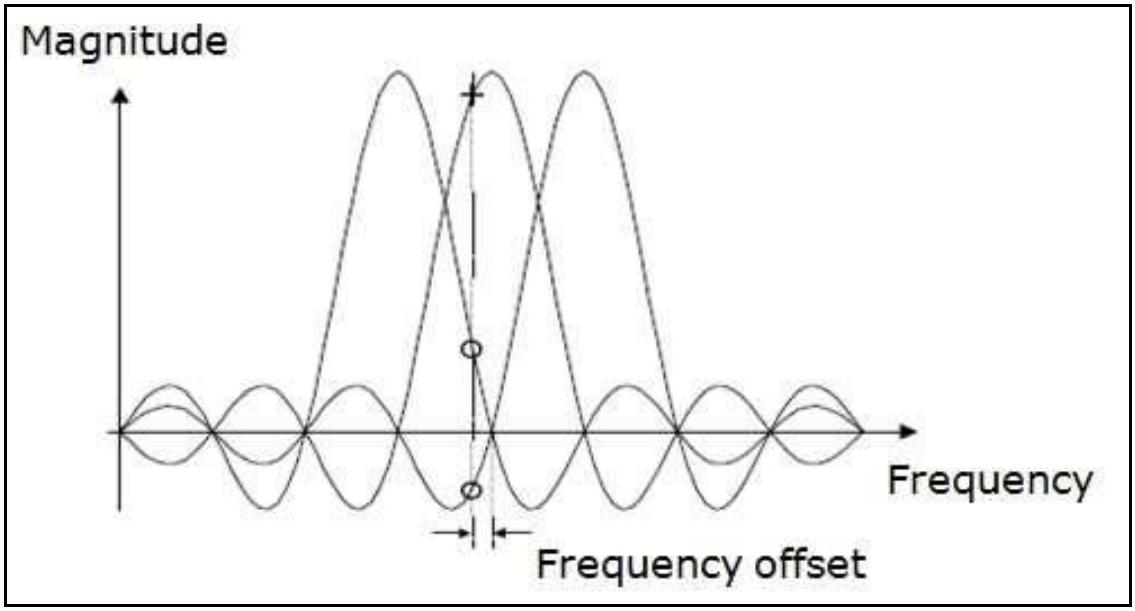


Figure 10-1The effect of CFO in frequency domain, Inter-carrier interference (ICI)

10.2 THE CAUSE FOR CARRIER FREQUENCY OFFSET ERROR

10.2.1 Mismatch between transmitter and receiver

We can develop a relationship for the frequency of a mismatch between transmitter and receiver offset by

$$f_{\text{offset,mismatch}} = f_{cT} - f_{cR} \qquad (1)$$

Where $f_{\text{offset,mismatch}}$ shows the frequency offset of a mismatch and f_{cT} &f_{cR} are the carrier frequencies at the transmitter and receiver end respectively.

10.2.2 The Doppler effect (or the Doppler shift)

The Doppler effect is observed whenever the source of waves is moving with respect to an observer. The Doppler effect can be described as the effect produced by a moving source of waves in which there is an apparent upward shift in frequency for observers towards whom the source is approaching and an apparent downward shift in frequency for observers from whom the source is receding.

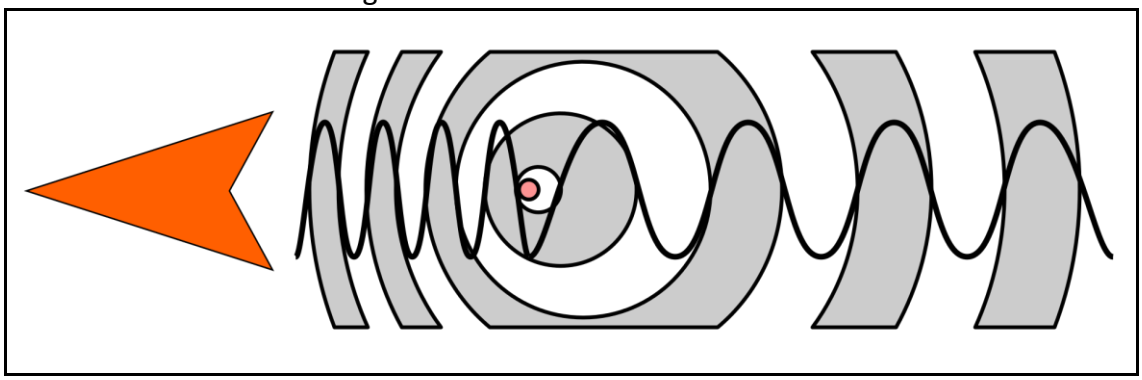


Figure 10-2 The Doppler effect of moving source

Changing velocity is associated with the Doppler shift. Thus, the relative velocity between transmitter and receiver dictates the frequency shift $f_{offset,d}$ is determined by the carrier frequency f_c and the velocity v of the receiver as:

$$\mathbf{f}_{\text{offset,d}} = \frac{\mathbf{v} \times \mathbf{f_c}}{\mathbf{c}} \qquad (2)$$

Where c is the speed of light, f_c is the carrier frequency.

$$f_{\text{offset}} = \mathbf{f}_{\text{offset,mismatch}} + \mathbf{f}_{\text{offset,d}} \qquad (3)$$

Where f_{offset} is the Carrier Frequency Offset (CFO).

Let us define the normalized CFO ε as a ratio of the CFO to subcarrier spacing Δf, as

$$\varepsilon = \frac{f_{\text{offset}}}{\Delta f} \qquad (4)$$

System	Carrier frequency (f_c)	Subcarrier spacing (Δf)	Velocity (v)	Maximum Doppler frequency (f_d)	Normalized CFO (ε)
DMB	375 MHz	1 kHz	120 km/h	41.67 Hz	0.042
3GPP	2 GHz	15 kHz	120 km/h	222.22 Hz	0.0148
Mobile WiMAX	2.3 GHz	9.765 kHz	120 km/h	255.55 Hz	0.0263

Table 10-1 Doppler frequency and normalized CFO: an example

10.3 EFFECTS OF CFO

Normalized CFO (ε) can be broken down into Fractional CFO (ε_f) which also known as FFO and into Integer CFO (ε_i) which usually known as IFO. Mathematically it can be written as:

$$\varepsilon = \varepsilon_i + \varepsilon_f \qquad (5)$$

10.3.1 Integer Frequency Offset (IFO)

The integer part of CFO ε_i, causes a cyclic shift in the receiver, and this result a significant degradation in the BER performance. However, integer CFO does not cause the loss of orthogonality among the subcarrier frequency components and thus, ICI does not occur.

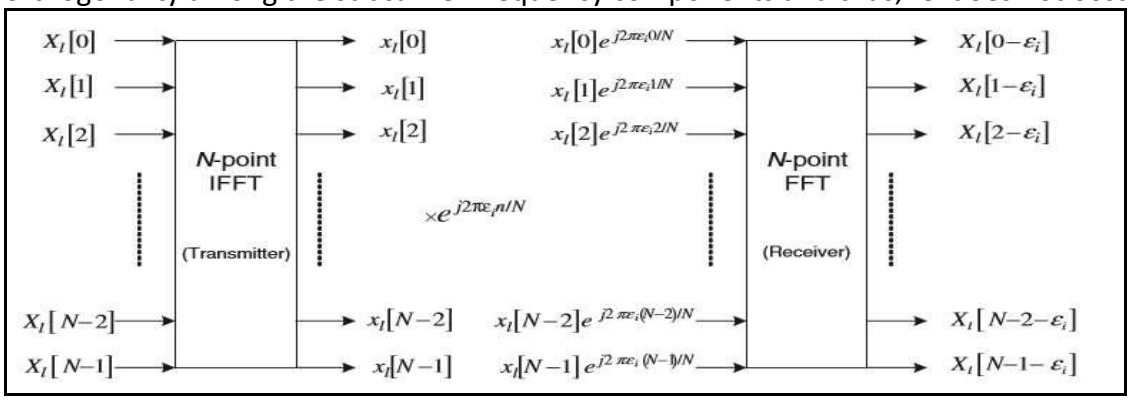


Figure 10-3 Effect of integer CFO on the received signal

10.3.2 Fractional Frequency Offset (FFO)

The fractional CFO is the part which destroys the orthogonality among the subcarriers and consequently introduces ICI. Let's now mathematically describe the effect of ε_f on the received OFDM signal. Ignoring the effects of the additive noise, the received signal before taking FFT can be expressed as

$$Y_1[n] = \frac{1}{N}\sum_{m=0}^{N-1} H[m]X_1[m]exp\frac{j2\pi n(m+\varepsilon_f)}{N} \qquad (6)$$

After taking FFT of the above equation, we will get the received signal which is affected by fractional CFO, and can be written as

$$Y_1[k] = \sum_{n=0}^{N-1} Y_1[n]exp\frac{-j2\pi kn}{N} \qquad (7)$$

Putting the value of $Y_1[n]$ in equation (7), we may write

$$Y_1[k] = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} H[m]X_1[m]exp\frac{j2\pi n(m+\varepsilon_f)}{N} exp\frac{-j2\pi kn}{N} \qquad (8)$$

$$Y_1[k] = \frac{1}{N}\sum_{m=0}^{N-1} H[m]X_1[m]\sum_{n=0}^{N-1} exp\frac{j2\pi(m-k+\varepsilon_f)}{N} \qquad (9)$$

Now, considering the case where k = m

$$Y_1[k] = \frac{1}{N} H[k]X_1[k]\sum_{n=0}^{N-1} exp\frac{j2\pi n\varepsilon_f}{N} + \frac{1}{N}\sum_{m=0}^{N-1}{}_{m\neq k} H[m]X_1[m]\sum_{n=0}^{N-1} exp\frac{j2\pi(m-k+\varepsilon_f)}{N}$$

(10)

After rearranging and simplifying the above equation, we have got

$$Y_1[k] = \frac{\sin \pi\varepsilon_f}{N \sin\frac{\pi\varepsilon_f}{N}} exp\frac{j\pi\varepsilon_f}{N} H[k]X_1[k] + I_1[k] \qquad (11)$$

Where

$$I_1[k] = exp\frac{j\pi\varepsilon_f(N-1)}{N} \sum_{m=0\ m\neq k}^{N-1} H[m]X_1[m] \frac{\sin \pi(m-k+\varepsilon_f)}{N \sin\frac{\pi(m-k+\varepsilon_f)}{N}} exp\frac{j\pi(m-k)(N-1)}{N} ($$

12)

The first term of equation (11), $\frac{\sin \pi\varepsilon_f}{N \sin\frac{\pi\varepsilon_f}{N}} exp\frac{j\pi\varepsilon_f}{N}$, represents the amplitude and phase distortion of the kth subcarrier frequency component due to the fractional CFO. Meanwhile, $I_1[k]$ in equation (11) represents the ICI from the other subcarriers into k^{th} frequency component, which implies that the orthogonality among subcarrier frequency components is not maintained any longer due to the fractional CFO.

2.3.2 SNR degradation due to CFO

According to [10], the degradation of SNR, D_{freq} , caused by the frequency offset is approximated as

$$D_{freq} \cong \frac{10}{3\ ln\ 10} (\pi\Delta f T)^2 \frac{E_b}{N_o} \qquad (13)$$

Where Δf is the frequency offset, T is the symbol duration in seconds, Eb is the energy per bit of the OFDM signal and No is the one-sided noise power spectrum density (PSD).

10.4 ESTIMATION TECHNIQUES FOR CFO

There are many techniques for estimated CFO we are discussed a two of these techniques as follows.

10.4.1 CP based Estimation Method

This method exploits the Cyclic Prefix (CP) of the OFDM symbol to estimate the CFO in time domain. Considering the channel effect is minimal and can be neglected, then the 1th OFDM symbol affected by CFO can be written as

$$Y_1[n] = x(n)exp\frac{j2\pi\varepsilon n}{N} \qquad (14)$$

We may now consider the corresponding CP in the OFDM symbol

$$Y_1[n+N] = x(n+N)exp\frac{j2\pi\varepsilon}{N}(n + N) \qquad (15)$$

$$Y_1[n+N] = x(n)exp\left[\frac{j2\pi\varepsilon n}{N} + j2\pi\varepsilon\right] \qquad (16)$$

From equations (15) and (16), we can find that the phase difference between CP and the OFDM symbol which the victim of CFO is $2\pi\varepsilon$. Therefore, the amount of CFO can be found by the multiplication of OFDM symbol (CFO affected) with the CP and after that taking their phase angle measurements, as show below

$$\tilde{\varepsilon} = \frac{1}{2\pi} arg\{y_1^*[n]y_1[n + N]\}, n = -1, -2, \cdots - N_g \qquad (17)$$

In order to reduce the noise effect, its average can be taken over the samples in a CP interval as:

$$\tilde{\varepsilon} = \frac{1}{2\pi} arg\left\{\sum_{n=-N_g}^{-1} y_1^*[n] y_1[n+N]\right\}, n = -1, -2, \cdots - N_g \quad (18)$$

Since **arg** () function if performed by using $\tan^{-1}()$ function which has the range of [−π, π] in an 2π interval, the above equation can estimate CFO in the range of [-0.5, +0.5]. Therefore, CP results CFO estimation in the range |ε| ≤ 0.5. Hence, this technique is useful for the estimation of Fractional CFO. The drawback of this technique is that it does not estimate the integer offset.

10.4.2 Symbol based Estimation Method

The basis of this technique, proposed by P.H. Moose [8], is that same data frame is repeated and the phase value of each carrier between consecutive symbols is compared.

If two identical training symbols are transmitted consecutively, the corresponding signals with CFO of ε are related with each other as follows

$$y_2[n] = y_1[n] \exp\frac{j2\pi\varepsilon N}{N} \leftrightarrow Y_2[k] = Y_1[k] \exp j2\pi\varepsilon \quad (19)$$

Using the relationship in the above equation, the CFO can be estimated as

$$\tilde{\varepsilon} = \frac{1}{2\pi}\tan^{-1}\left\{\frac{\sum_{k=0}^{N-1} \mathrm{Im}\,[Y_1^*[k]Y_2[k]]}{\sum_{k=0}^{N-1} \mathrm{Re}\,[Y_1^*[k]Y_2[k]]}\right\}$$

(20)

The range of CFO estimation of this method is also limited to $|\tilde{\varepsilon}| \leq \pi/2\pi = 1/2$. However, the range can be increased by D times by using a training symbol with D repetitive patterns.

10.4.3 Pilot Tone based Estimation Method

This method, proposed by Classen [9], suggests the use of pilot tones to estimate CFO in the frequency domain. Pilot tones can be inserted in the frequency domain and transmitted in every OFDM symbol for CFO tracking. First, two symbols, $y_l[n]$ and $y_{l+D}[n]$, are saved in the memory after synchronization. Then, the signals are transformed into $\{Y_l[k]\}_{k=0}^{N-1}$ and $\{Y_{l+D}[k]\}_{k=0}^{N-1}$ via FFT, from which pilot tones are extracted.

In this process, two different estimation modes for CFO estimation are implemented: acquisition and tracking modes. In the acquisition mode, a large range of CFO including an integer CFO is estimated. In the tracking mode, only fine CFO is estimated. The integer CFO is estimated by

$$\widetilde{\varepsilon_{acq}} = \frac{1}{2\pi T_{sub}} \max\left\{\left|\sum_{j=0}^{L-1} Y_{l+D}[P[j],\varepsilon] Y_l^*[P[j],\varepsilon] X_{l+D}^*[P[j]] X_l[P[j]]\right|\right\} \quad (21)$$

where L, P[j] and $X_l[P[j]]$ denote the number of pilot tones, the location of the jth pilot tone and the pilot tone located at P[j] in the frequency domain at the l^{th}symbol period, respectively. Meanwhile, the fine CFO is estimated by

$$\tilde{\varepsilon}_f = \frac{1}{2\pi Tsub} \max\left\{\left|\sum_{j=0}^{L-1} Y_{l+D}[P[j],\widetilde{\varepsilon_{acq}}] Y_l^*[P[j],\widetilde{\varepsilon_{acq}}] X_{l+D}^*[P[j]] X_l[P[j]]\right|\right\} \quad (22)$$

10.5 SIMULATION RESULTS

In the Figure (*10.4*) a plot of constellation diagram of QPSK modulated symbols for different values of Normalized CFO(FFO) using MATLAB simulations. First graph with CFO = 0, the

4 other graphs CFO = 0.001, 0.01,0.1, 0.3, as the CFO increase the amplitude and phase distortion becomes severe which is the contribution of ICI.

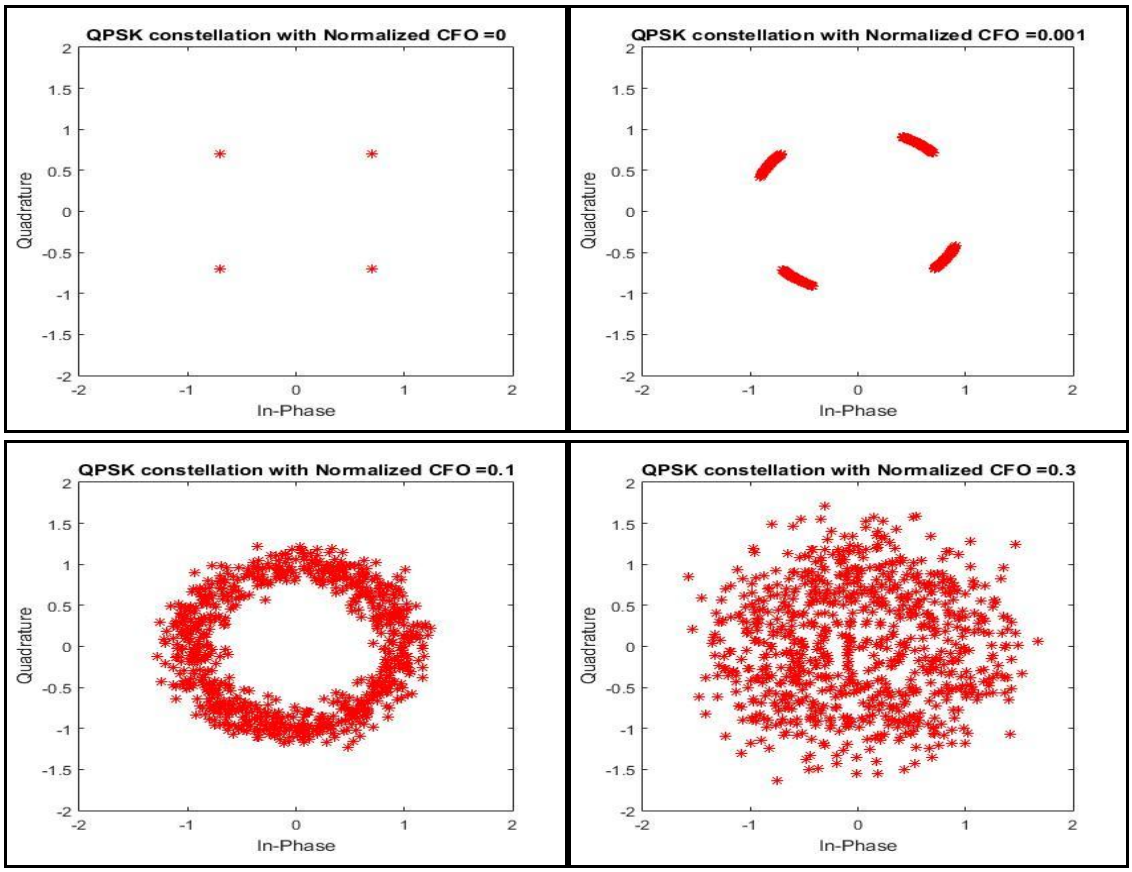


Figure 10-4 Constellation of received symbols with different CFO(FFO)

In the Figure (*10.5*) indicates that the SNR deterioration is bigger in higher SNR values signals.

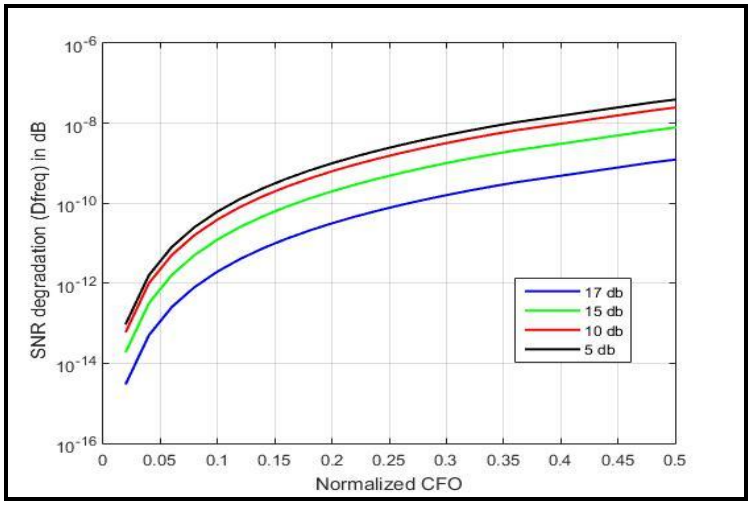


Figure 10-5 SNR degradation due to CFO

In the Figure (*10.6*) a plot of BER sensitivity of PSK modulated symbols for different values of Normalized CFO(FFO) using MATLAB simulations.

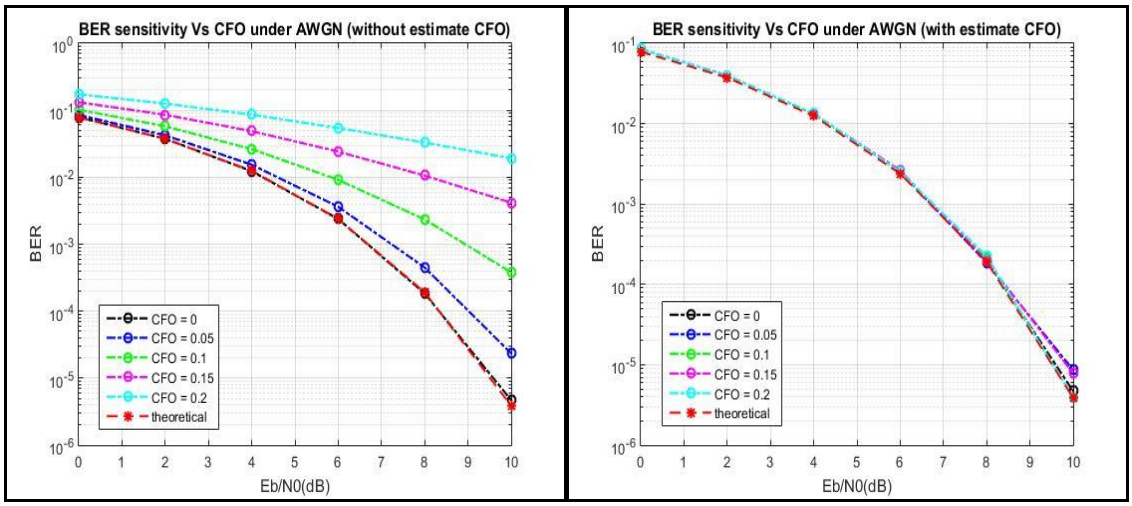


Figure 10-6 Bit Error Rate vs Eb/N0 (dB) with/without estimation

In the Figure (*10.7*) a plot of BER Vs Eb/N0(dB) of QPSK modulated symbols for different methods of Normalized CFO estimations using MATLAB simulations.

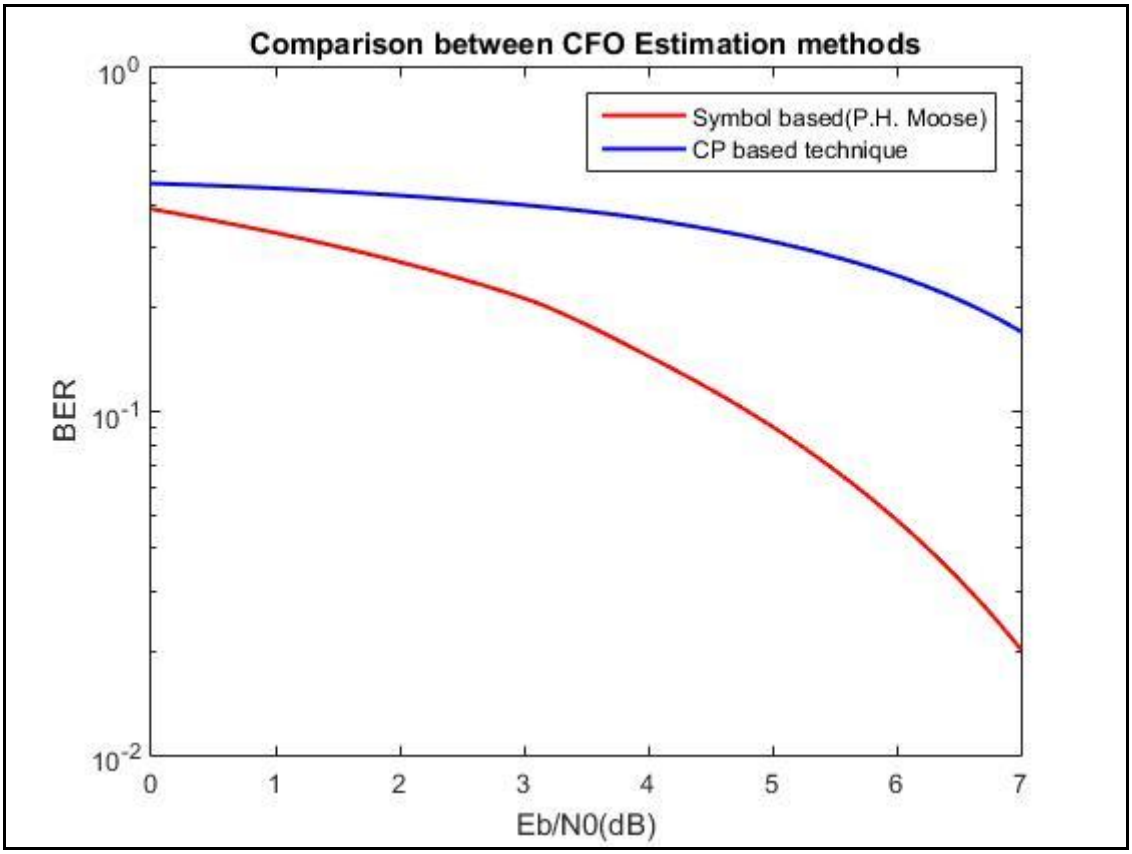


Figure 10-7 Bit Error Rate vs Eb/N0 (dB) with estimation methods

11 TIME SYNCHRONIZATION AND CELL ID DETECTION

11.1 CELL SEARCH

A cell search is the procedure by which the terminal finds a cell for a potential connection to. AS a part of the cell search procedure, the terminal obtains the physical identity of the cell (Cell ID) and estimates the frame timing of the identified cell. Furthermore, the cell search procedure also provides estimates of parameters essential for reception of system information on the broadcast channel containing the remaining parameters required for accessing the system.

To avoid complicated cell planning, the available number of physical layer cell identities is 504, these identities are divided into 168-layer identity group in which each group consists of 3 physical layer identities.

In order to reduce cell search complexity, cell search for LTE is done in several steps. To assist these steps LTE provides 2 synchronization signals

- Primary synchronization signal (PSS)
- Secondary synchronization signal (SSS)

The two signals, carrying the physical layer identity **(N_2^{ID})** , and the physical layer cell identity group (N_1^{ID}) , are the primary and the secondary synchronization signals respectively. This means that the complete cell search procedure consists of two steps to identify the cells' identity.

In cell search procedures in LTE, the UE uses two special signals broadcast on each cell: Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS). The detection of these signals allows the UE to complete time synchronization and to acquire useful system parameters such as cell identity, cyclic prefix length, and access mode (FDD/TDD). At this stage, the UE can also decode the Master Information Block from the Physical Broadcast Control Channel (PBCH) and obtain important system information.

Synchronization signals are transmitted twice per 10 ms radio frame with periodicity 5 ms between each, the position of each depends on the Access mode used (FDD/TDD) and the length of the cyclic prefix and will be described in the next points.

Through the PSS the receiver can achieve time slot synchronization and after that through the SSS can achieve the frame synchronization.

11.1.1 step I- Primary synchronization signals

The receiving terminal first looks for the PSS which is transmitted in the Last OFDM symbol of the first and 11th time slot each radio subframe (symbol 7 in case of normal CP & symbol 6 in case of extended CP) for Frame type 1 as shown in 'figure 11.1'. PSS allows the UE to acquire the slot boundary timing independent of the type of cyclic prefix length. The PSS signal is the same for any given cell in every subframe.

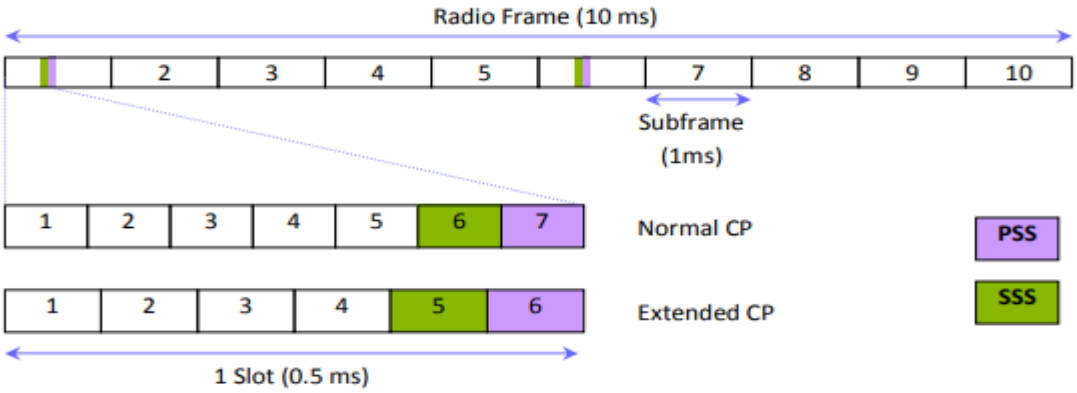


Figure 11-1 PSS in case of different cyclic prefix lengths for frame structure 1 (FDD)

For frame structure type 2, the primary synchronization signal shall be mapped to the third OFDM symbol in subframes 1 and 6.

In the frequency domain, six Resource Blocks (RB) around the DC subcarrier are reserved for transmission of the synchronization signals. In the frequency domain, an RB is formed by 12 subcarriers. With a subcarrier spacing of 15 kHz a bandwidth of 180 kHz (12*15 kHz) is occupied, reserving a frequency range of 1.08 MHz (6*180 kHz) around the center frequency for transmission of synchronization signals (that is, 72 subcarriers). This is independent from the defined channel bandwidth that is configured for the cell. This leaves 5 sub-carriers at each extremity of the 6 central RBs unused. This is also applied on the Secondary Synchronization signal (SSS) which will be illustrated next.

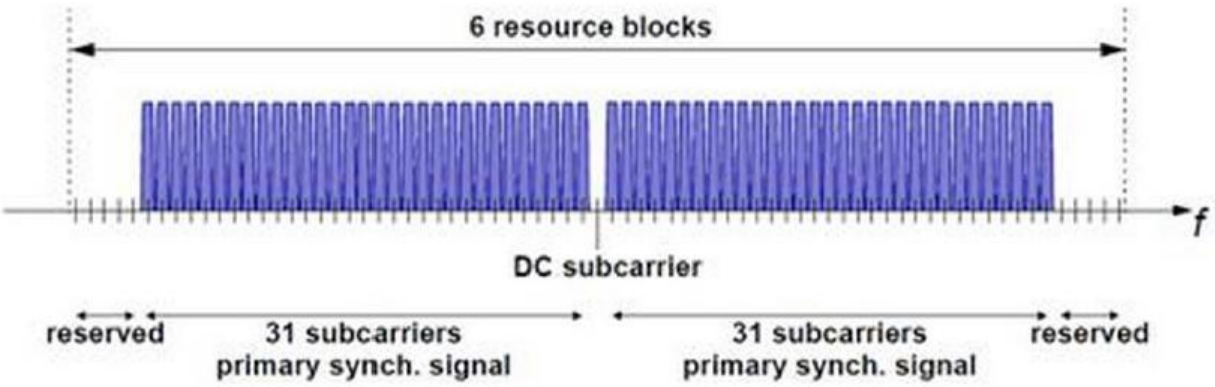


Figure 11-2 PSS in frequency domain

The type of signal used for primary synchronization is a Zadoff- Chu (ZC) sequence. ZC sequences are ‘’CAZAC’’ sequences, which stands for Constant Amplitude Zero Auto Correlation and describes the characteristic of this type of sequences. With a constant amplitude, a low peak-to-average power ratio is achieved, where zero auto correlation equates with good time domain behavior.

11.1.1.1 PSS sequence generation

The sequence d(n) used for the primary synchronization signal is generated from a frequency-domain Zadoff-Chu sequence according to

$$d_u(n) = \begin{cases} e^{-j\frac{\pi un(n+1)}{63}} & n = 0,1,\ldots,30 \\ e^{-j\frac{\pi u(n+1)(n+2)}{63}} & n = 31,32,\ldots,61 \end{cases}$$

where the Zadoff-Chu root sequence index u is given by Table 11.1

$N_{ID}^{(2)}$	Root index u
0	25
1	29
2	34

Table 11-1 Root indices for the PSS

By generating these sequences in MATLab we can observe that it has constant amplitude and only the phase changes

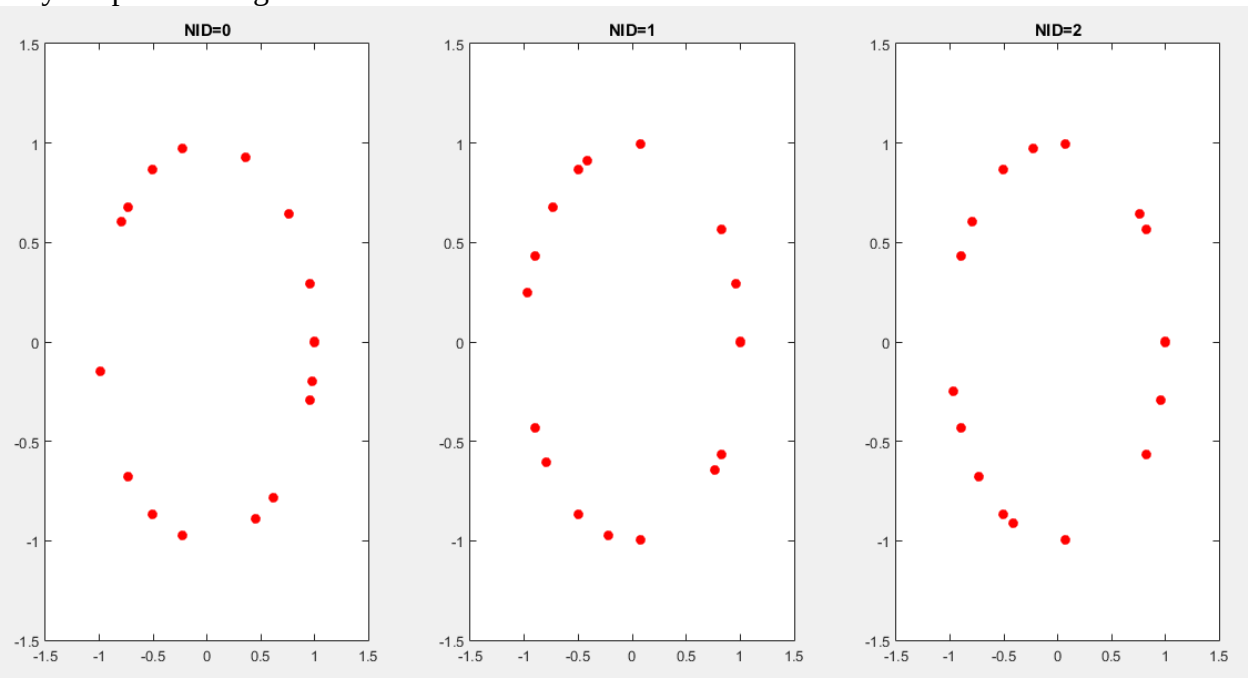


Figure 11-3 PSS generation by MATLab

Correlation between different PSS sequences:

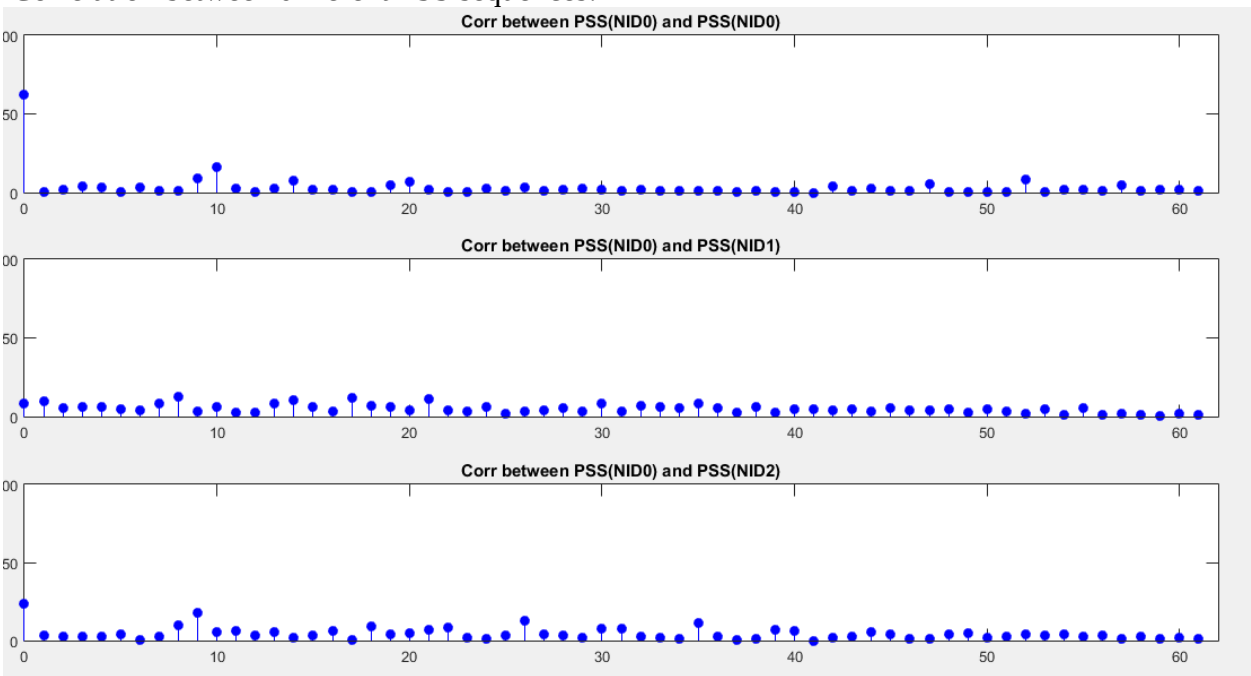


Figure 11-4 Correlation between different PSS sequences

Auto correlation between same PSS sequences shows that it is almost zero except when shift=0 samples

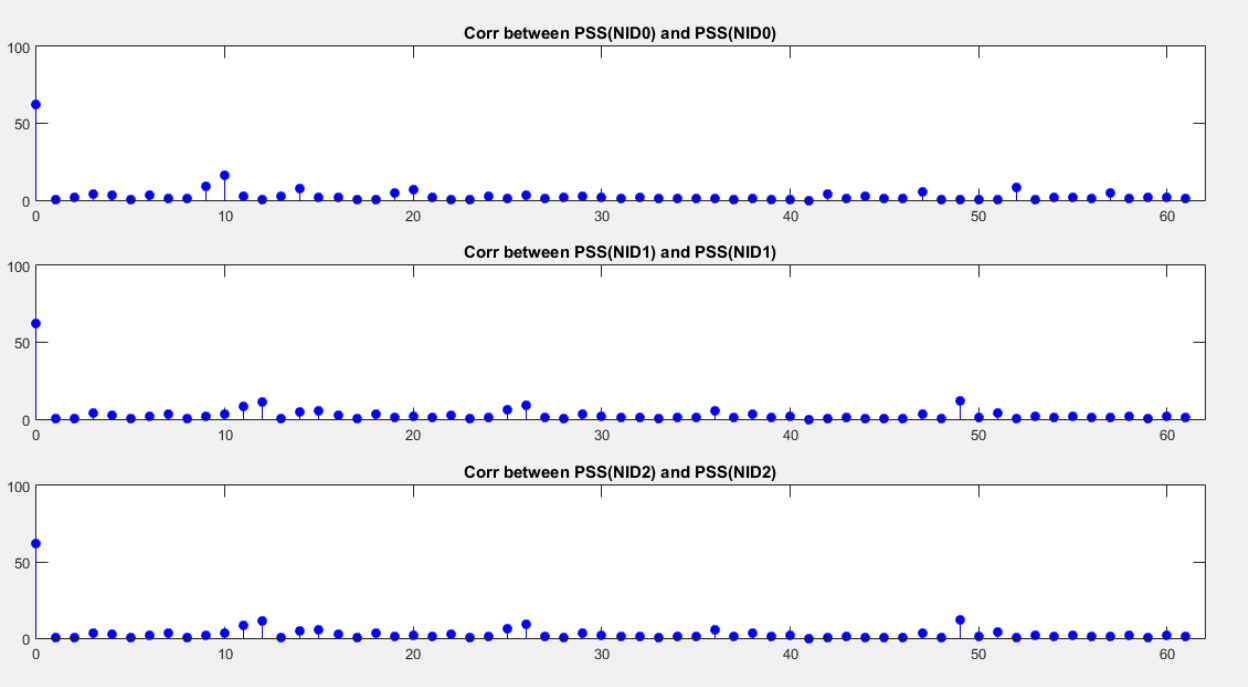


Figure 11-5 Auto correlation for pss sequences

11.1.1.2 PSS Mapping to resource element

The sequence d(n) shall be mapped to the resource elements according to

$$a_{k,l} = d(n), \qquad n = 0,\ldots,61$$

$$k = n - 31 + \frac{N_{\mathrm{RB}}^{\mathrm{DL}} N_{\mathrm{sc}}^{\mathrm{RB}}}{2}$$

- K is the freq. domain index and l is the time domain index
- N_{SC}^{RB} is the Resource block size in freq. domain, expressed as number of subcarriers.
- N_{RB}^{DL} is the Downlink bandwidth configuration expressed ass multiples of N_{SC}^{RB}

11.1.2 Step II- Secondary synchronization signal (SSS)

The SSS is transmitted in the symbol before the PSS. The SSS also has 5 ms periodicity, which means it is transmitted in the first and sixth subframes as shown in 'Figure 11.1'. Like the PSS, the SSS is transmitted on 62 of the 72 reserved subcarriers around the DC subcarrier.

For frame structure type 2, the secondary synchronization signal shall be mapped to the second OFDM symbol in subframes 1 and 6 (the symbols before the PSS).

11.1.2.1 SSS Sequence generation

The combination of two length-31 sequences defining the secondary synchronization signal differs between subframes according to

$$d(2n) = \begin{cases} s_0^{(m_0)}(n)c_0(n) & \text{in subframes } 0,1,2,3,4 \\ s_1^{(m_1)}(n)c_0(n) & \text{in subframes } 5,6,7,8,9 \end{cases}$$

$$d(2n+1) = \begin{cases} s_1^{(m_1)}(n)c_1(n)z_1^{(m_0)}(n) & \text{in subframes } 0,1,2,3,4 \\ s_0^{(m_0)}(n)c_1(n)z_1^{(m_1)}(n) & \text{in subframes } 5,6,7,8,9 \end{cases}$$

The sequence used for the second synchronization signal is an interleaved concatenation of two length-31 binary sequences. The concatenated sequence is scrambled with a scrambling sequence given by the primary synchronization signal.
It is always a pair of sequences ($S_0^{(m0)}$, $S_1^{(m1)}$) which is transmitted in a subframe. Sequence $S_0^{(m0)}$ is mapped in the case of subframe 0 to the even numbered subcarriers ($d(2n)$), whereas sequence $S_1^{(m1)}$ is mapped to odd numbered subcarriers ($d(2n+1)$). For subframe 5 it is the other way around. The following figure shows the SSS signal mapping for subframe 0.

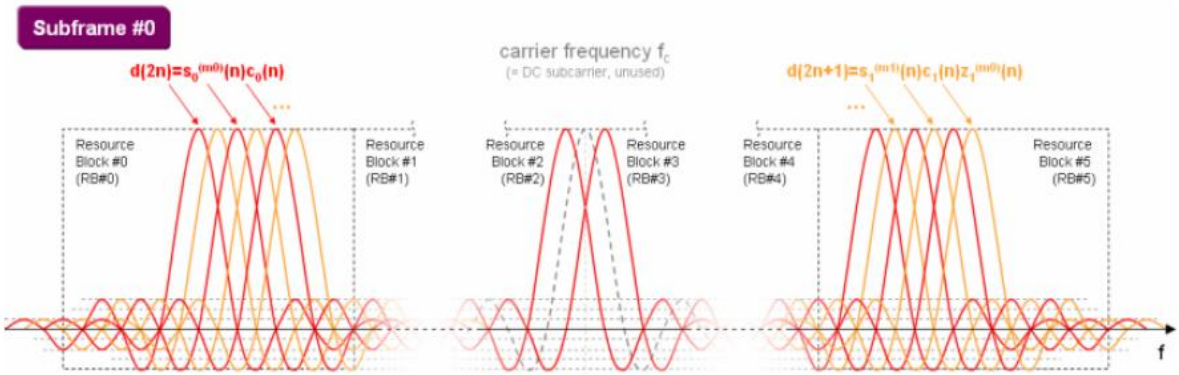


Figure 11-6 SSS signal mapping in subframe 0

The combination of the indices m_0 and m_1 defines the physical layer cell identity group $N_{ID}^{(1)}$, and the possible combinations of m_0 and m_1 *for the 168 groups are defined in 'table 11.2'*

Depending on the subcarrier – even or odd – another scrambling sequence is used for sequences $S_0^{(m0)}$, $S_1^{(m1)}$ which are $c_0(n)$ or $c_1(n)$ transmitted, an additional scrambling

sequence is used. For subframe 0 it is $z_1^{(m0)}(n)$ and for subframe 5 it is $z_1^{(m1)}(n)$ is used. Each of these two scrambling sequences depends on the indices m_0 and m_1.

Scrambling with an additional scrambling sequence optimizes cell search at the cell edge. Here the UE receives signals of several eNBs. At this point differentiation is required, which is achieved using a cell-identity-dependent scrambling sequence such as $z_1^{(m0)}(n)$ and $z_1^{(m1)}(n)$

$N_{ID}^{(1)}$	m_0	m_1	$N_{ID}^{(1)}$	m_0	m_1	$N_{ID}^{(1)}$	m_0	m_1	$N_{ID}^{(1)}$	m_0	m_1	$N_{ID}^{(1)}$	m_0	m_1
0	0	1	34	4	6	68	9	12	102	15	19	136	22	27
1	1	2	35	5	7	69	10	13	103	16	20	137	23	28
2	2	3	36	6	8	70	11	14	104	17	21	138	24	29
3	3	4	37	7	9	71	12	15	105	18	22	139	25	30
4	4	5	38	8	10	72	13	16	106	19	23	140	0	6
5	5	6	39	9	11	73	14	17	107	20	24	141	1	7
6	6	7	40	10	12	74	15	18	108	21	25	142	2	8
7	7	8	41	11	13	75	16	19	109	22	26	143	3	9
8	8	9	42	12	14	76	17	20	110	23	27	144	4	10
9	9	10	43	13	15	77	18	21	111	24	28	145	5	11
10	10	11	44	14	16	78	19	22	112	25	29	146	6	12
11	11	12	45	15	17	79	20	23	113	26	30	147	7	13
12	12	13	46	16	18	80	21	24	114	0	5	148	8	14
13	13	14	47	17	19	81	22	25	115	1	6	149	9	15
14	14	15	48	18	20	82	23	26	116	2	7	150	10	16
15	15	16	49	19	21	83	24	27	117	3	8	151	11	17
16	16	17	50	20	22	84	25	28	118	4	9	152	12	18
17	17	18	51	21	23	85	26	29	119	5	10	153	13	19
18	18	19	52	22	24	86	27	30	120	6	11	154	14	20
19	19	20	53	23	25	87	0	4	121	7	12	155	15	21
20	20	21	54	24	26	88	1	5	122	8	13	156	16	22
21	21	22	55	25	27	89	2	6	123	9	14	157	17	23
22	22	23	56	26	28	90	3	7	124	10	15	158	18	24
23	23	24	57	27	29	91	4	8	125	11	16	159	19	25
24	24	25	58	28	30	92	5	9	126	12	17	160	20	26
25	25	26	59	0	3	93	6	10	127	13	18	161	21	27
26	26	27	60	1	4	94	7	11	128	14	19	162	22	28
27	27	28	61	2	5	95	8	12	129	15	20	163	23	29
28	28	29	62	3	6	96	9	13	130	16	21	164	24	30
29	29	30	63	4	7	97	10	14	131	17	22	165	0	7
30	0	2	64	5	8	98	11	15	132	18	23	166	1	8
31	1	3	65	6	9	99	12	16	133	19	24	167	2	9
32	2	4	66	7	10	100	13	17	134	20	25	-	-	-
33	3	5	67	8	11	101	14	18	135	21	26	-	-	-

Table 11-2 mapping between NID (1) and indices m0 & m1

More details on the scrambling sequences equations are mentioned in 3GPP TS 36.211 V8.7.0; Physical Channels and Modulation (Release 8) in section 6.11.2.1

Generating SSS sequences with MATlab

```
Command Window
d_sub5 =

  Columns 1 through 22

     1    -1     1     1    -1     1     1     1    -1     1     1    -1    -1     1    -1     1    -1     1    -1    -1    -1     1

  Columns 23 through 44

     1     1    -1    -1    -1     1    -1    -1    -1    -1    -1     1     1    -1    -1    -1     1     1    -1     1     1    -1

  Columns 45 through 62

     1     1    -1    -1     1     1     1     1    -1     1    -1    -1     1     1     1    -1    -1     1

fx >>
```

Figure 11-7 SSS Sequence for NID(1)=120 , NID(2)=1 IN Subframe 5

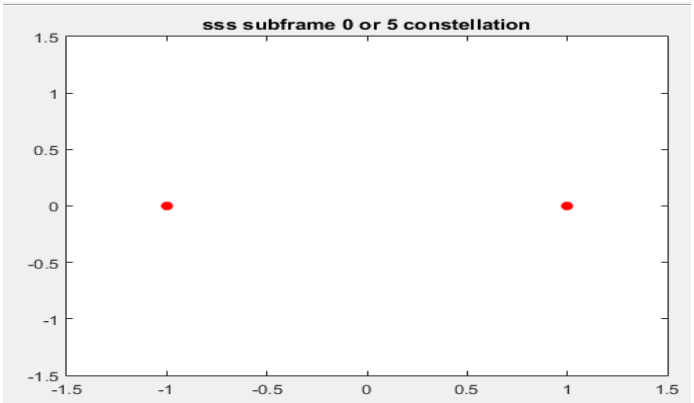


Figure 11-8 constellation diagram for SSS

And by performing the autocorrelation and cross correlation we found that similarly to PSS their autocorrelation is very low except when there is zero shift in samples while cross correlation is almost zero between different samples.

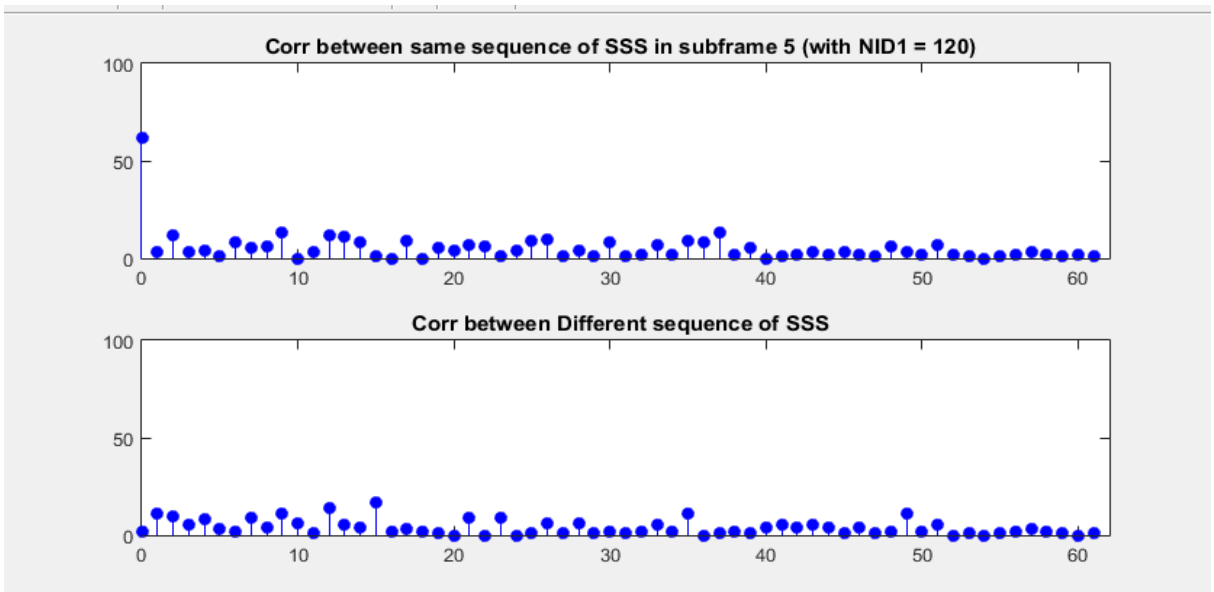


Figure 11-9 cross and auto correlation for SSS sequences

11.1.3 Timing synchronization and Cell ID

11.1.3.1 time slot synchronization

In the first step of the cell search procedure, the receiver terminal uses the primary synchronization signal to find the timing on a 5 ms basis as the PSS is transmitted twice in each frame. Thus the primary synchronization signal can provide the frame timing synchronization with a 5ms ambiguity.
The implementation of the estimation algorithm is vendor specific but the approach we used in this project was by correlating the received waveforms with the possible 3 sequences for the primary synchronization signals (PSS) then finding the max one of them, This procedure is not executed on the received analog RF signal, but rather in the digital domain. and from the maximum correlated sequence we can find the timing of the 5 ms basis and though the time slot synchronization. Also we acquire the physical layer identity **(N_2^{ID})**

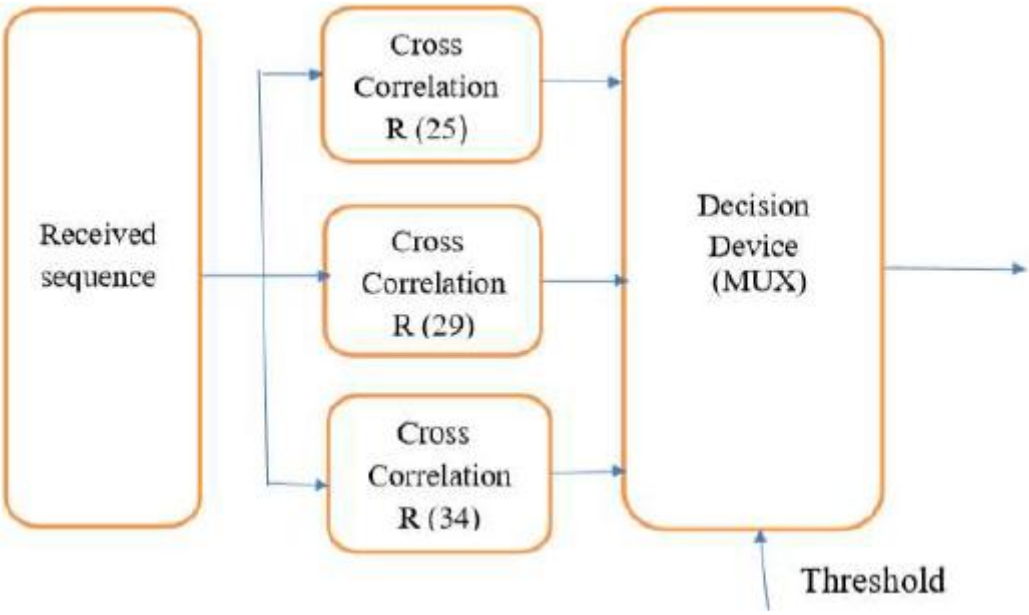


Figure 11-10 PSS detection using correlator

11.1.3.1 Frame synchronization

After the receiver finds the 5 ms timing, the second step is to find the cell identity group and determines the frame timing. This information can find from the secondary synchronization signals (SSS), this is done by observing pair of slots where the secondary synchronization signal is transmitted. From the property of $S_0^{(m0)}$ and $S_1^{(m1)}$ which states that basically if (S1,S2) is an allowable pair of sequences, where s1 and s2 represent the secondary synchronization signal in subframe 0 & 5 respectively, the reverse pair (S2,S1) is not a valid sequence pair. By exploiting this property, the terminal can resolve the 5 ms timing ambiguity resulting from the first step in the cell search procedure and determine the frame timing.

As each combination (S1,S2) represents one of the cell identity groups (N_1^{ID}) , also cell identity group is obtained from second cell search steps.

11.1.3.3 Physical layer Cell Identity (Cell ID)

After performing the first 2 steps in the cell search procedure the receiver obtained the physical layer identity (N_2^{ID}) , and the physical layer cell identity group (N_1^{ID}) and using these values physical layer cell identity (Cell ID) can be calculated by

$$N_{ID}^{cell} = 3 N_1^{ID} + N_2^{ID}$$

11.1.4 Differences between FDD & TDD

In case of cell search for LTE TDD , the position of the synchronization signals is different than in LTE FDD . The primary synchronization signal is always transmitted in the second subframe in the third OFDM symbol . As the repetition rate for synchronization signals is in TD-LTE the same as for LTD FDD (i.e., 5 ms), the PSS is again transmitted in subframe 6, which is a special subframe with Downlink Pilot Time Slot (DwPTS), Guard Period and Uplink Pilot Time Slot (UpPTS), or a subframe directly assigned for downlink transmission. The secondary synchronization is instead transmitted in the first subframe in time slot 1, OFDM symbol 13 when a normal cyclic prefix is used.

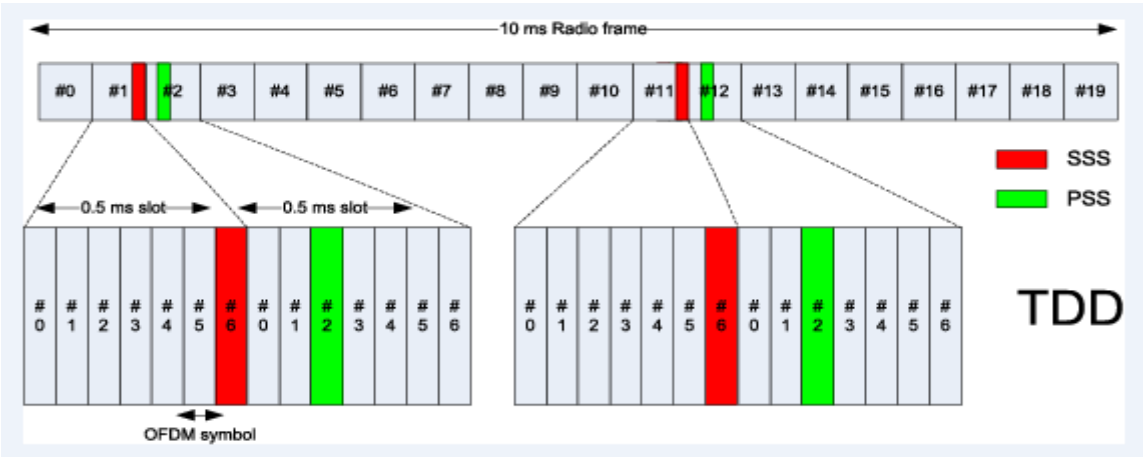


Figure 11-11 primary/secondary synchronization signals for LTE TDD

11.1.4.1 Differences between FDD & TDD for different Cyclic prefix

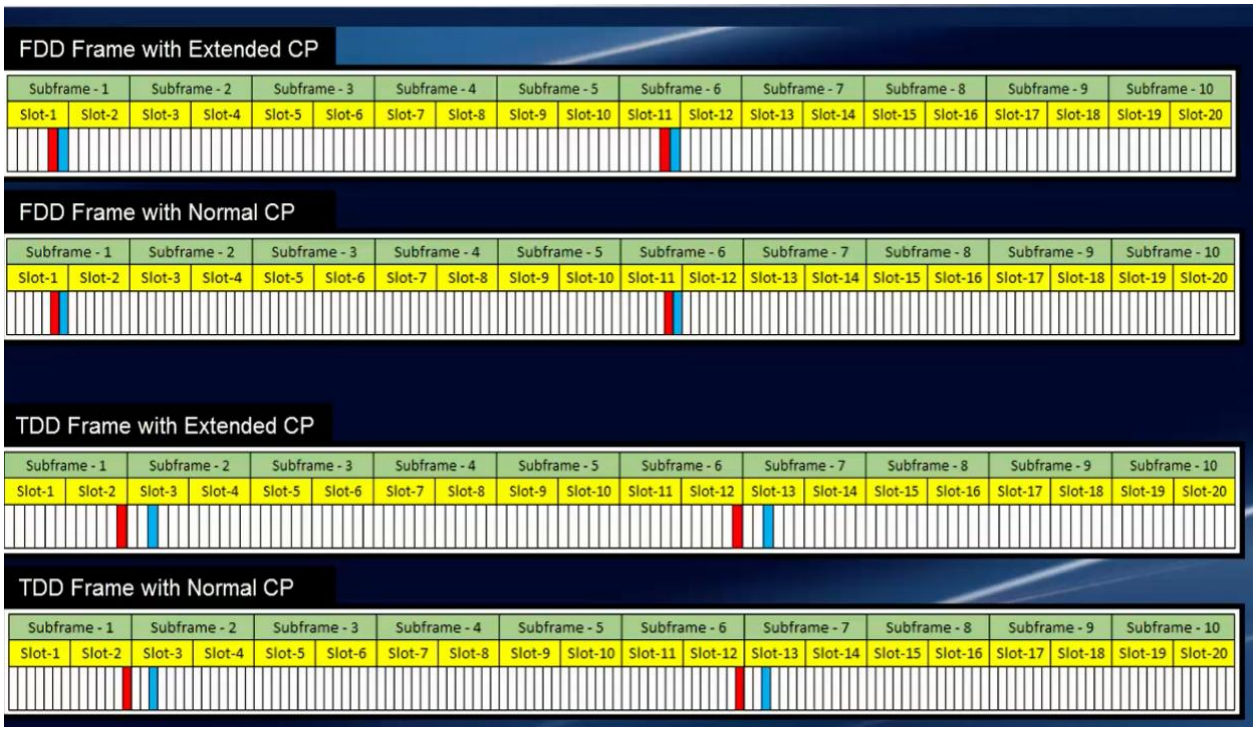


Figure 11-12 Differences between all possible transmission schemes

During the first step in cell search procedure the receiver will correlate the received waveform with the 3 possible PSS sequences with the 2 possible lengths for both normal and extended CP and then it can decide from the max correlation which length of CP is used.
In the second step the receiver still doesn't know whether the duplexing technique used is FDD or TDD so it will correlate the SSS sequence in both possible positions and after this stage the receiver will obtain the cell ID and estimates the frame timing of the identified cell as well as the length of the CP and which duplexing technique is used.

11.2 MASTER INFORMATION BLOCK

The very first step for UE to gain initial access to the network after completing initial cell synchronization is to read the Master information block (MIB) PBCH (Physical broadcast channel). The PBCH is transmitted using QPSK modulation scheme. A combination of repetition coding and convolutional coding,

MIB carries very little but most important information for UE initial access.The MIB is transmitted using a physical layer channel called PBCH on downlink. The MIB is a 24 bit information with following information within

- 3 bits for system bandwidth in term of resource blocks (RBs)
- 3 bits for PHICH information,

  1 bit to indicate normal or extended PHICH
  2 bit to indicate the PHICH Ng value
- 8 bits for system frame number
- 10 bits are reserved for future use

Apart from the information in the payload, the MIB CRC also conveys the number of transmit antennas used by the eNodeB.

11.2.1 Master Information Block Importance

The system bandwidth is a must knowledge a UE must have to decode any other physical layer channels. PHICH info, which is nothing but the Ng value (or the number of PHICH groups used in the first symbol) is very important too, because unless the UE knows that, he will not be able to decode PDCCH and hence any DCI. This field also tells UE about the PHICH duration, if it is normal PHICH or extended PHICH, which is also essential for the UE to decode the control channels completely
System frame number (SFN) is transmitted in the MIB for mainly two reasons, one, for initial synchronization and two for periodic sync that a UE can do to find out any drifts of its timing vs the eNodeB's timing The system frame number within each MIB will be a rolling number between 0-255 and keeps changing every MIB but other contents may or may not change.

11.2.2 Master information Block positioning

New MIB is broadcasted every radio frame for which SFN mod 4 = 0 (40ms repetition) while its copies are broadcasted in the middle 10ms radio frames as shown in the figure below

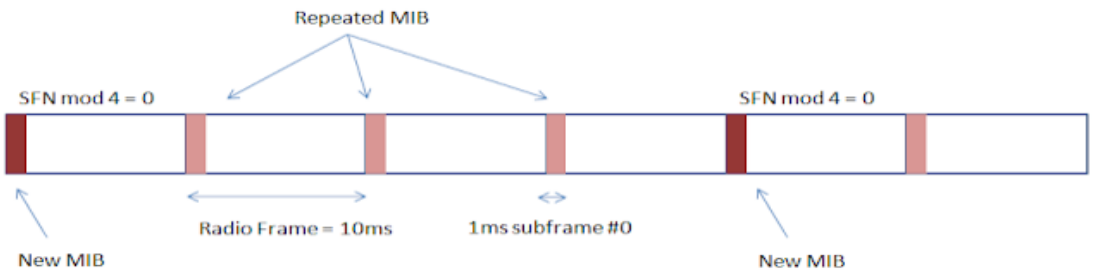


Figure 11-13 MIB periodicity and repetition

Resource elements used by MIB are the first 4 OFDMA symbols of second slot of first subframe of a radio frame for both FDD and TDD as show in 'figure 11.15'.
In frequency domain it occupies 72 subcarriers which is equivalent to 6 Resource Blocks (RB).

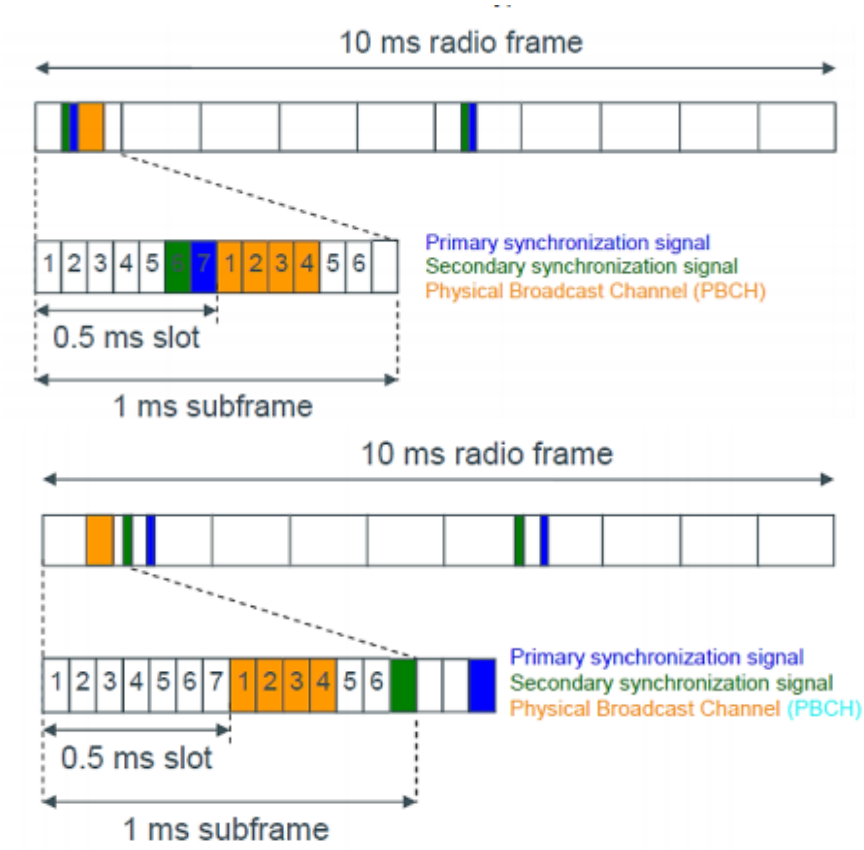


Figure 11-14 Position of the PBCH,PSS and SSS for both frame struc

11.3 DIGITAL TEAM CONTRIBUTION

As mentioned in the previous sections of chapter 2 that the reason behind this chapter's work is to know the PCI and to synchronize time with the EnodeB. Firstly, to know the PCI(Physical Cell ID) you need to know which version of PSS and which version of SSS are sent either, as PSS tells us the NID2 and SSS tells us NID1 and we have both we can use the formula PCI= 3NID1+NID2, and to synchronize time you need to know by how much the synchronization signals are delayed.

Also previously in this chapter we knew that PSS can have one of three sequences, which gives the three different NID2 values {0,1,2} ,and SSS can have one of 168 sequences, which gives the 168 different NID1 values {0 till 167}.

Each PCI has a unique combination of PSS and SSS signal. (Note that 3 × 168 = 504). To find PCI all 504 combinations of PSS and SSS must be correlated with all the delayed versions of the signal. The PSS, SSS, and delay combination which gives the highest correlation provides an estimate of the PCI as well as the timing offset.

As for the timing synchronization, when correlating the PSS, we can achieve slot synchronization as we know the symbol position where pss is sent. While SSS correlation provide us with frame synchronization because two different SSSs is sent in each frame, therefore if we knew which one is correlated we can know the which slot we are in , and hence knowing the beginning and end of the frame

Therefore, as a Digital Team We implemented a cross correlator Verilog code that takes two inputs and correlates them together and outputs the correlation Product. Then the correlator output is then fed to a Max finder circuit that find where the correlation peak is, this should be done for each possible combination of The PSS and SSS.

11.3.1 Cross Correlation

In general, correlation describes the mutual relationship which exists between two or more things. The same definition holds good even in the case of signals. That is, correlation between signals indicates the measure up to which the given signal resembles another signal.

In other words, if we want to know how much similarity exists between the signals 1 and 2, then we need to find out the correlation of Signal 1 with respect to Signal 2 or vice versa.

Cross correlation is a kind of correlation, in which the signal in-hand is correlated with another signal so as to know how much resemblance exists between them. Mathematical expression for the cross-correlation for the discrete time signals *x* [*n*] and *y* [*n*] is expressed as

$$R_{xy}[m] = \sum_{n=-\infty}^{\infty} x[n]\, y^{\star}[n-m]$$

Figure 11-15 Cross correlation Formula

where $y^{\star}[n-m]$ denotes the complex Conjugate of y[n-m].

As an example, consider two real valued functions x[n] and y[n] differing only by an unknown shift along the x-axis. One can use the cross-correlation to find how much y[n] must be shifted along the x-axis to make it identical to x[n]. The formula essentially slides the y[n] function along the x-axis, calculating the sum of their product at each position. When the functions match, the Rxy is maximized. This is because when peaks are aligned, they make a Large contribution to the Sum. Similarly, when troughs align, they also make a large contribution to the integral because the product of two negative numbers is positive.

Also, cross-correlation of any two given signals can be found via graphical techniques. Here, one signal is slid upon the other while computing the samples at every interval. That is, in the case of digital signals, one signal is shifted by one sample to the right each time, at which point the sum of the product of the overlapping samples is computed.

For example, cross-correlation of the digital signals x [n] = {-3, 2, -1, 1} and y [n] = {-1, 0, -3, 2} can be computed as shown by Figure 11.16.

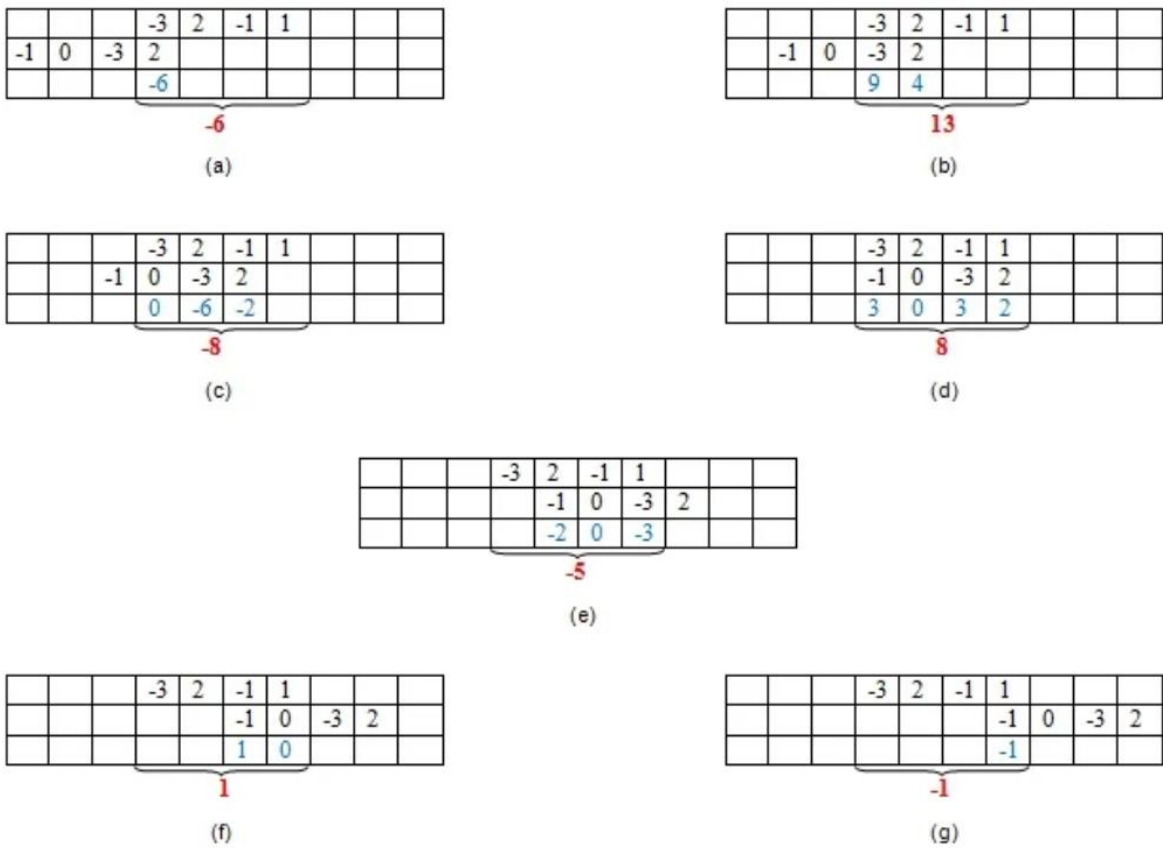


Figure 11-16 Cross correlation calculation using graphical method.

Here, the first set of samples (in the first row of every table) refers to the signal x [n] and the second set refers to the samples (in the second row of every table) of the signal y [n].

Next, the samples shown in blue color—those in the third row—are obtained by multiplying the corresponding samples of the first two rows. Finally, we add the samples in the last row (contained within the curly brackets) so as to obtain the samples of the cross-correlated signal.

Thus, here we see that the samples of the cross-correlated signal Rxy are obtained as {-6, 13, -8, 8, -5, 1, -1}, where 8 is the zeroth sample.

Further, the example presented shows that the sample of the cross-correlated signal is at its highest peak, with value 13, when the last two samples of *y* [*n*] overlap with the first two samples of *x* [*n*]. This is because, in this case, the second signal overlaps with the first at its best, as the two samples in each of the signals are identical.

Hence, it can be concluded that the cross-correlation reaches its maximum when the two signals considered become most similar to each other.

In our Case we will use the Cross correlation For two Reasons, the first is to get the PCI by finding which PSS and SSS signals are sent By cross correlating the input with all possible values of either PSS or SSS this will give us 3 peaks in case of PSS and, 167 peaks in case of SSS these peaks are fed to comparator to get the maximum and know which signals were sent, the second is to synchronize

11.3.2 Cross correlator code for real inputs

This code is used for SSS as it has only real values, note that this code is simulated for inputs with three values for easier checking of simulation results, but we will put the simulation results of inputs with 62 values corresponding to the 62 central subcarriers at the end of this section.

Two Verilog codes were written and simulated for the real input correlator to verify the outputs , one is behavioral, and one is structural , behavioral, and structural are different styles of coding in HDL(both codes can be found in the appendix with their testbenches). Then the structural code is synthesized, and its bit file is generated.

11.3.2.1 behavioral code

The behavioral simply describes the behavior of the circuit without instantiating any instances. Figure 11.17 shows the simulation results of the testbench of the behavioral code : input x has 3 values fed serially one at each positive clock edge (1 2 1) at the same time input y also has 3 values fed serially one at each positive clock edge (2 1 2) , then the correlation

output r has 5 values due the shifting and multiplication according to the cross correlation formula mentioned before. These values were verified using MATLAB and hand analysis.

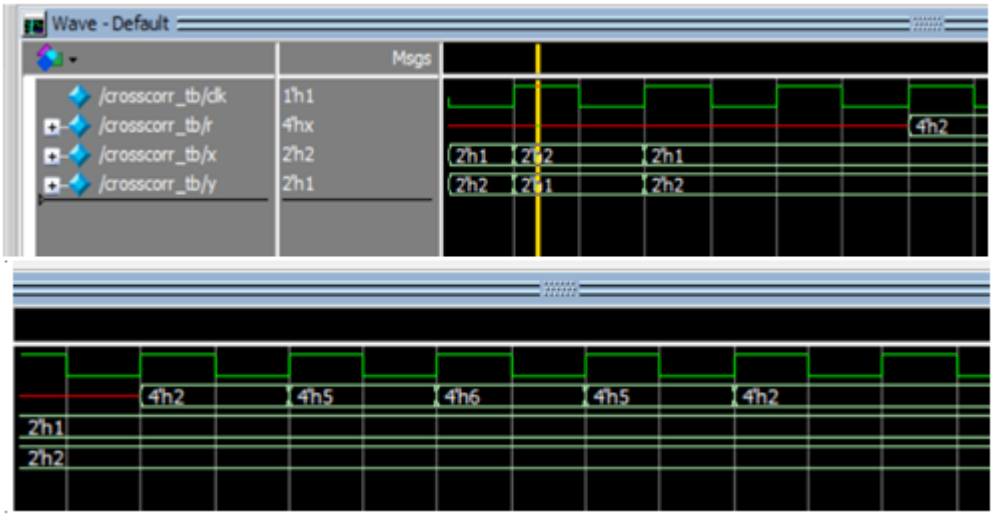


Figure 11-17 Simulation Results of behavioural code

After verifying the operation of the circuit, As shown in figure11.18 we simulated the circuit with inputs having 62 values fed serially at each clock cycle, we kept the x and y values @ 1 so that the value of r gets incremented by 1 each clock cycle. We can also see from the figure that the first output comes after 63 clock cycles(630 ns) as the input values are fed serially to the circuit in the first 62 cycles and then it takes one clock cycle to produce the first output. Also, we can see that the output is incremented by one each clock cycle until it reaches 62 then it decreases until it reaches one meaning that since all values are positive maximum correlation appears at zero shift.

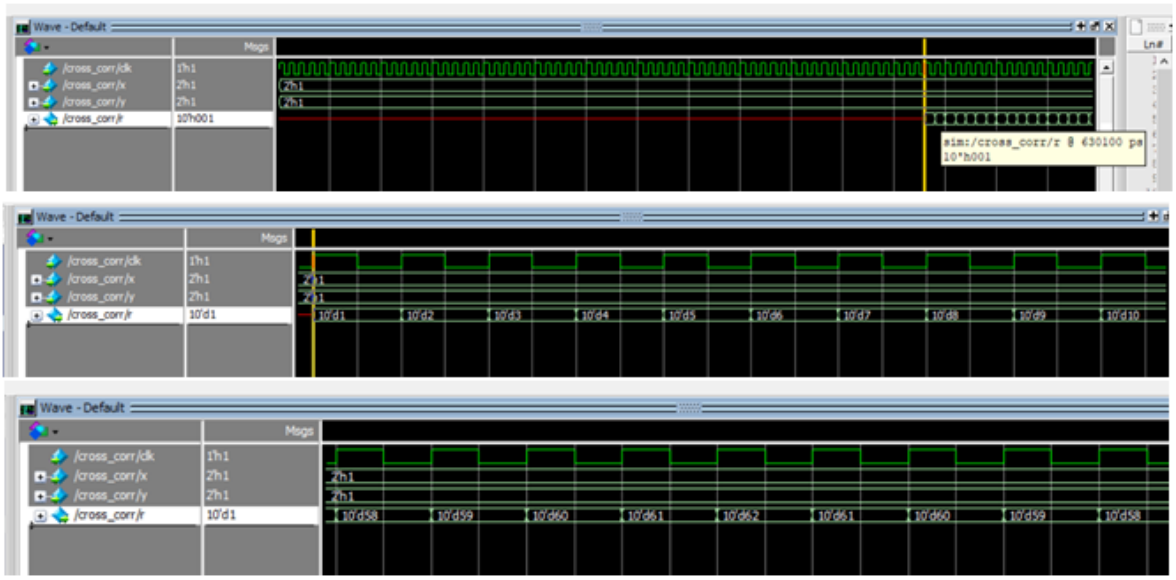


Figure 11-18 Simulation Results of Behavioural code with 62 input value

11.3.2.2 Structural code

The structural code shows the blocks that make the code by calling these blocks through instantiation of these instants or blocks.

The difference we will see in the simulation results from the results of the behavioral code is that the x and y values are fed into different inputs meaning we have 6 inputs fed parallelly instead of 2 inputs fed serially having 3 values also the code feeds output parallelly in 5 different outputs instead of one output having 5 different values fed serially.

Figure 11.19 shows the simulation results of the testbench of the structural code : The same x and y values are fed into the code (1 2 1) and (2 1 2) and also the output is the same as the behavioral code.

Note that the outputs and inputs in the results seems to be in one variable (serial inputs and outputs) that's because in the test bench i wrote them in the form of one matrix but in the appendix you can see in the code that the inputs and outputs are parallel, also you can see from the results that all the outputs are generated in the same clock cycle meaning the circuit works parallelly.

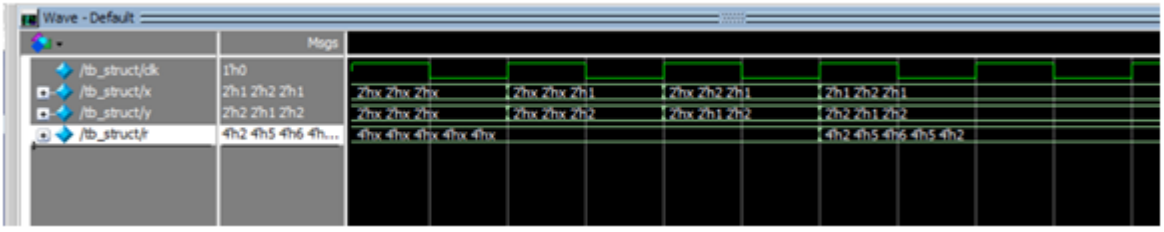


Figure 11-19 Simulation Results of structural code

11.3.2.3 Synthesis

The synthesis process is done through Xilinx Ise as there is N210 SDR in the faculty available having a Xilinx FPGA. The code is synthesized completely and ready to be burnt on the FPGA, but unfortunately due to the pandemic we were not able to test it.

Figure 11.20 shows that the structural code was completely synthesized, and the programming(bit) file is ready to be burnt on the FPGA.

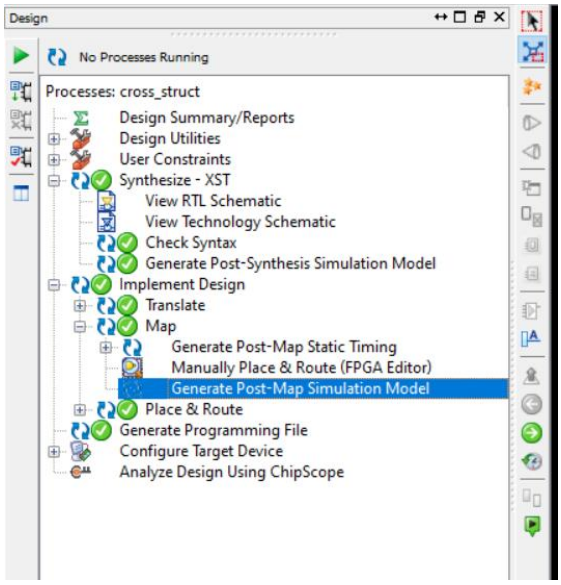


Figure 11-20 programming file generated and ready to be put on FPGA

Figure 11.21 shows the RTL schematic of the Code which consists of a number of Multipliers, adders and, D-flipflops, Which was expected as the formula of the cross correlation contains adding and multiplying functions, while D-flipflops are added for synchronous operation as we described in the code.

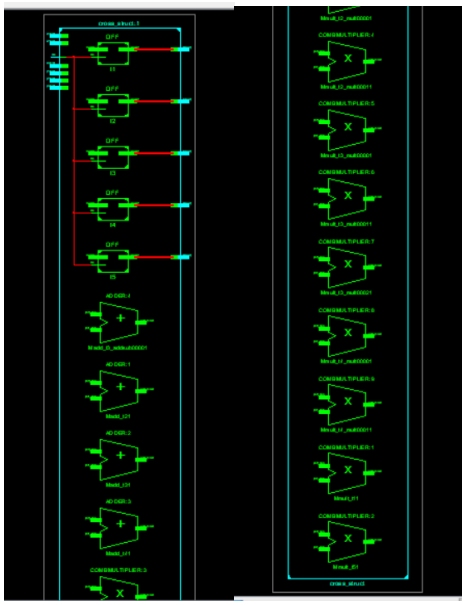


Figure 11-21 schematic of the correlator

Figure 11.22 shows a part of the technology schematic of the Code which consists of the FPGA resources allocated for this circuit e.g. LUTs, DSP Slices, and buffers.

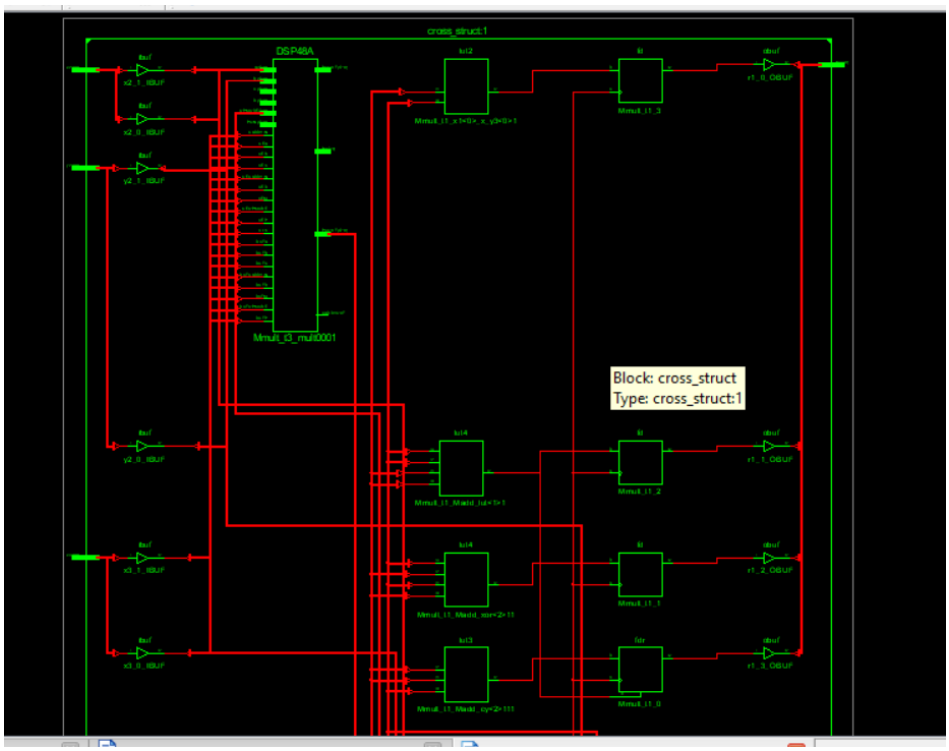


Figure 11-22 part of the Technology schematic of the correlator

11.3.3 Cross correlator code for complex inputs

This code is used for PSS since it uses complex values, note that this code is simulated using modelsim for inputs with three values same as the real valued code for easier verification, then inputs with 62 values were simulated for more realistic simulation.

We also wrote two codes one behavioral and one structural, for writing the complex correlator we first wrote a code for a complex multiplier (found in the appendix) as the normal Verilog multiplication function would not be of use. The complex multiplier code was instantiated in our code and we wrote the correlator function in a structural way. After that, a testbench was written to test the structural code, finally synthesis using Xilinx ISE was performed for the structural code.

11.3.3.1 behavioral code

Firstly, to verify the operation we used 3 valued inputs as shown in figure 11.23, these values were verified by MATLAB and hand-analysis. Then to simulate the real circuit inputs with 62 values were fed to the circuit.

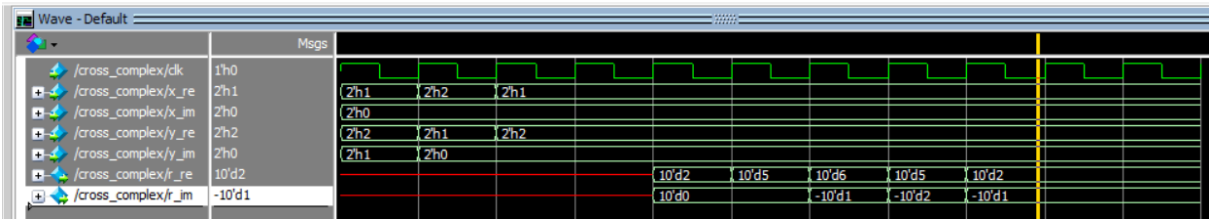


Figure 11-23 simulation results of behavioural complex Correlator

After verifying the operation of the circuit, As shown in figure11.24 we simulated the circuit with inputs having 62 values fed serially at each clock cycle, we kept the x_re and y_re values @ 1 so that the value of r_re gets incremented by 1 each clock cycle, and we kept x_im @1 and y_im @ 0 so that the value of r_im gets incremented by 1 each clock cycle .We can also see from the figure that the first output comes after 63 clock cycles(630 ns) as the input values are fed serially to the circuit in the first 62 cycles and then it takes one clock cycle to produce the first output. Also, we can see that the outputs is incremented by one each clock cycle until it reaches 62 then it decrease until it reaches one.

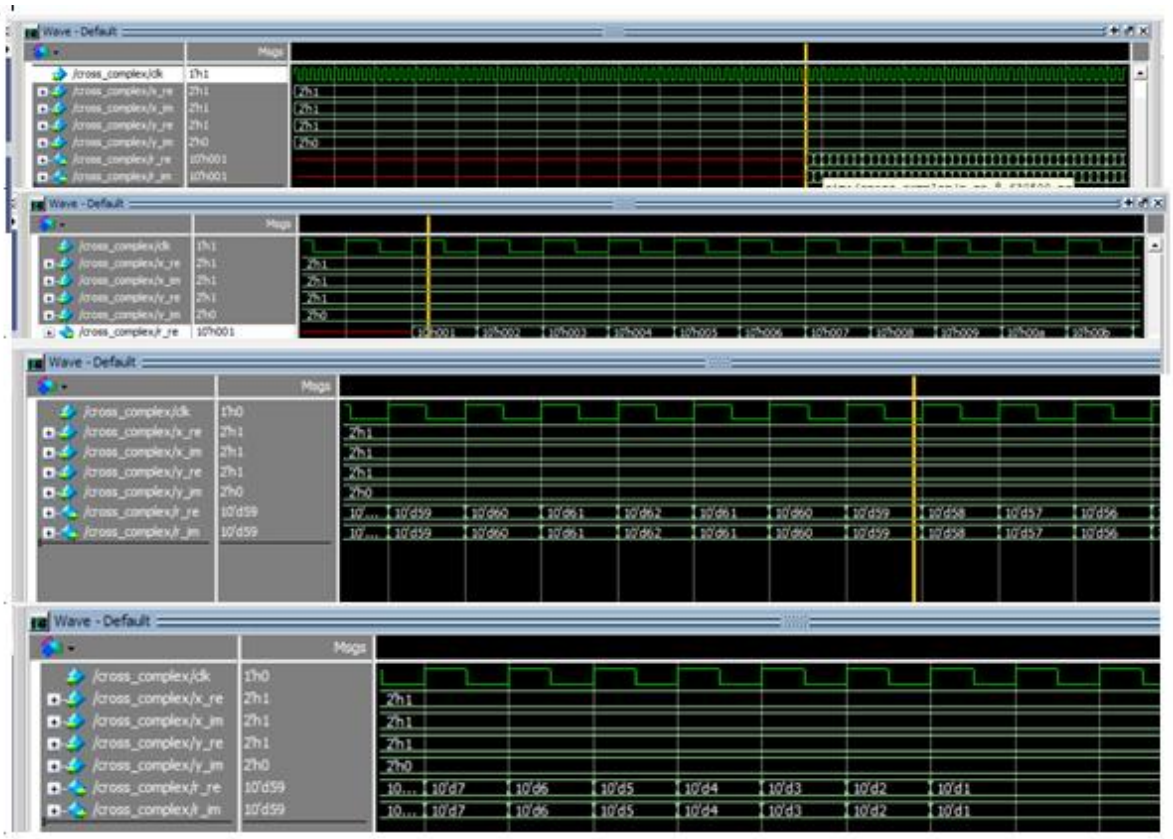


Figure 11-24 simulation results of behavioural complex Correlator with 62 input values

11.3.3.2 structural code

Figure 11.25 shows the simulation results of the structural code testbench : the real part of the input x (x_re) has 3 values fed serially one at each positive clock edge (1 2 1) at the same time the imaginary part of the input x (x_im) also has 3 values fed serially one at each positive clock edge (0 0 0) , similarly y_re has values (2 1 2) and y_im has values (1 0 0), then the correlation output r has 5 real values , and 5 complex values obtained from the shifting and multiplication according to the cross correlation formula mentioned before.

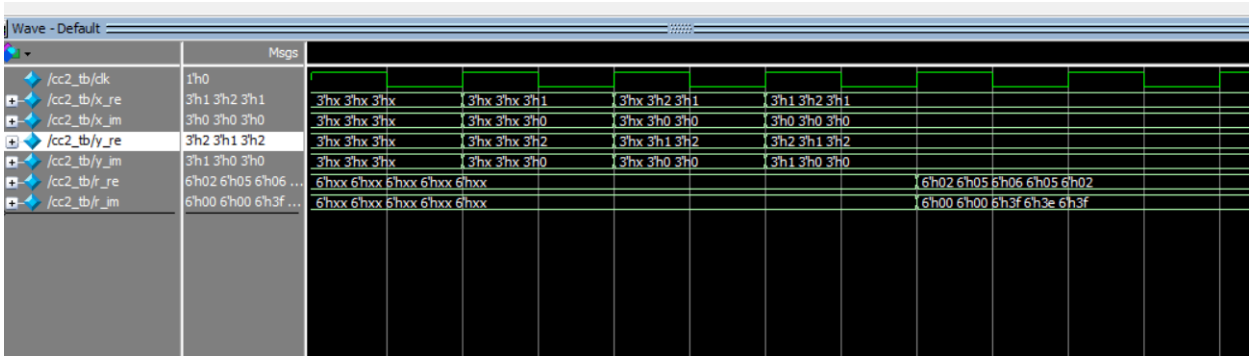


Figure 11-25 simulation results of structural complex Correlator

11.3.3.3 Synthesis

As mentioned before the synthesis process is done using Xilinx ISE. Figure 11.26 shows that the design was completely synthesized, and the programming(bit) file is ready to be burnt on the FPGA.

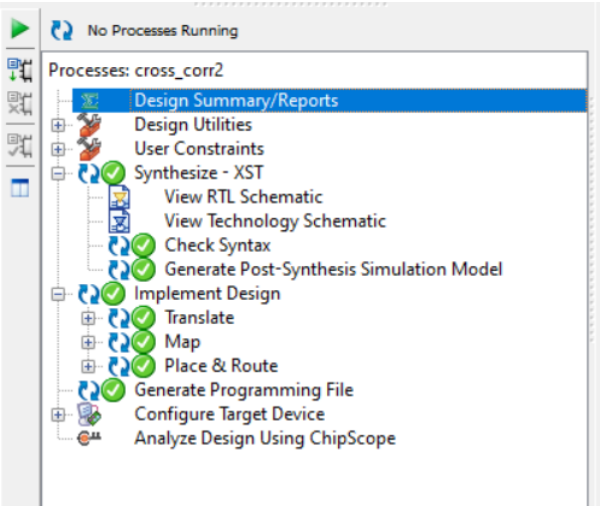


Figure 11-26 complex correlator programming file generated and ready to be put on FPGA

Figure 11.27 shows the block diagram and parts of the RTL schematic of the Code which consists of a number of adders, modules of the created complex multipliers and, inverters. You can see that the number of multipliers and adders have increased when compared with the real valued correlator which is due to increased number of inputs and outputs.

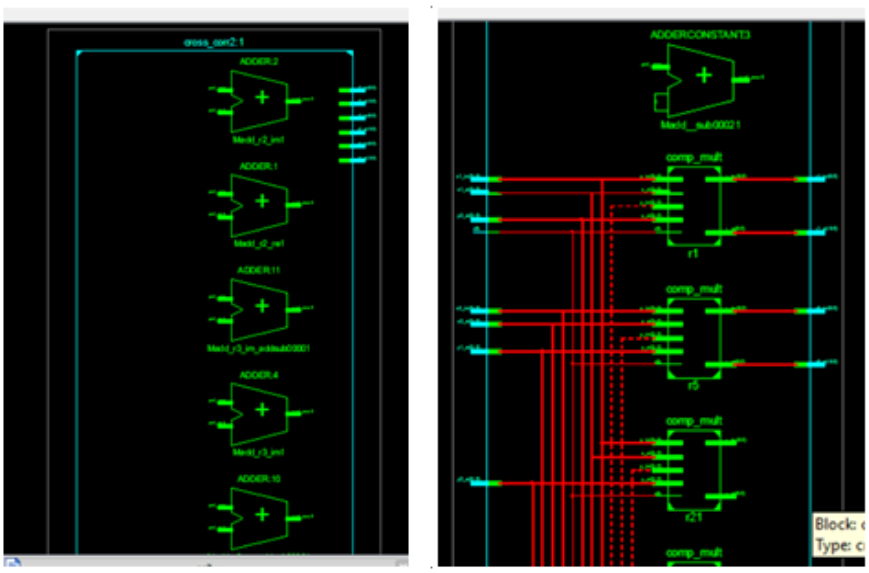


Figure 11-27 RTL schematic of complex correlator.

Figure 11.28 shows the RTL schematic of the complex multiplier module consisting of adders, multipliers, and D-flipflops.

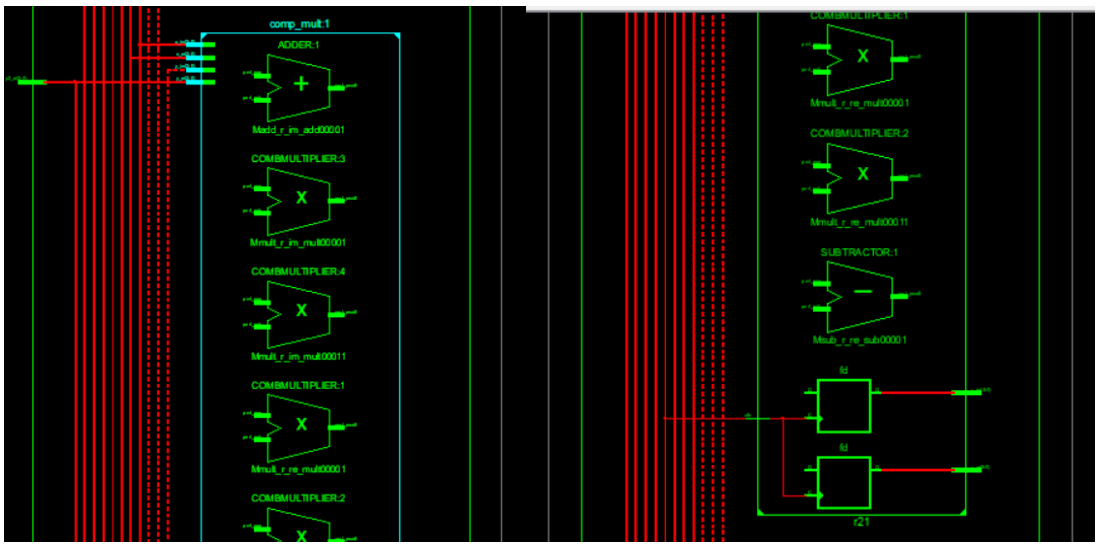


Figure 11-28 RTL schematic of complex multiplier.

Figure 11.29 shows a part of the technology schematic of the Code which consists of the FPGA resources allocated for this circuit e.g. LUTs, DSP Slices, and buffers.

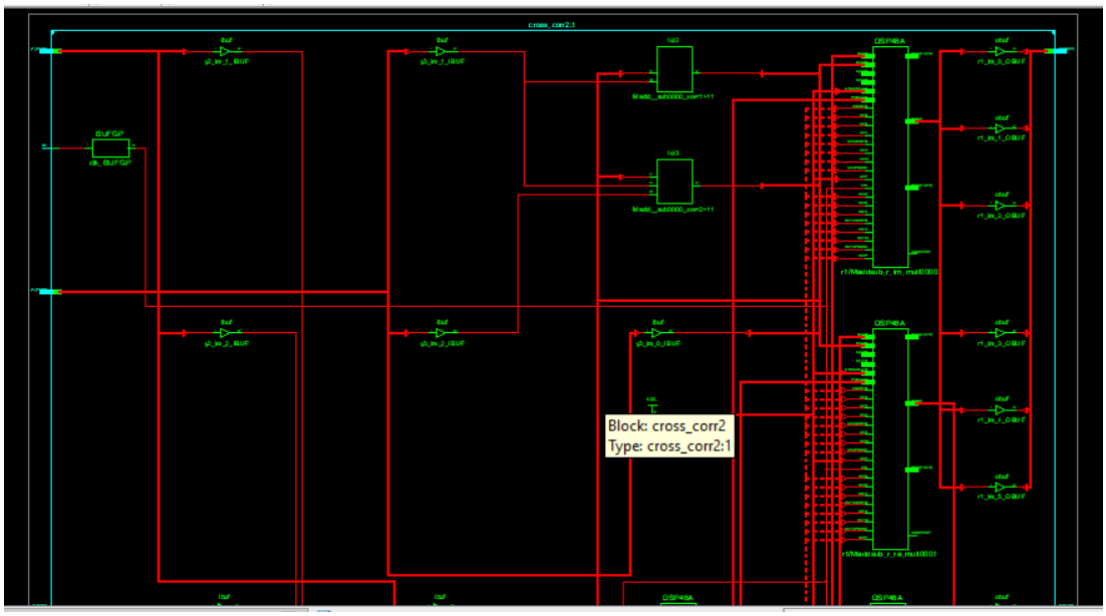


Figure 11-29 Technology schematic of complex correlator.

11.3.4 Max Finder Circuit

After the correlator operates, we need a circuit that finds the maximum correlation value to determine which shift of the input relative to the previously known sequence is best. Therefore, we designed a circuit that takes the correlator outputs and find the maximum and output it and also outputs the value of the shift of the input relative to the previously known sequence. Supposedly we should have three circuits doing this in case of PSS and 168 in the case of SSS and then a comparator would find the final maximum value and the final shift value of all the Max finder circuits in each case, but in our case we made only one because the other one is a very huge circuit that would be very hard to simulate on our computers , but its can easily be done by simply repeating these circuits we designed.

Since we made two correlators one for complex valued inputs and one for real valued inputs, we had to make two max finder circuits accordingly. The max finder circuit for the complex inputs first calculate the magnitudes of the circuit inputs(correlator outputs) and finds the maximum.

11.3.4.1 Real input Max finder

We fed 5 inputs to the circuit with values (2 5 6 5 2) which we assumed to be the outputs of a three valued input correlator as the one we designed therefore the shift indices equals (-2 -1 0 1 2) then, as shown in figure 11.30 the max was 6 and the shift was 0 which is the correct values.

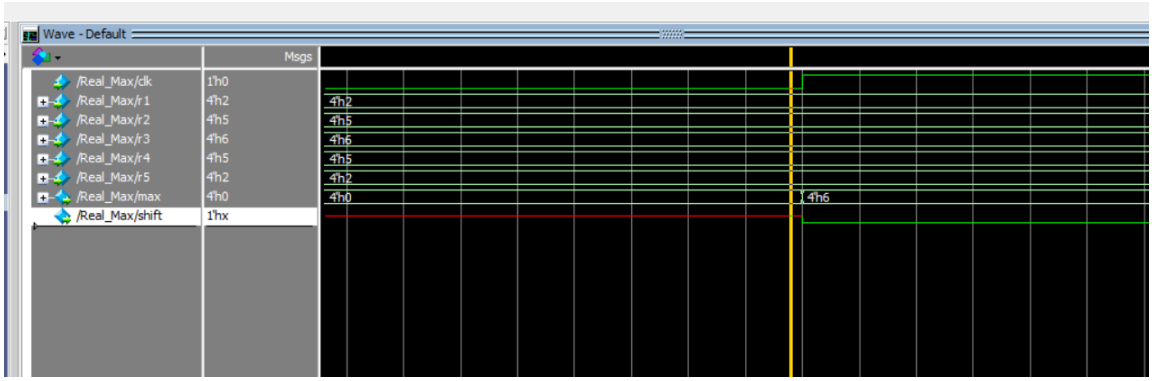


Figure 11-30 simulation results of real input max finder.

11.3.4.2 Complex input Max finder

When simulating this code we fed the same inputs as the real input max finder to the real part of the inputs and for the imaginary part we fed the same value to all the inputs which is 1 so that the circuit produce the same answers as the real input max finder.
As shown in Figure 11.31 the 'max_re=6' which is the real part of the maximum number and , as for the imaginary part 'max_im=1' as it's the same for all inputs also, the 'shift =0' which is the same value as the last circuit.

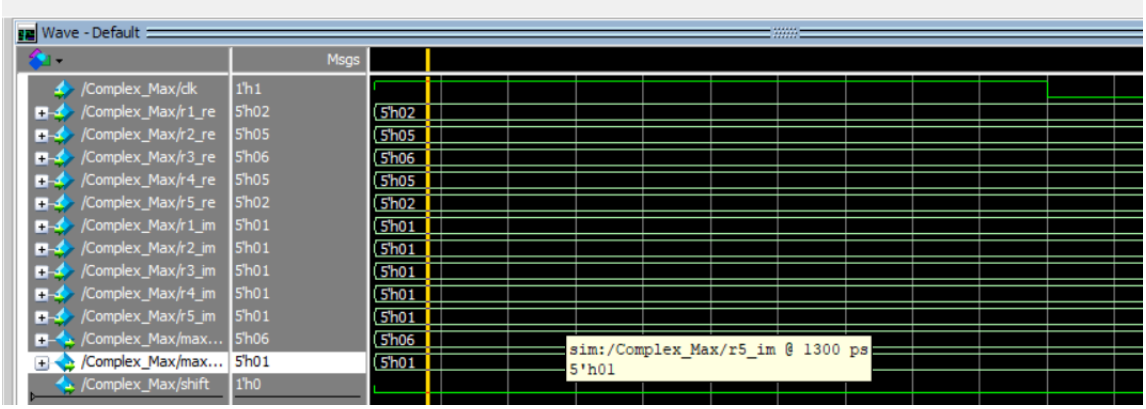


Figure 11-31 simulation results of complex input max finder

Synthesis of these codes will be provided in the next section as part of synthesizing the final circuit (Top module) including both the correlator and the max finder circuits.

11.3.5 Top Module

The top module or the final circuit is the Verilog module that includes all the smaller modules that make up the circuit in our case it contains the correlator and the max finder. We made two top modules one for real numbers and one for complex numbers, in the real one we instantiated the real correlator and max finder while, for the complex one we instantiated the complex correlator and max finder.

11.3.5.1 Top Module for real valued inputs simulation results

First, we instantiated the real cross correlator module and the real max finder module. The max finder module inputs are the outputs of the correlator. And inputted x and y 62 values with ones so that the maximum would be 62 @ shift=0, as shown in figure 11.32.

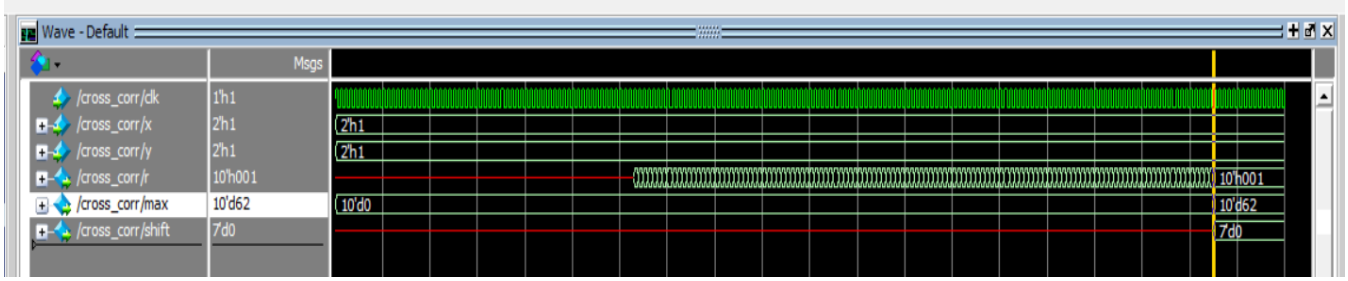


Figure 11-32 simulation results of real valued input top module.

11.3.5.2 Synthesis of Top Module for real valued inputs

The synthesis process is done using Xilinx ISE. Figure 11.33 shows that the design was completely synthesized, and the programming(bit) file is ready to be burnt on the FPGA.

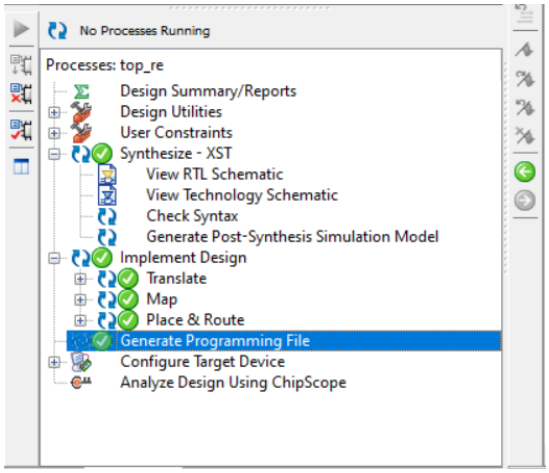


Figure 11-33 Top module Programming file generated and ready to be put on FPGA

Figure 11.34 the RTL schematic of the top module which consists of a cross correlator and max finder modules.

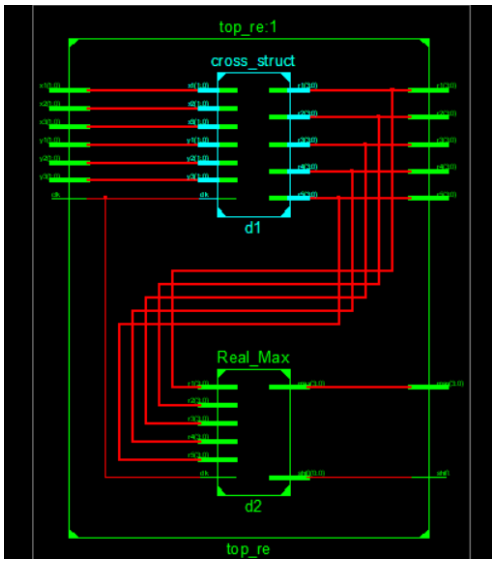


Figure 11-34 RTL Schematic of real valued input top module

Figure 11.35 shows the RTL schematic of the Max finder which consists of a number of comparators, nand gates, and buffers

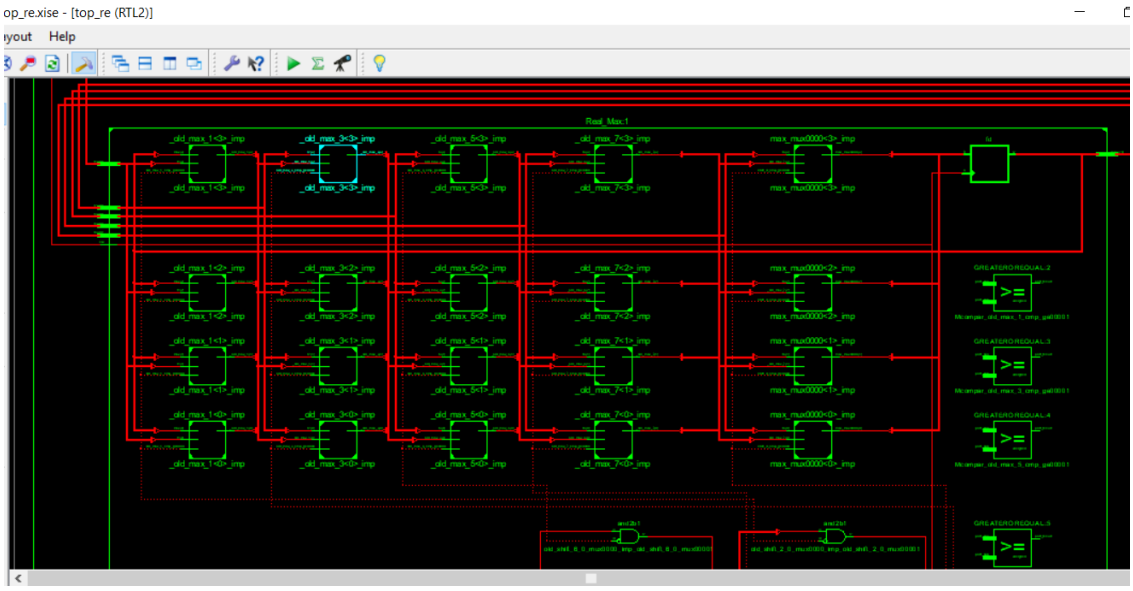


Figure 11-35 RTL schematic of Max finder

Figure 11.36 shows parts of the technology schematic of the Code.

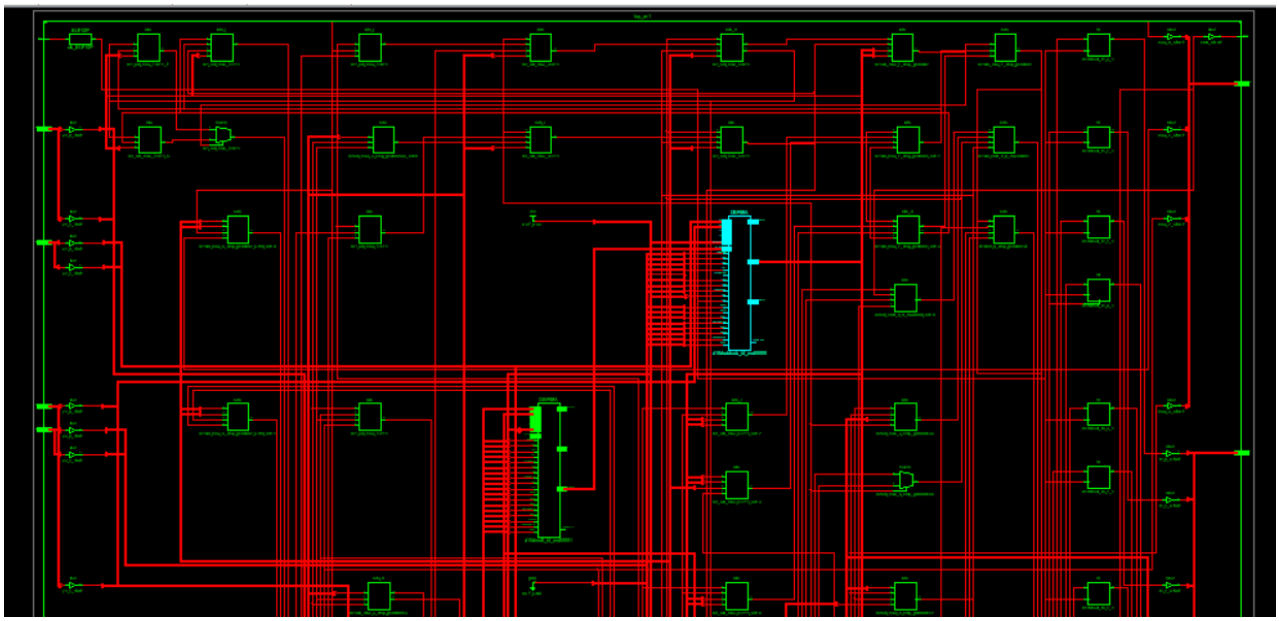


Figure 11-36 Technology schematic of real valued input top module.

11.3.5.3 Top Module for complex valued inputs simulation results

We did the same procedure as last time but using the complex versions of the correlator and the max finder. Inputs x_re, y_re, x_im are kept @1 while y_im is kept @ zero this will cause r_re and r_im to have peaks at the same time = 62 when the shift=0. This is shown in figure 11.37.

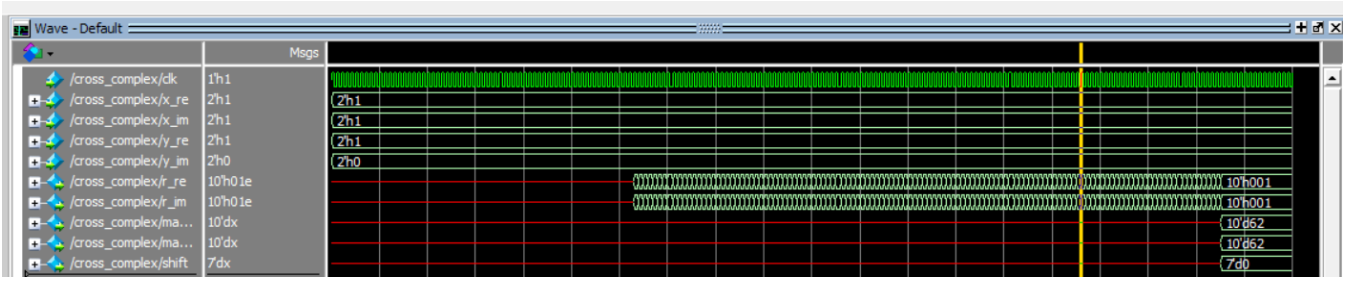


Figure 11-37 simulation results of complex valued input top module

11.3.5.4 Synthesis of Top Module for complex valued inputs

The synthesis process is done using Xilinx ISE. Figure 11.38 shows that the design has some warnings but was completely synthesized, and the programming(bit) file is ready to be burnt on the FPGA

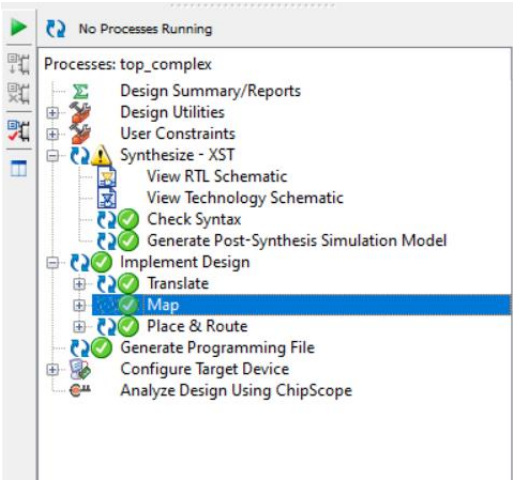


Figure 11-38 Complex Top module Programming file generated and ready to be put on FPGA.

Figure 11.39 shows the block diagram, and the RTL schematic of the complex top module which consists of a complex cross correlator and a complex max finder module.

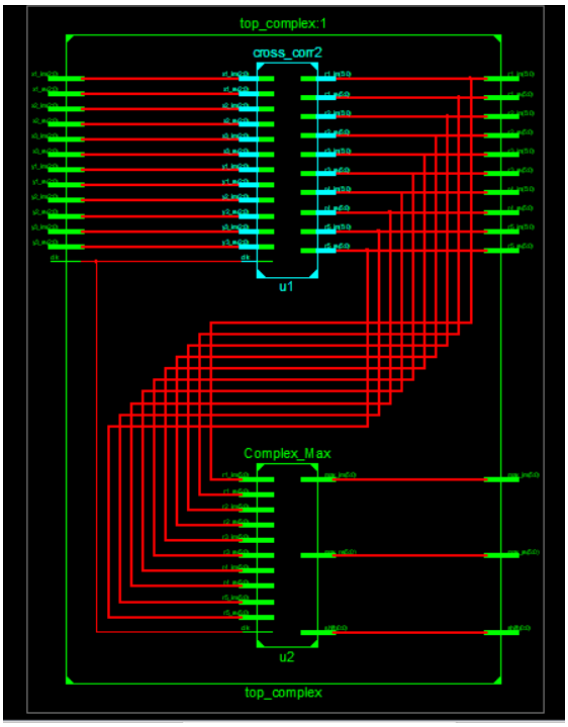


Figure 11-39 RTL Schematic of complex valued input top module

The complex max finder is the same as the real one but with more inputs, Figure 11.40 shows the Technology schematic of the complex top module consisting of mainly DSP slices and D-flipflops.

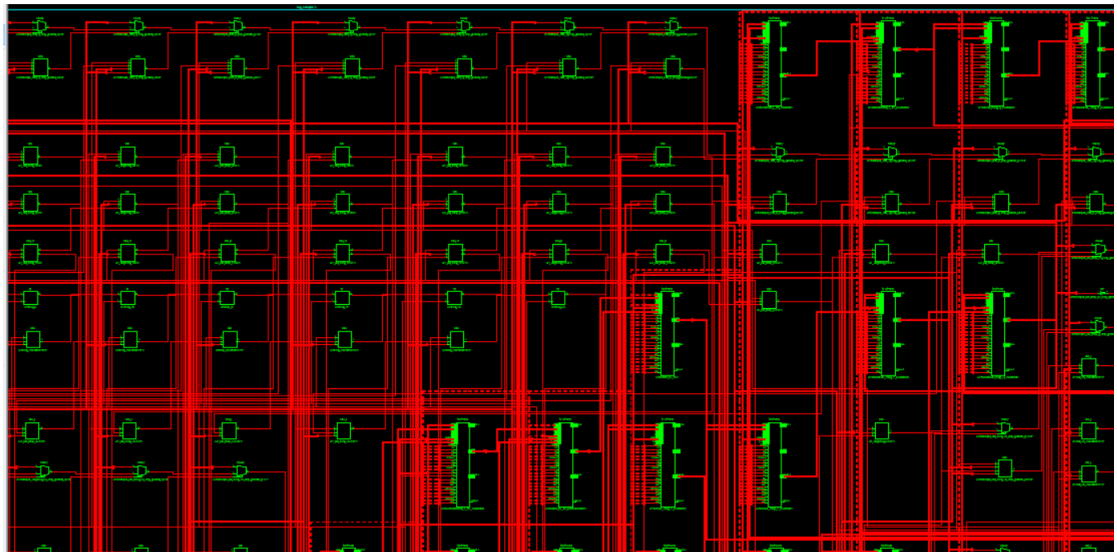


Figure 11-40 Technology Schematic of complex valued input top module

11.3.6 Conclusion

To perform a complete cell search we need to have 168 correlator working in parallel assuming that PSS correlation is done before SSS, to do so we need a massive number of multipliers and adders ,therefore this solution is impractical as it needs very large area .

One way to overcome this problem is not performing all the correlations at the same time, but instead reusing the available resources at different times thus reducing area but on the other hand this method has drawbacks.

Firstly, to perform multiple correlations one after the other you need to store the data until you perform all the correlation operations. Therefore, you need more memory which leads to higher area, of course it’s much less than they are needed when using the first method.

Secondly you need a control circuit to control the resources used in correlation and decide which sequence gets correlated and tell the comparator when to compare between all the correlation peak when all of them are finished.

Finally, but most importantly this method takes more time , and the time increases linearly by decreasing the number of used resources for example if we used half the used correlators(168/2=84) the time to retrieve the PCI doubles and so on, and this is a huge drawback as the cell search should take very low time.

To conclude, there is a trade-off mainly between area and time, so we need to know the maximum required time to do the cell search procedure to be able to decide the shape and size of our circuit.

Another choice we have to make is whether to generate the 3 PSS and 168 SSS sequences on the circuit ,each time we need to do the cell search procedure which will need more processing time, or just store them in memory inside our circuit which will increase the memory needed on the circuit thus the area .

Therefore, we have another trade-off between time and area, so we need to make the optimum choice according to the required specifications

12 OFDM

12.1 INTRODUCTION

OFDM: Orthogonal Frequency Division Multiplexing, is a form of signal modulation that divides a high data rate modulating stream placing them onto many slowly modulated narrowband close-spaced subcarriers, and in this way is less sensitive to frequency selective fading.

12.1.1 Why OFDM?

1 - it can be seen as a kind of multi-carrier transmission by allowing the base station to communicate with several different mobiles at the same time
2 - it solve the most big problem in wireless communication which is the multipath between radio transmitters and receivers

Multipath :
It's a phenomenon happens when radio signals reach the receiving antenna by two or more paths.
Causes of multipath:
Causes include terrestrial objects such as mountains and buildings and atmospheric phenomena, and
reflection from water bodies and terrestrial objects such as mountains and buildings
Results of multipath
Interference including constructive and destructive interference, and phase shifting of the signal.
Destructive interference causes fading, where the magnitudes of the signals arriving by the various paths .

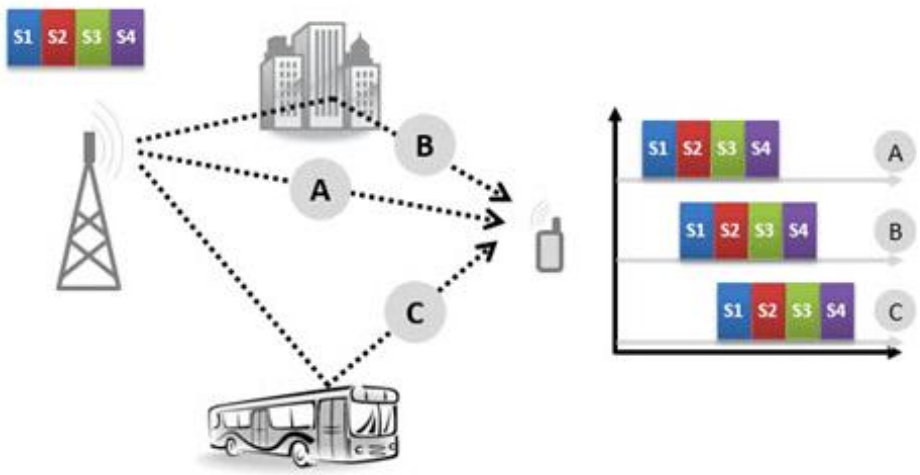


Figure 12-1Multipath propagation from the Transmitter and the Receiver

At the figure the signal has more than one path to reach antenna , path A took less time .If the path lengths of the longest and shortest rays are different, then symbols travelling on those rays will reach the receiver at different times

The following figure shows the power delay profile (PDP) which gives the intensity of a signal received
through a multipath channel as a function of time delay.

Time delay: is the difference in travel time between multipath arrivals

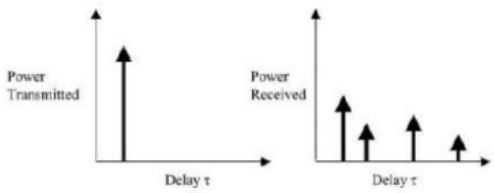


Figure 12-2Power Delay Profile

12.1.2 Intersymbol Interference (ISI)

Intersymbol interference (ISI) is a form of distortion of a signal in which one symbol interferes with subsequent symbols. This is an unwanted phenomenon as the previous symbols have similar effect as noise, thus making the communication less reliable.

All these "multipath" components are summed up . And as a result, there are multiple symbols being received simultaneously, Delayed version of one symbol from an indirect path interferes with a different symbol causing overlapping of symbols occurs and that what is called “intersymbol interference (ISI)” and it happens specially in high data rate because when data rate increases, so the symbol duration falls and be less than delay time and the problem becomes progressively worse

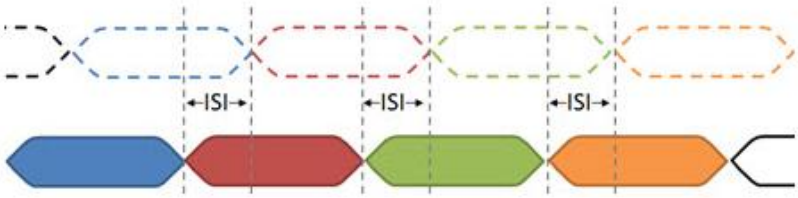


Figure 12-3Multipath-Induced Time Delays Result in ISI

Multipath distortion can also be considered in the frequency domain. Each different path length and reflection will result in a specific phase shift. As all of the signals are combined at the receiver, some frequencies within the signal passband undergo constructive interference, while others encounter destructive interference. The composite received signal is distorted by frequency selective fading

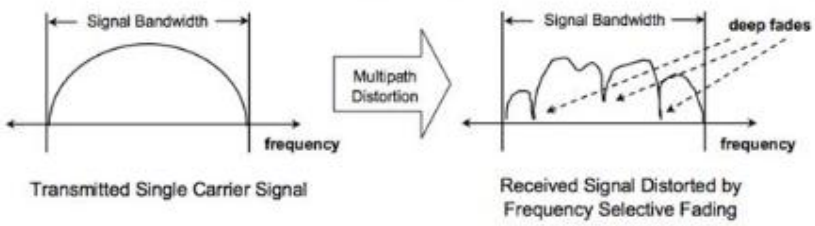


Figure 12-4Longer Delay Spreads Result in Frequency Selective Fading

Delay Spread

A transmitted symbol can be received multiple times at the receiver, more or less as an “echo” effect. This echo is what we call “Delay Spread”.

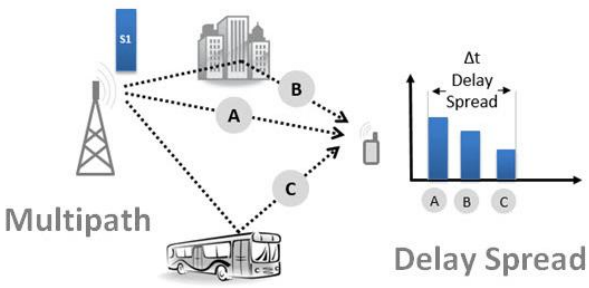


Figure 12-5 delay spread

In the above figure, the transmitter transmits a single symbol. This symbol is propagated along different paths (A, B and C), and eventually reaching the receiver at multiple time instants, and therefore with multiple “replication.”

The total elapsed time between the first and last is determined by the environment (including the structures, how close they are, etc…). For example, in an urban environment, where the reflection is high (many buildings, many vehicles parked and moving), this delay has a typical value of 5-10 microseconds.

So, the most significant factor concerning the ISI is the Symbol Duration. If the symbol period (T) is very short compared to the "Delay Spread" (t) the impact is significant (T << t)

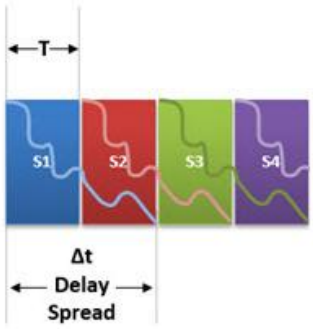


Figure 12-6 Symbols with time delay spread

But if the symbols‟ length is extended (T >> t), there will be less of the ISI‟s impact on the symbols as one
small part of the symbol will be impacted but, for most of its duration, the symbol will remain not affected by
the reflections. By that, ISI is would be minimized if a higher symbol period was used

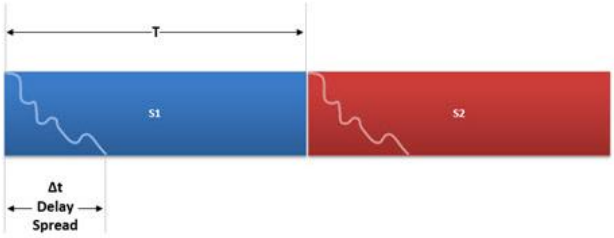


Figure 12-7 Symbols with time delay spread

That is why the ISI is minimized when we use a higher symbol period (or Lower Symbol Rate).

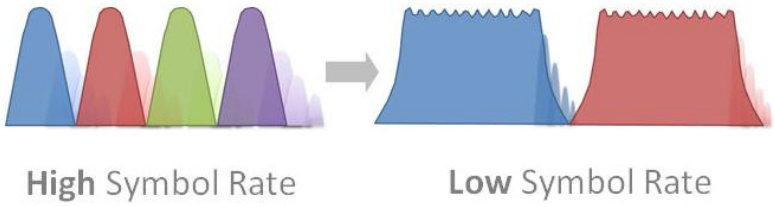


Figure 12-8 Symbol Rate

12.2 CYCLIC PREFIX INSERTION

The next key operation in the generation of an OFDM signal is the creation of a guard period at the beginning of each OFDM symbol x [k] by adding a Cyclic Prefix (CP), to eliminate the remaining impact of ISI caused by multipath propagation. The CP is generated by duplicating the last G samples of the IFFT output and appending them at the beginning of sympols Thus, the receiver can identify the end points of each symbol and correctly correlate the information, thereby eliminating the interference problem.

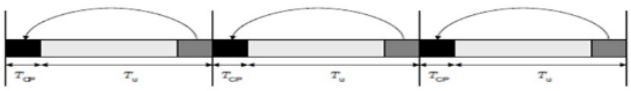


Figure 12-9 Cyclic Prefix Insertion

And what can we conclude about the CP? The higher the CP, the more 'useful' it is to eliminate the ISI, but also higher the full time of the symbol - it takes longer to transfer the 'same' information.
Furthermore, depending on the propagation conditions, maybe the part that would be 'lost' in the symbols is different. For example, in an urban environment, multiple components

(multipath) comes from relatively small distances - and the CP can be smaller. And also there's locations where components can come from greater distances until 4 km, as rural areas.

And what do you do to 'adjust' each case?
The solution found was to create different CP sizes. A larger, which could be applied in areas with higher probability of short distance multipath, and another complementary to be used in locations with probability of longer distance multipath.
We then have two sizes of CP defined by 3GPP:

Normal CP: with a duration of 4.7 microseconds. To be used in the first case described above. Equivalent to 7 symbols per slot. Note: In Normal CP, we have a different case, where we have 160 time slots for the first symbol (5.2 micro seconds) and 144 for the others (4.7 microseconds).

Extended CP: with a duration of 16.67 microseconds. Used in the second case. Equivalent to 6 symbols per
slot (or 16.7 microseconds).

Note: in general, the ranges of CP in OFDM systems ranges from 1/4 to 1/32 of the symbol period.

Then overall the CP The advantage is that it eliminates the ISI, and the disadvantage is that it reduces the number of symbols that can be transmitted in the same time interval.

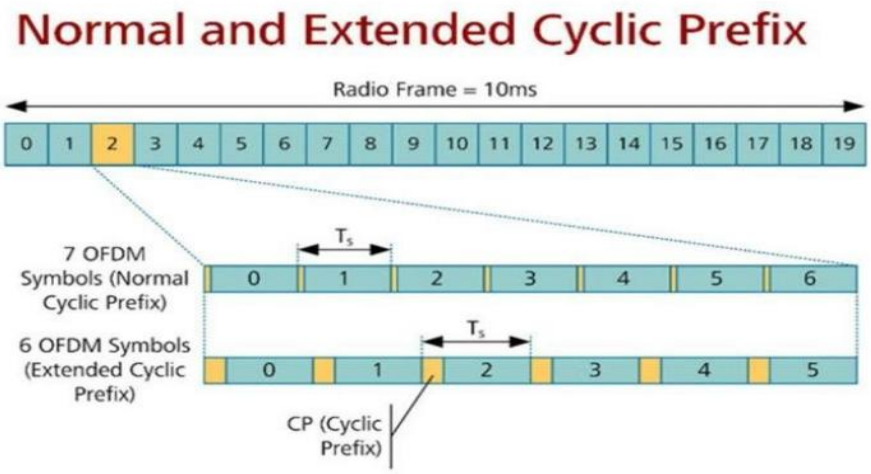


Figure 12-10 Normal and Extended Cyclic Prefix

12.3 OFDM (ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING)

Transmission by means of OFDM can be seen as a kind of multi-carrier transmission. But the basic characteristics of OFDM transmission, which distinguish it from a straightforward multi-carrier extension of a more narrowband transmission.

multi-carrier transmission :

to increase the overall transmission bandwidth, without suffering from increased signal corruption due to radio-channel frequency selectivity, is the use of so-called multi-carrier transmission. As illustrated in Figure below, multi-carrier transmission implies that, instead of transmitting a single more wideband signal, multiple more narrowband signals, often referred to as subcarriers, are frequency multiplexed and jointly transmitted over the same radio link to the same receiver. By transmitting M signals in parallel over the same radio link, the overall data rate can be increased up to M times. At the same time, the impact in terms of signal corruption due to radio-channel frequency selectivity depends on the bandwidth of each subcarrier. Thus, the impact from a frequency-selective channel is essentially the same as for a more narrowband transmission scheme with a bandwidth that corresponds to the bandwidth of each subcarrier.

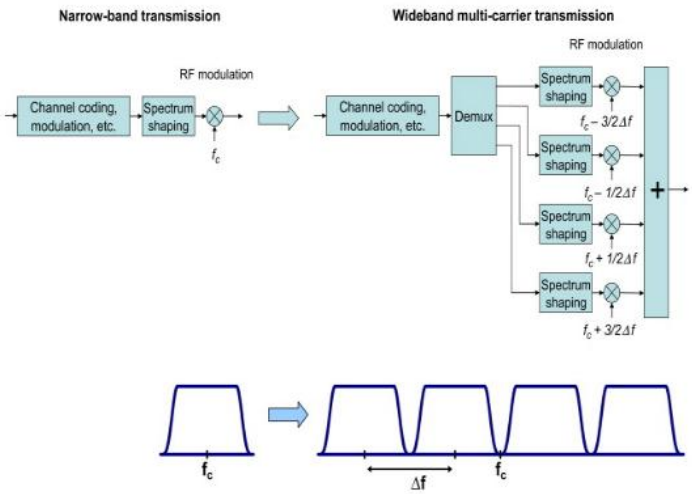


Figure 12-11 multi-carrier transmission

What makes OFDM different?

• The use of a typically very large number of relatively narrowband subcarriers. In contrast, a straightforward multi-carrier extension would typically consist of only a few subcarriers, each with a relatively wide bandwidth.
OFDM transmission may imply that several hundred subcarriers are transmitted over the

same radio link to the same receiver

• Simple rectangular pulse corresponds to a sin square-shaped per-subcarrier spectrum,

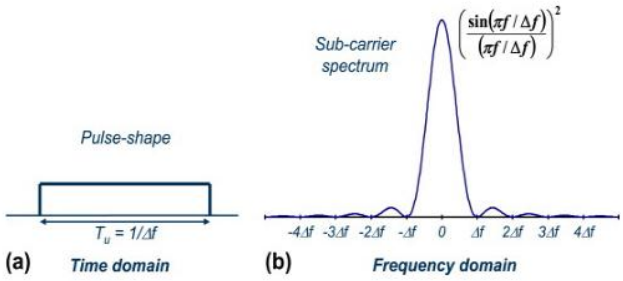


Figure 12-12 time and frequency domains

Tight frequency-domain packing of the subcarriers with a subcarrier spacing Df ¼ 1/Tu, where Tu is the per-subcarrier modulation-symbol time and The subcarrier spacing is thus equal to the per-subcarrier modulation rate 1/Tu.

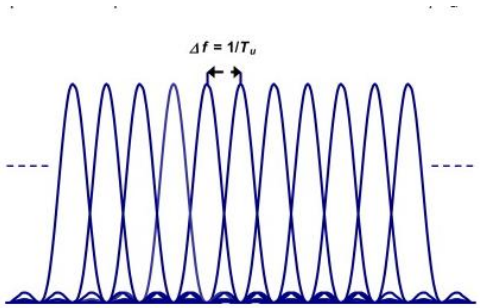


Figure 12-13 Orthogonally

12.4 ORTHOGONALITY

OFDM signal can be described as a set of closely spaced FDM subcarriers. In the frequency domain, each transmitted subcarrier results in a sinc function spectrum with side lobes that produce overlapping spectra between subcarriers which. This results in subcarrier interference but at orthogonally spaced frequencies. The individual peaks of subcarriers all line up with the nulls of the other subcarriers. This overlap of spectral energy does not interfere with the system's ability to recover the original signal. And The use of orthogonal subcarriers allows more subcarriers per bandwidth resulting in an increase in spectral efficiency.

12.5 OFDM BLOCK DIAGRAM

12.5.1 On the transmitter side

The transmission process includes mapping. After converting serial streams to parallel, the signal goes through IFFT operation. The signal to be sent through a wireless channel is then prepared by applying a parallel to serial convertor. Then, there is an insertion of a guard period to prevent information overlapping.

12.5.2 On the receiving side

The receiving process includes removal of cyclic prefix, then the series signal will be converted to parallel one. Afterwards, a FFT will be applied to the signal before the estimating stage using the channel estimator. To demodulate the signal using either QAM demodulator or PSK modulator, a parallel to serial converter is applied.

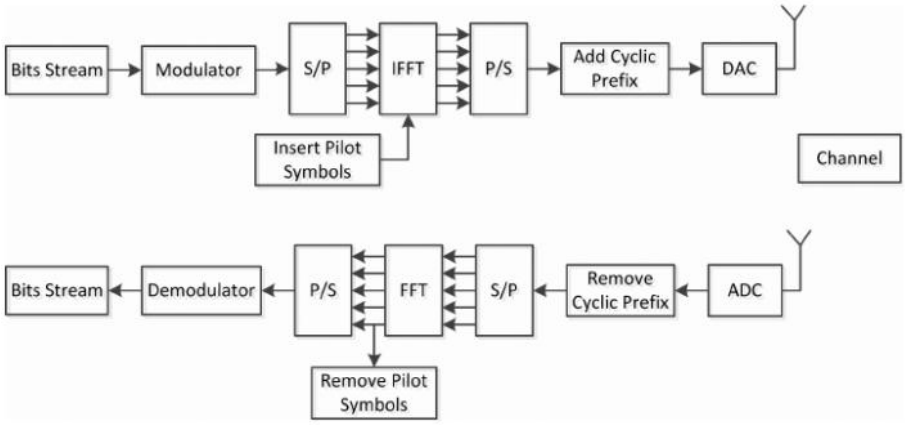


Figure 12-14 Block diagram of OFDM system

12.6 LTE IN DOWNLINK AND UPLINK

12.6.1 LTE OFDMA in the downlink

The OFDM signal used in LTE comprises a maximum of 2048 different sub-carriers having a spacing of 15 kHz. Although it is mandatory for the mobiles to have capability to be able to receive all 2048 sub-carriers, not all need to be transmitted by the base station which only needs to be able to support the transmission of 72 sub-carriers. In this way all mobiles will be able to talk to any base station.

Downlink carriers and resource blocks

Different LTE signal bandwidths will have different numbers of resource blocks.

LTE DOWNLINK CARRIERS AND RESOURCE BLOCKS						
Channel bandwidth (MHz)	1.4	3	5	10	15	20
Number of resource blocks	6	15	25	50	75	100

Table 12-1 LTE DOWNLINK CARRIERS AND RESOURCE BLOCKS

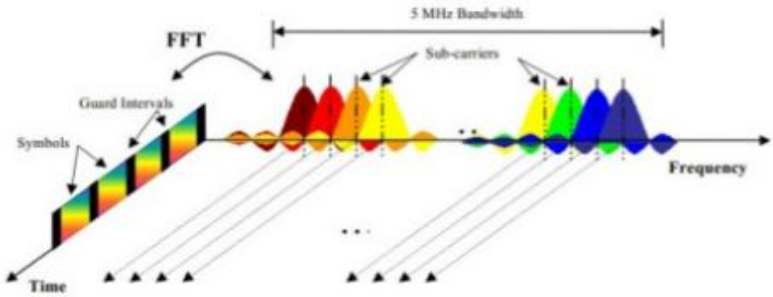


Figure 12-15 downlink in OFDM system

. OFDMA is used on the downlink, but since it presents a high Peak-to-average Power Ratio it is not possible to use it on the uplink. For the uplink SC-FDMA will be used.

12.6.2 LTE SC-FDMA in the uplink

LTE Uplink (SC-FDMA) is Similar to OFDM signal, but in OFDM, each sub-carrier only carries information related to one specific symbol in SC-FDMA, each sub-carrier contains information of ALL transmitted symbols.

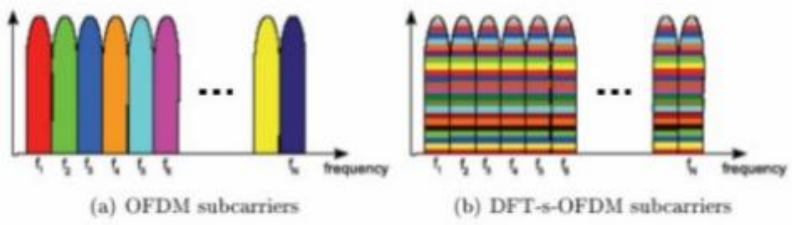


Figure 12-16 uplink in OFDM system

One of the key parameters that affects all mobiles is that of battery life. Even though battery performance is improving all the time, it is still necessary to ensure that the mobiles use as little battery power as possible.

With the RF power amplifier that transmits the radio frequency signal via the antenna to the base station being the highest power item within the mobile, it is necessary that it operates in as efficient mode as possible. This can be significantly affected by the form of radio frequency modulation and signal format.

Signals that have a high peak to average ratio and require linear amplification do not lend themselves to the use of efficient RF power amplifiers. As a result it is necessary to employ a mode of transmission that has as near a constant power level when operating. Unfortunately OFDM has a high peak to average ratio.

While this is not a problem for the base station where power is not a particular problem, it is unacceptable for the mobile. As a result, LTE uses a modulation scheme known as SC-FDMA - Single Carrier Frequency Division Multiplex. This combines the low peak to average ratio offered by single-carrier systems with the multipath interference resilience and flexible subcarrier frequency allocation that OFDM provides.

LTE Uplink (SC-FDMA) is Similar to OFDM signal, but in OFDM, each sub-carrier only carries information related to one specific symbol in SC-FDMA, each sub-carrier contains information of ALL transmitted symbols

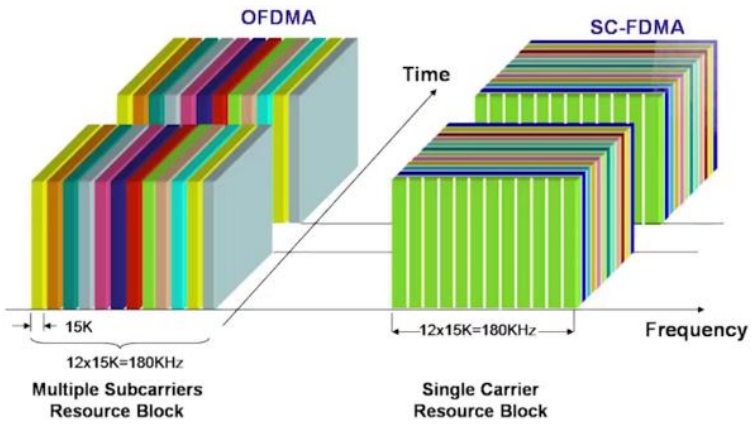


Figure 12-17 downlink and uplink in OFDM system

The most obvious difference between the two schemes is that OFDMA transmits the four QPSK data symbols in parallel, one per subcarrier, while SC-FDMA transmits the four QPSK data symbols in series at four times the rate, with each data symbol occupying a wider M x 15 kHz bandwidth.

12.7 OFDM ADVANTAGES & DISADVANTAGES

12.7.1 OFDM Advantages

1 - The OFDM spectrum is composed of overlapped narrow subcarriers. This makes efficient usage of frequency spectrum compare to traditional FDM method.
2 - In OFDM broadband channel is divided into smaller narrowband sub channels. This makes OFDM resistive to frequency selective fading. Moreover OFDM transmit/receive chain uses channel encoder/decoder and interleaver /deinterleaver which help in recovering lost OFDM symbols due to fading.
3 - OFDM makes use of cyclic prefix to eliminate ISI (Inter Symbol Interference) found in the multipath channel environment. Hence it is robust to multipath fading.
4 - Interference appearing on a channel may be bandwidth limited and in this way will not affect all the sub channels. This means that not all the data is lost.
5 - It is possible to allocate bandwidth as per resource requirements. Hence OFDM is bandwidth scalable technique.

6- Simpler channel equalization: One of the issues with CDMA systems was the complexity of the channel equalization which had to be applied across the whole channel. An advantage of OFDM is that using multiple sub-channels, the channel equalization becomes much simpler.

12.7.2 OFDM Disadvantages

1-High peak to average power ratio: An OFDM signal has a noise like amplitude variation and has a relatively high large dynamic range, or peak to average power ratio. This impacts the RF amplifier efficiency as the amplifiers need to be linear and accommodate the large amplitude variations and these factors mean the amplifier cannot operate with a high efficiency level.

2-it is prone to ISI (Inter Symbol Interference) and ICI (Inter Carrier Interference). This requires time offset and frequency offset correction algorithms.
3- As OFDM spectrum travels through multiple paths which require guard band to avoid ISI errors due to timing offsets.

OFDM, orthogonal frequency division multiplexing has gained a significant presence in the wireless market place. The combination of high data capacity, high spectral efficiency, and its resilience to interference as a result of multi-path effects means that it is ideal for the high data applications that have become a major factor in today's communications scene.

12.8 BRIEF OF THE CODE STEPS

Modulation Technique: QPSK
Number of symbols: 50
cyclic prefix ratio = 1/8 ;
Channel Models used for system simulation: (AWGN)

1 - We made generation for stream of bits of zeros and ones by command “randi” and used QPSK built in function to modulate the data

2 – Then we made block IFFT to convert the matrix to number then we can treat with it and make the last step which is Cyclic prefix that is coping a small part of the initial information to the end of each symbol so we made the end - number of cyclic prefix .

By the three steps :
a- convert the serial data to parallel to prepare the data for the IFFT operation
b– then applied IFFT operation by the built in function and used for loop
c- adding cyclic prefix

3-then last step at the transmitter to convert data from parallal to serial to transmit it

4- Applying the effect of channel – using AWGN built in function
5 - at receiver we removed cyclic prefix then we made FFT then we converted parallel to series

12.9 SIMULATION RESULTS

Using different FFT parameters

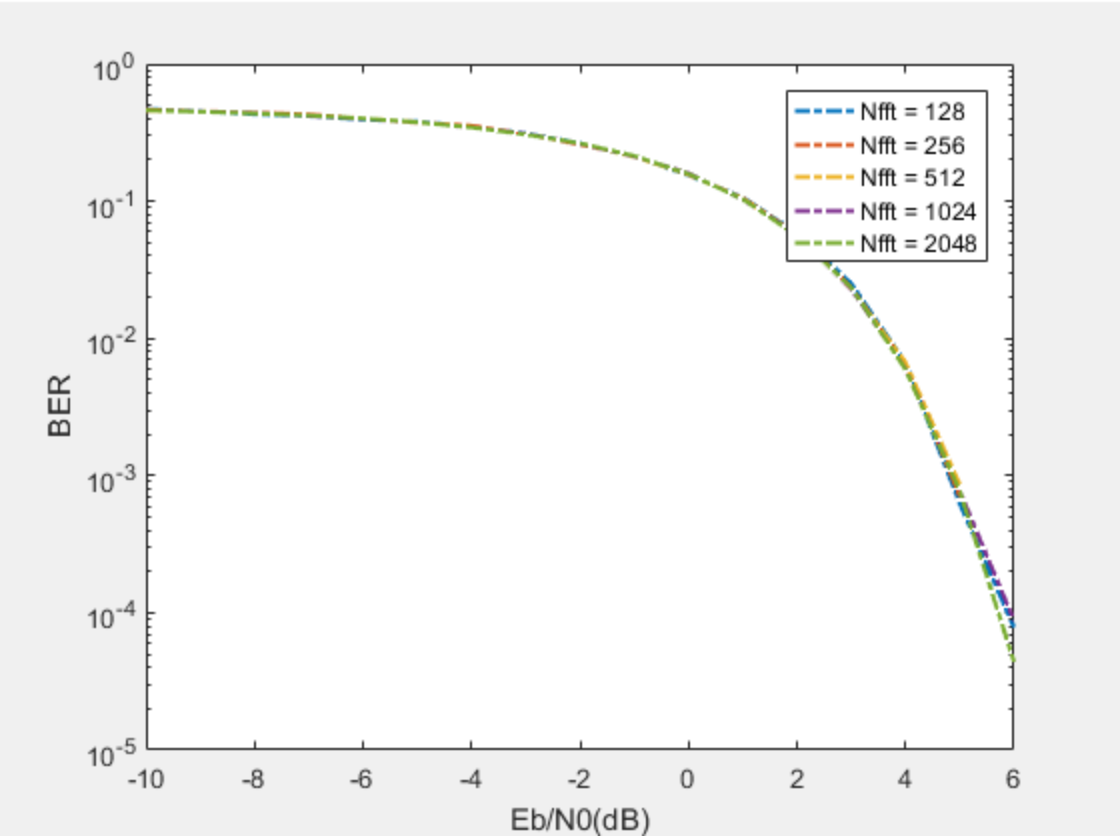


Figure 12-18 simulation results

Conclusion:

Using different FFT parameters will not affect the BER too much as shown

12.10 DIGITAL PART

12.10.1 FFT in OFDM system

The main part in the OFDM system is FFT(& IFFT),the role of the IFFT is in the next lines. A single carrier that occupies the entire bandwidth might suffer from frequency selective fading. The technique to get rid of the frequency selectivity is to divide the bandwidth in smaller parts and transmit multiple carriers (we can call these carriers as subcarriers). Lets assume that we have divided the bandwidth in N parts and want to transmit N parallel symbols with N subcarriers.

Now for multicarrier transmission (this is not OFDM yet), we need to modulate N-parallel BPSK/QPSK/QAM symbols to place them on N subcarriers that are operating on different parts of the total transmission bandwidth. We need a bank of N modulators in the transmitter for that purpose. Implementing a bank of N modulator or demodulator is not practical as it would be computationally very complex.
The same job of modulating the N-parallel subcarriers can be done by DFT/IDFT with much less complexity. OFDM is a multicarrier transmission technique where the subcarriers are orthogonal to one another. The use of DFT/IDFT made the OFDM practical and popular. As mentioned earlier, FFT/IFFT is a less complex form of DFT/IDFT.
Note that, the use FFT/IFFT is just implementation choices and not something mandatory. Nothing forbids the implementation of a bank of N modulators (though it would be impractical).

12.10.2 FFT algorithms

The basic principle of divide-and-conquer approach leads to a variety of efficient algorithms. As these algorithms improve the performance in terms of computation time, these are known as fast algorithms or fast Fourier transform algorithms. In this section, we discuss three major classes of FFT algorithms. The first class of algorithms is complex-valued FFTs, where the input of the sequence is complex valued. In the second class of algorithms, the input sequence is real-valued known as real-valued FFTs. At the end of this section, we discuss the special cases of the FFT.

- **Complex-Valued FFT Algorithms**

We discuss here some popular FFT algorithms to reduce the computational complexity and hardware implementation complexity. In this subsection, we discuss the popular FFT algorithm followed by some algorithms that can improve the computational speed and reduce the hardware complexity.

i. Radix-2 FFT Algorithms

The basic FFT algorithms are decimation-in-time (DIT) and the decimation-infrequency (DIF) radix-2 algorithms. These algorithms are applicable to compute the DFT of integer power-of-two lengths.

- Decimation-in-Time Radix-2 FFT Algorithm

This algorithm decomposes the time domain sequence $\{x(n)\}$ into successively smaller subsequence. Therefore, it is called as decimation-in-time algorithm .
The principle of radix-2 DIT FFT algorithm is illustrated in the following by considering $N = 2M$, where $M = 1, 2, 3, \ldots$. Since N is an even integer, the N-point input data can be split into two $(N/2)$-point subsequences $\{x1(n)\}$ and $\{x2(n)\}$,
Which correspond to the even- and the odd-indexed samples of the input $\{x(n)\}$, respectively, that is,

$x1(n) = x(2n)$
$x2(n) = x(2n + 1)$, $n = 0, 1, 2, \ldots, N/2 - 1$ (1)

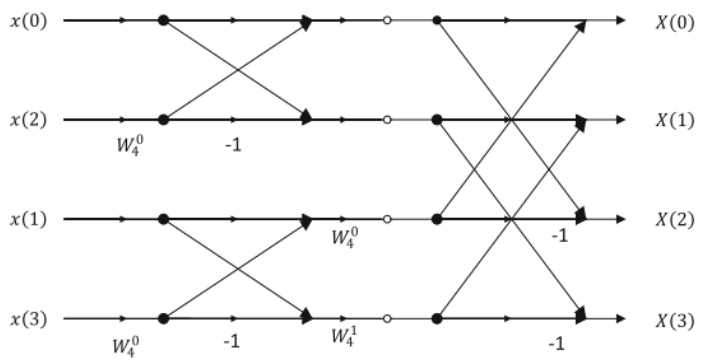


Figure 12-19 Length-4, DIT radix-2 FFT

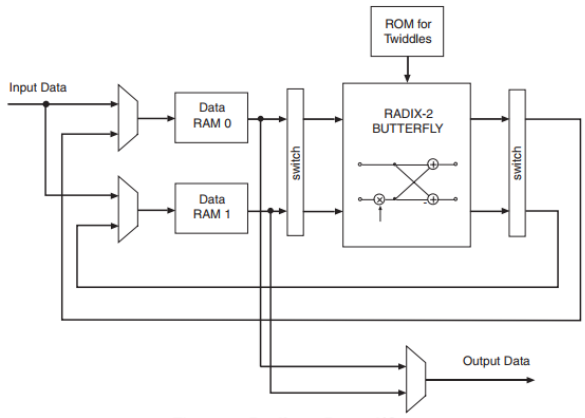


Figure 12-20 Radix-4 Burst i/o

Now the *N*-point DFT can be derived from two half-length DFTs by the decimation n-time process as follows:

$$
\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{(2n+1)k} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_{N/2}^{nk} \qquad (2)
\end{aligned}
$$

Similarly, the *(N/2)*-point DFTs can be computed from a pair of (*N*/4)-point DFTs.
The decimation process is continued till it contains only two-point DFTs. For a power of-two length sequences, decomposition of *N*-point DFT into 2-point DFTs could be completed in $M = \log 2\ N$ steps of decimation.
Figure 13.19 shows the decomposition of 4-point radix-2 DIT FFT using the simplified butterflies which involves two stages, each with two butterflies per stage. The input data to this are in bit-reversed order and the DFT output is in normal order.

- **Decimation-in-Frequency Radix-2 FFT Algorithm**

This algorithm is based on computing the DFT by decomposition of the sequence of DFT coefficients *X(k)*s into smaller subsequences, hence called as decimation-infrequency algorithm
In case of radix-2 DIF FFT, the DFT computation is split into two parts such that the first part involves the first *N*/2 data points and the second part involves the next *N*/2 data points.

ii. Radix-4 FFT Algorithms

It can be used when the DFT length *N* is a power-of-four (i.e., $N = 4M$). Unlike the radix-2 FFT algorithm in the radix-4 algorithm during every step, decimation is carried out by a factor of 4 .
In the first step of radix-4 DIT FFT, the input *N*-point data are split into four subsequences as *x(4n)*, *x(4n+1)*, *x(4n+2)*, and *x(4n+3)*,where $n = 0, 1, \ldots, N/4-1$. Then,

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{4}-1} x(4n) W_{N/4}^{nk} + W_N^{k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+1) W_{N/4}^{nk} \\ &\quad + W_N^{2k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+2) W_{N/4}^{nk} + W_N^{3k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+3) W_{N/4}^{nk} \end{aligned} \tag{3}$$

As the FFT length of a radix-4 is $N = 4M$, it requires $M = \log 4\ N = \log 22\ N$
stages of decimation where each stage involves *N*/4 butterflies. The radix-4 FFT butterfly structure is shown in Fig. 12.21 The decimation process of each stage is similar to radix-2 algorithm. Since $WN0 = 1$, each radix-4 butterfly involves three complex multiplications and eight complex additions. Therefore, the number of complex multiplications is $3N/4 \log 4\ N$. Comparing with the radix-2 approach, this requires less number of complex multiplications, although it uses the same number of complex additions. The total operation count for *N*-point radix-4 FFT is:

$$\begin{aligned} R_{\mathrm{M}} &= \frac{9N}{8} \log_2 N - \frac{43N}{12} + \frac{16}{3} \\ R_{\mathrm{A}} &= \frac{25N}{8} \log_2 N - \frac{43N}{12} + \frac{16}{3} \end{aligned} \tag{4}$$

(5)

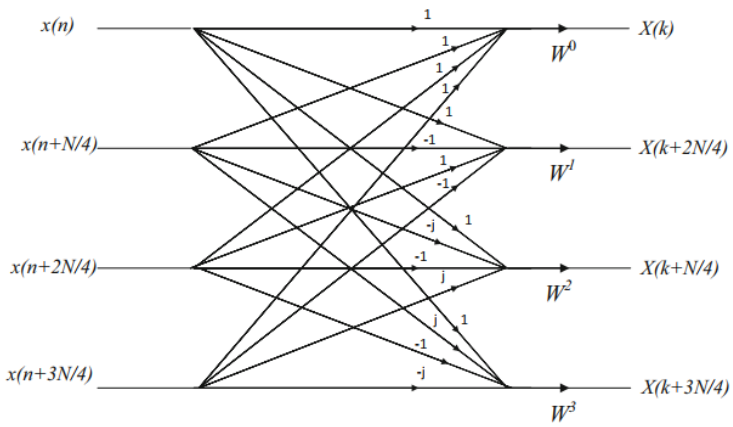


Figure 12-21 Radix-4 FFT butterfly

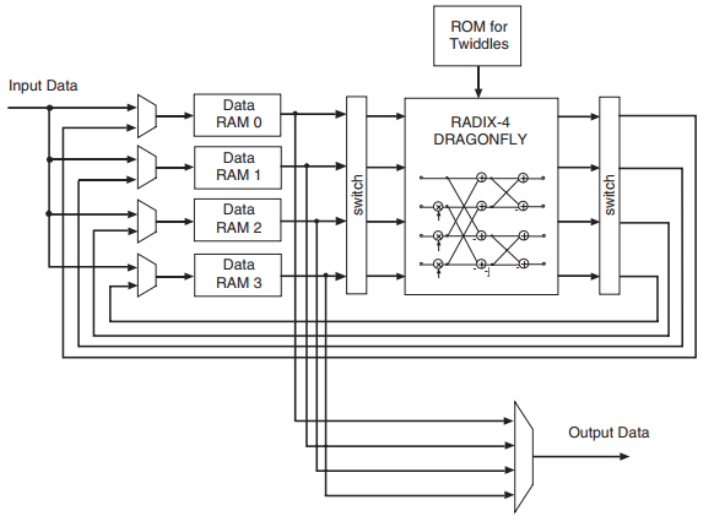


Figure 12-22 Radix-4, Burst I/O implementation

iii. Radix-2^I and higher radix FFT Algorithms

The twiddle factor multiplicative complexity can be reduced by using higher radices like radix-8 or radix-16. But, the implementation complexity grows as the radix becomes higher. In 1996, He and Torkeson discussed about radix-22 and radix-23 FFT algorithms. These algorithms have the same number of nontrivial1 multiplications as radix-4 and radix-8 algorithms, respectively. However, these algorithms differ in the twiddle factors at different FFT stages, but maintain the same butterfly structure of radix-2 algorithm. Followed by He and Torkeson, several radix-2*i*

algorithms are developed for higher radices that include radix-24, modified radix-24, radix-25, and modified radix-25 algorithms. The idea of these radix-2*i* algorithms is to get simpler butterfly structure with less multiplicative complexity. The following subsection explains the derivation of the radix-22 algorithm, which can be extended for higher radices.

iv. Split radix FFT Algorithms

The split-radix FFT (SRFFT) algorithm was introduced in, but this was clearly described in.
The split-radix algorithm decomposes an *N*-point DFT into one *N*/2-point DFT and two *N*/4-point DFTs as:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$
$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{4}-1} x(4n+1) W_{N/4}^{nk} + W_N^{3k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+3) W_{N/4}^{nk}$$

This algorithm makes use of both radix-2 and radix-4 (radix-2/4) behavior simultaneously on upper and lower half of the signal flow graph as shown in Fig. 12.23. The arithmetic complexity of SRFFT algorithm is given by :

$RM = N \log2 N - 3N + 4$	(6)
$RA = 3N \log2 N - 3N + 4$	(7)

The SRFFT algorithm requires less number of multiplications and additions compared to radix-2 and radix-4 algorithms. Followed by the SRFFT algorithm of, many split-radix algorithms were suggested to further reduce the number of complex multiplications and additions over the radix-2, radix-4 or any higher radix-based algorithms.

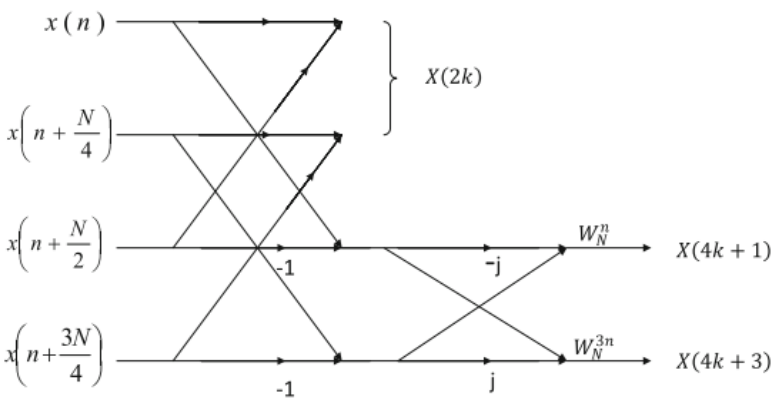


Figure 12-23 Split-radix FFT

12.10.3 Comparison between different FFT algorithms

N	Radix-2	Radix-4/ Radix-2^2	Split radix
16	24	20	20
32	88	–	68
64	264	208	196
128	712	–	516
256	1800	1392	1284
512	4360	–	3076
1024	10,248	7856	7172

Table 12-2 Number of real multiplications to compute a length-N complex DFT

N	Radix-2	Radix-4/ Radix-2^2	Split-radix
16	152	148	148
32	408	–	388
64	1032	976	964
128	2504	–	2308
256	5896	5488	5380
512	13,566	–	12,292
1024	30,728	28,336	27,652

Table 12-3 Number of real additions to compute a length-N complex DFT

Fig12.24 illustrates the trade-off of throughput versus resource use for the four architectures. As a rule of thumb, each architecture offers a factor of 2 difference in resource from the next architecture. The example is for an even power of 2 point size. This does not require the Radix-4 architecture to have an additional Radix-2 stage. All four architectures may be configured to use a fixed-point interface with one of three fixed-point arithmetic methods (unscaled, scaled or block floating-point) or may instead use a floating-point interface.

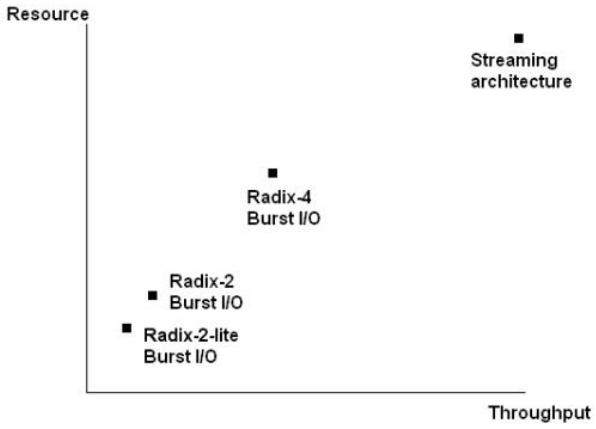


Figure 12-24 Resource VS Throughput for Architecture Options

Latency

Transform Length	Transform Cycles	Latency (µs)
128	821	3.284

Figure 12-25 Radix-2 latency

Latency

Transform Length	Transform Cycles	Latency (µs)
128	467	1.868

Figure 12-26 Radix-4 latency

Fig12.25 & fig12.26 illustrate the difference in the latency (the time between the start flag and the last output) between radix-2 & radix-4 algorithms.

12.10.4 HDL implementation of radix-2 FFT

Using xilinx FFT ip core(v7.1) I implemented the FFT block used in this project, in the next few pages I will illustrate how to use the FFT ip core then the results and simulation of the core.

A. Core GUI :

A. page 1:

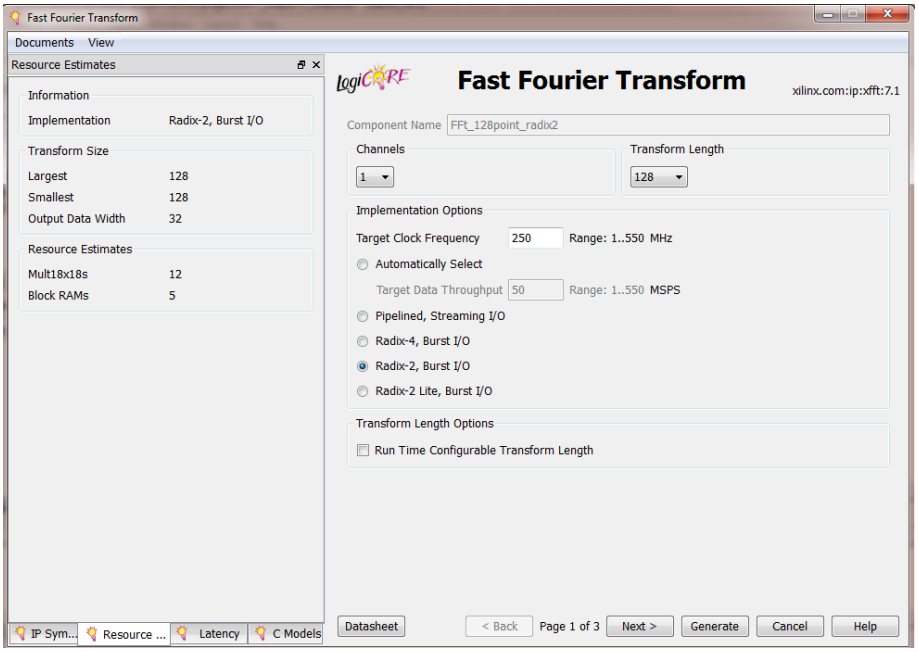


Figure 12-27 page1 in the GUI

- **Component Name**: The name of the core component to be instantiated. The name must begin with a letter and
  be composed of the following characters: a to z, A to Z, 0 to 9, and "_".

- **Channels**: Select the number of channels from 1 to 12. Multichannel operation is available for the three Burst
  I/O architectures.
- **Transform Length**: Select the desired point size. All powers of two from 8 to 65536 are available.
- **Implementation Options**: Select an implementation option
- **Transform Length Options**: Select the transform length to be run-time configurable or not. The core uses fewer
  logic resources and has a faster maximum clock speed when the transform length is not run-time configurable.

B. Page 2:

- **Data Format**: Select whether the input and output data samples are in Fixed Point format, or in IEEE-754

single precision (32-bit) Floating-Point format. Floating-Point format is not available when the core is in a multichannel configuration.

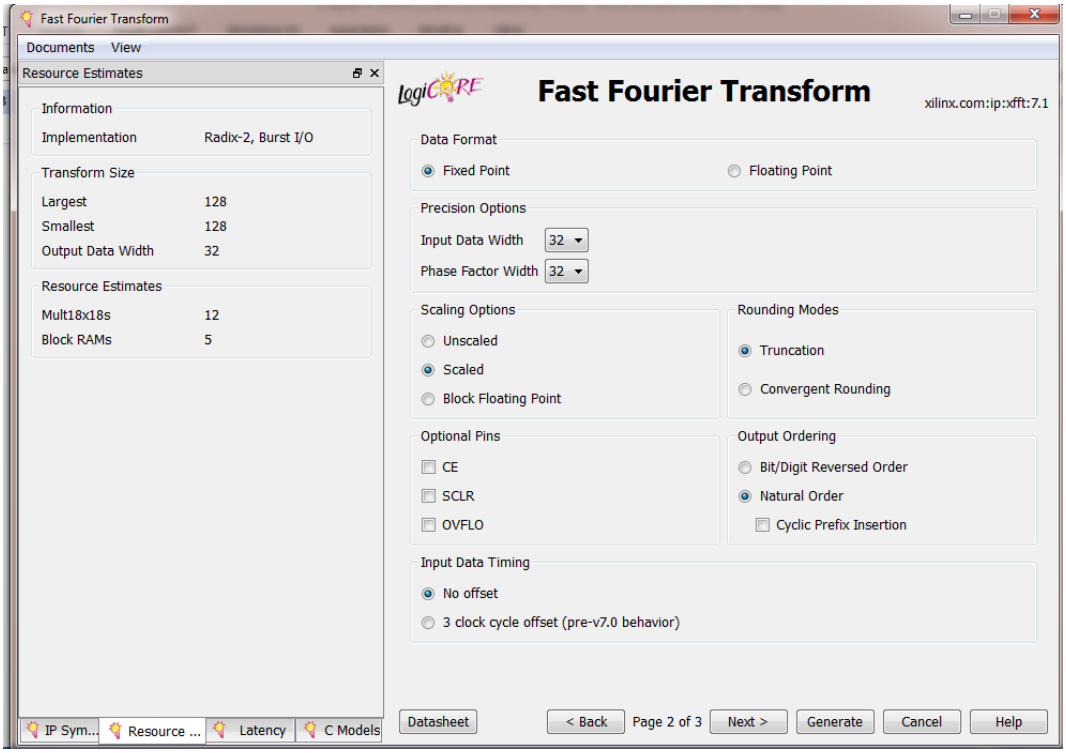


Figure 12-28 page2 in the GUI

- **Precision Options**: Input data and phase factors can be independently configured to widths from 8 to 34 bits, inclusive. When the Data Format is Floating-Point, the input data width is fixed at 32 bits and the phase factor width can be set to 24 or 25 bits depending on the noise performance required and available resources.

- **Scaling Options**: Three options are available, for all architectures.

- **Optional Pins**: Clock Enable (`CE`), Synchronous Clear (`SCLR`), and Overflow (`OVFLO`) are optional pins.
  Synchronous Clear overrides Clock Enable if both are selected. If an option is not selected, some logic
  resources may be saved and a higher clock frequency may be attainable.

- **Rounding Modes**: At the output of the butterfly, the LSBs in the data path need to be trimmed. These bits can be truncated or rounded using convergent rounding, which is an unbiased rounding scheme. When the
  fractional part of a number is equal to exactly one-half, convergent rounding rounds up if the number is odd, and rounds down if the number is even. Convergent rounding can be used to avoid the DC bias that would otherwise be introduced by truncation after the butterfly stages. Selecting this option will increase slice usage and yields a small increase in transform time due to additional latency.

- **Output Ordering**: Output data selections are either Bit/Digit Reversed Order or Natural Order. The Radix 2 based architectures (Pipelined, Streaming I/O, Radix-2, Burst I/O and Radix-2 Lite, Burst I/O) offer
  bit-reversed ordering, and the Radix-4 based architecture (Radix-4, Burst I/O) offers digit-reversed ordering.For the Pipelined, Streaming I/O architecture, selecting natural order output ordering results in an increase inmemory used by the core. For Burst I/O architectures, selecting natural order output increases the overall transform time because a separate unloading phase is required.

- **Input Data Timing**: In previous versions of the Xilinx FFT core, input data was applied three cycles after the corresponding sample index to allow a block memory containing data samples to be addressed. In many cases, this was not necessary, and applying data on the wrong cycle made it appear the core was functioning incorrectly. This timing may now be configured to be backwards-compatible with previous versions, or have no delay between sample index and applied data (default).

C. Page 3:

- **Memory Options**:
  - **Data And Phase Factors (Burst I/O architectures)**: For Burst I/O architectures, either block RAM or distributed RAM can be used for data and phase factor storage. Data and phase factor storage can be in distributed RAM for all point sizes up to and including 1024 points.

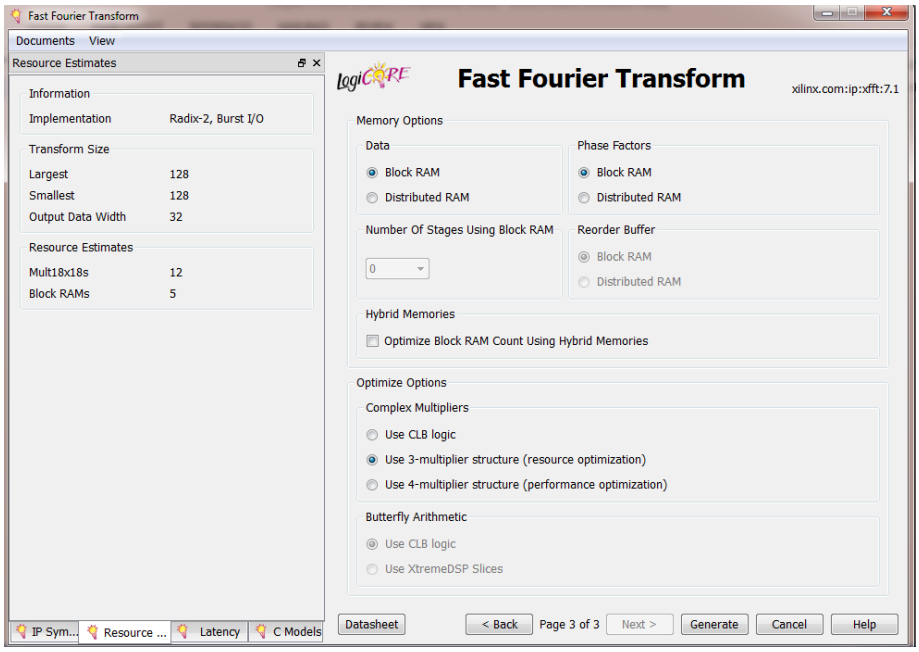


Figure 12-29 Page 3 of the GUI

- **Data And Phase Factors (Pipelined, Streaming I/O)**: In the Pipelined, Streaming I/O solution, the data
  can be stored partially in block RAM and partially in distributed RAM. Each pipeline

stage, counting from the input side, uses smaller data and phase factor memories than preceding stages. The user can select the number of pipeline stages that use block RAM for data and phase factor storage. Later stages use distributed RAM. The default displayed on the GUI offers a good balance between both. If output ordering is Natural Order, the memory used for the reorder buffer can be either block RAM or distributed RAM. The reorder buffer can use distributed RAM for point sizes less than or equal to 1024.

- **Optimize Options**:
  - **Complex Multipliers**: Three options are available for customization of the complex multiplier implementation:
    - **Use CLB logic**: All complex multipliers will be constructed using slice logic. This is appropriate for target applications that have low performance requirements, or target devices that have few XtremeDSP slices/Mult18x18s.
    - **Use 3-multiplier structure (resource optimization)**: All complex multipliers will use a three real multiply, five add/subtract structure, where the multipliers use XtremeDSP slices/Mult18x18s. This reduces the XtremeDSP slice/Mult18x18 count, but uses some slice logic. In Spartan-3A DSP, Spartan-6, and Virtex-6 devices, this structure can make use of the XtremeDSP slice pre-adder to reduce or remove the need for extra slice logic, and improve performance.
    - **Use 4-multiplier structure (performance optimization)**: All complex multipliers will use a four real multiply, two add/subtract structure, utilizing XtremeDSP slices/Mult18x18s. This structure yields the highest clock performance at the expense of more dedicated multipliers. In devices with XtremeDSP slices, the add/subtract operations are implemented within the XtremeDSP slices. In devices with Mult18x18s, the add/subtract operations use slice logic.
    -

D. **Information Tabs:**

- **Resource Estimates**:

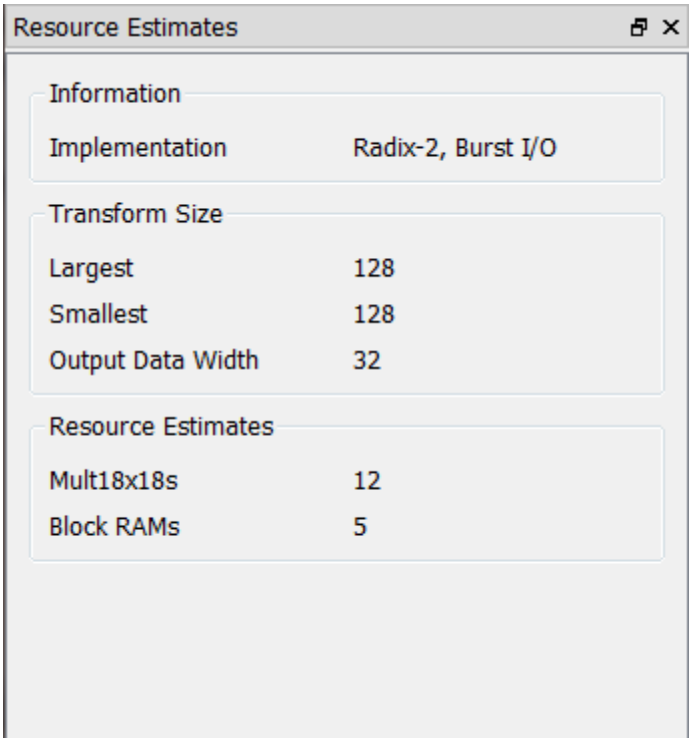


Figure 12-30 Resource Estimates tab in the GUI

- **Implementation**: This field displays the currently selected architecture. This is useful to see the result of automatic architecture selection.
- **Transform Size**: When the transform length is run-time configurable, the core has the ability to reprogram the point size while the core is running; that is, the core can support the selected point size and any smaller point size. This field displays the supported point sizes based on the Transform Length, Transform Length Options, and the Implementation Options selected.
- **Output Data Width**: The output data width equals the input data width for scaled arithmetic and block floating-point arithmetic. With unscaled arithmetic, the output data width equals (input data width + log2(point size) + 1).
- **Resource Estimates**: Based on the options selected, this field displays the XtremeDSP slice or Mult18x18 count and 18K block RAM numbers (9K block RAM numbers for Spartan-6 devices). The resource numbers are just an estimate. For exact resource usage, and slice/LUT-FlipFlop pair information, a MAP report should be consulted.

- **Latency:**

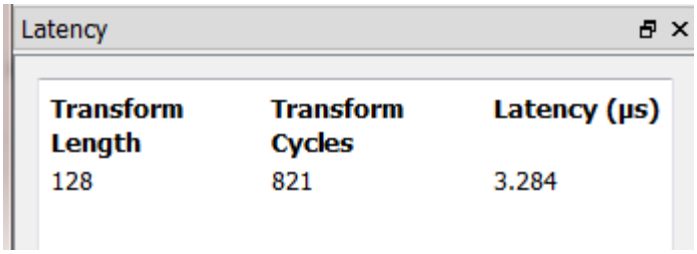


Figure 12-31 Latency tab in the GUI

- This tab shows the latency of the FFT core in clock cycles and microseconds (µs) for each point size supported. The latency is from asserting the START input to the last sample of output data coming out of the core, assuming that the UNLOAD input (if present) is asserted as soon as DONE goes High. Note that this is not the minimum number of cycles between starting consecutive frames, as frames may overlap in some cases. The latency in microseconds is based on the target clock frequency. The latency figures can be copied to the Clipboard and pasted as plain text into other applications.

- **C Model**:
  - This tab provides a link to the Xilinx LogiCORE IP FFT web page where the core C model can be
    downloaded. For details of the C model

B. Schematic:

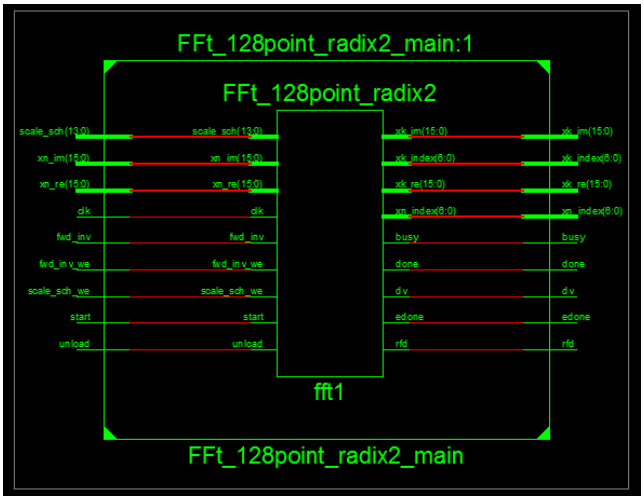


Figure 12-32 Technology schematic

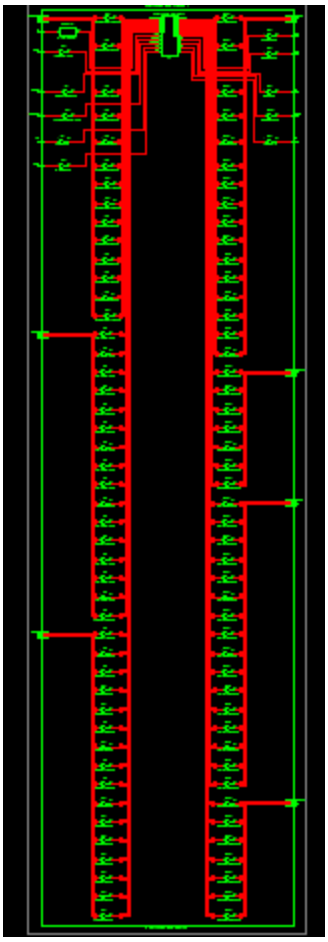


Figure 12-33 RTL schematic

C. Simulation:

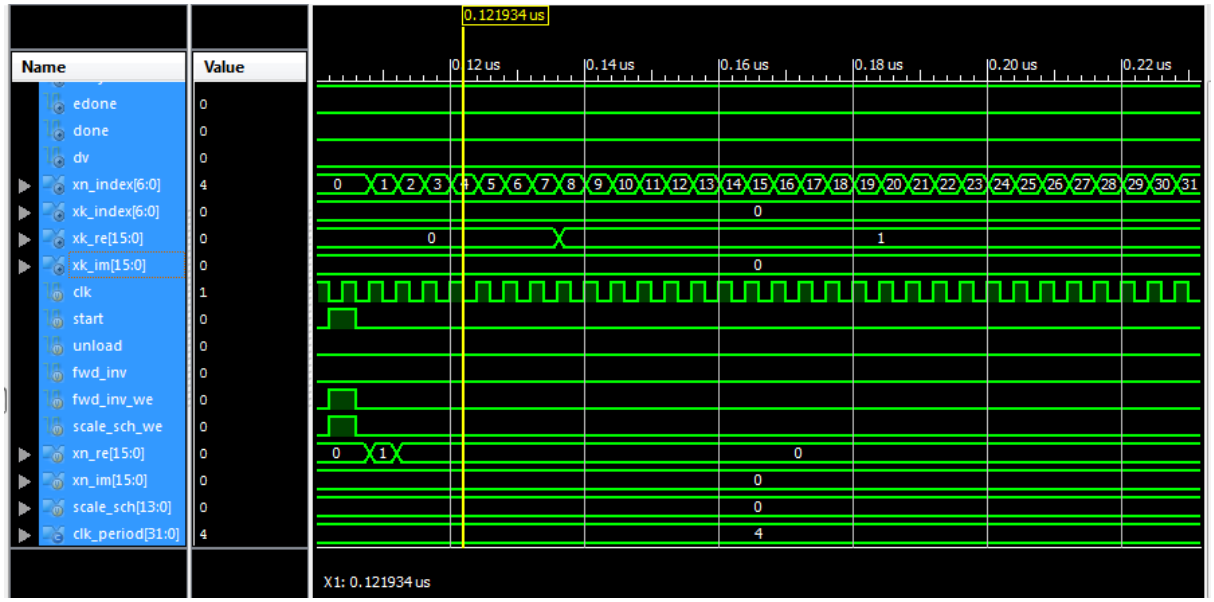


Figure 12-34 Simulation(1)

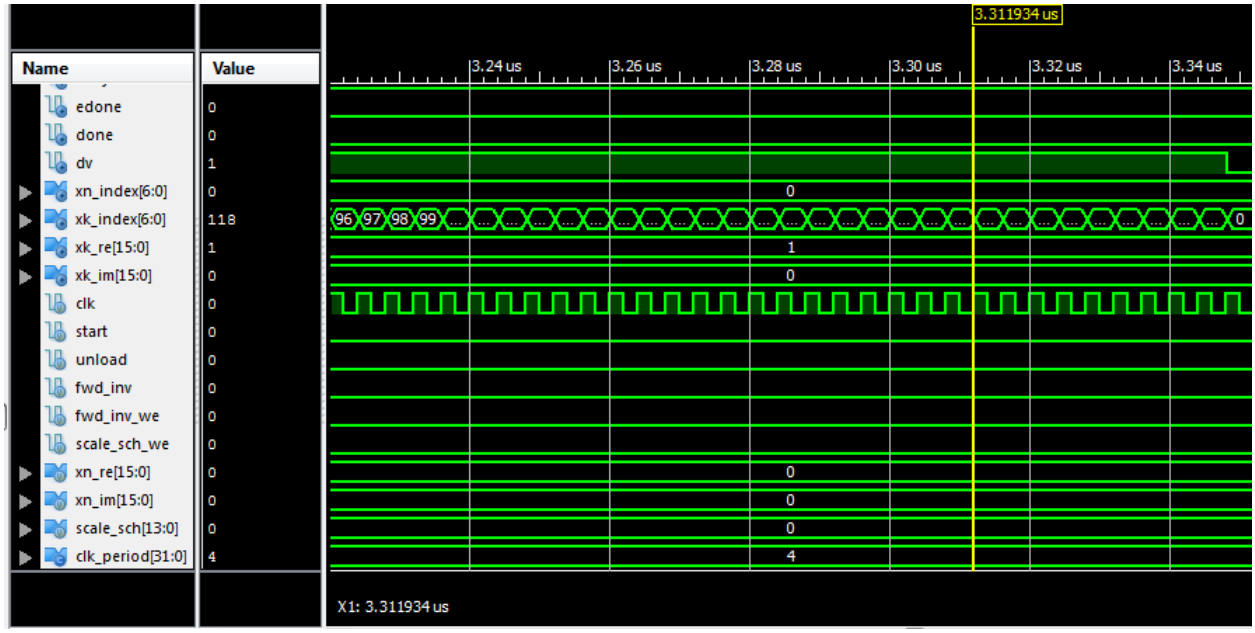


Figure 12-35 Simulation(2)

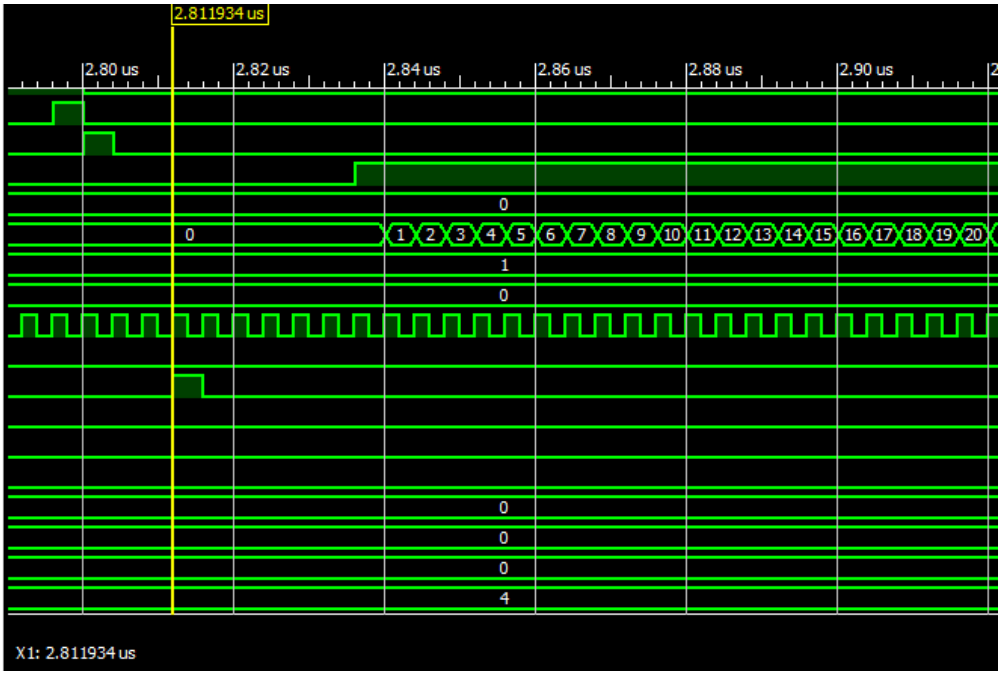


Figure 12-36 Simulation(3)

D. Setting the core for working

a) **Setting Up and Starting the Transform**

The START signal begins the data loading phase, which leads directly to the calculation phase. START is only valid when the core is idle or if the core is in bit reversed output mode and unloading the processed data.

b) **Applying Data**

Data is applied in a contiguous burst. The point at which data input should start relative to the START pulse is determined by the Input Data Timing parameter set in the GUI.
If "No offset" was selected for the Input Data Timing parameter, the input data (XN_RE, XN_IM) corresponding to the given XN_INDEX should arrive on the same cycle as the XN_INDEX it matches. The first data sample should therefore be applied as soon as RFD goes High, such that the first sample pair is read into the core on the first transition of XN_INDEX.
If "3 clock cycle offset" was selected for the Input Data Timing parameter, the input data (XN_RE, XN_IM) corresponding to the given XN_INDEX should arrive three clock cycles later than the XN_INDEX it matches . In this way, XN_INDEX can be used to address external memory or a frame buffer storing the input data.
RFD remains High with XN_INDEX during the loading phase and so indicates that data may be input.

c) **Data Processing**

BUSY goes High while the core is calculating the transform. DONE goes High when calculation is complete. EDONE goes High one cycle before that, that is, during the last cycle of the calculation phase.

d) **Data Output**

After the data is loaded and processed, two options are available to unload data:

• If Natural Order output order was selected, UNLOAD should be asserted or if cyclic prefix insertion is used) to output the data. During the unloading phase, while valid output results are present on XK_RE/XK_IM, DV (Data Valid) is High. During unloading, XK_INDEX corresponds to the XK_RE/XK_IM being presented. If cyclic prefix insertion is used, the cyclic prefix is unloaded first. CPV goes High to indicate that the cyclic prefix is being unloaded, and XK_INDEX counts from (point size) - (cyclic prefix length) up to (point size) - 1. After the cyclic prefix has been unloaded, or if the

cyclic prefix length is zero, or if cyclic prefix
insertion is not used, the whole frame of output data is unloaded. CPV goes Low (if present) and XK_INDEX
counts from 0 up to (point size) - 1. UNLOAD can be asserted any time from when EDONE goes High. UNLOAD is
ignored except when the core can begin unloading. In addition to using pulses, START and UNLOAD can be tied
High . In this case, the core continuously loads, processes, and unloads data. shows the
timing of entire frames. It does not show the small skews between signals which occur at the start and end of
frames and does not show the length of each phase of the transform to scale. The processing time may be much
longer than the time required to input or output a frame.
• If Bit/Digit-Reversed output order was selected, the user can assert START again . While the next
frame of data is loaded, the results are output at the same time. START can be asserted any time from when
EDONE goes High. If START is tied High, the core continuously loads/unloads then processes, loads/unloads
then processes, and so on .
DV remains High during data unloading in both cases.
There is a latency of several clock cycles after triggering an unload with UNLOAD or START before the output data
XK_RE/XK_IM is presented. This latency varies as a function of several core parameters, but the output data is qualified by DV (Data Valid) and XK_INDEX, so should be considered as a handshake.

E. Design report:

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	3,444	11,776	29%	
Number of 4 input LUTs	2,982	11,776	25%	
Number of occupied Slices	2,486	5,888	42%	
Number of Slices containing only related logic	2,486	2,486	100%	
Number of Slices containing unrelated logic	0	2,486	0%	
Total Number of 4 input LUTs	3,227	11,776	27%	
Number used as logic	2,469			
Number used as a route-thru	245			
Number used as Shift registers	513			
Number of bonded IOBs	152	372	40%	
Number of BUFGMUXs	1	24	4%	
Number of MULT18X18SIOs	12	20	60%	
Number of RAMB16BWEs	7	20	35%	
Average Fanout of Non-Clock Nets	2.10			

Figure 12-37 Device Utilization Summary

F. Code:

- Main code:

```
`timescale 1ns / 1ps

module FFt_128point_radix2_main (
clk, start, unload, fwd_inv, fwd_inv_we, scale_sch_we, rfd, busy, edone, done, dv,
xn_re, xn_im,
scale_sch, xn_index, xk_index, xk_re, xk_im
)/* synthesis syn_black_box syn_noprune=1 */;
input clk;
input start;
input unload;          //order to output thr results
input fwd_inv;         //Control signal that indicates if a forward FFT or an
inverse FFT is performed.
input fwd_inv_we;      //Enable for fwd_inv
input scale_sch_we;    //Scaling schedule enable
output rfd;                //Ready for data (Active High)
output busy;
output edone;
output done;
output dv;                 //Data valid (Active High
input [31 : 0] xn_re; //Input data(real part)
input [31 : 0] xn_im; //Input data(imaginary part)
input [13 : 0] scale_sch;////Scaling schedule
output [6 : 0] xn_index;//Input index
output [6 : 0] xk_index;//Output index
output [31 : 0] xk_re;//Output data(real part)
output [31 : 0] xk_im;//Output data(imaginary part)


//Port mapping the core to the user pins

FFt_128point_radix2 fft1(
.clk            (clk),
.start              (start),
.unload             (unload),
.fwd_inv        (fwd_inv),
.fwd_inv_we     (fwd_inv_we),
.scale_sch_we   (scale_sch_we),
.rfd                (rfd),
.busy               (busy),
.edone          (edone),
.done               (done),
.dv             (dv),
.xn_re          (xn_re),
.xn_im          (xn_im),
.scale_sch      (scale_sch),
.xn_index       (xn_index),
.xk_index       (xk_index),
.xk_re          (xk_re),
.xk_im          (xk_im)
);

endmodule
```

- Test bench code:

```
`timescale 1ns / 1ps

module radix2_tb;

    // Inputs
    reg clk;
    reg start;
    reg unload;
    reg fwd_inv;
    reg fwd_inv_we;
```

```
reg scale_sch_we;
reg [31:0] xn_re;
reg [31:0] xn_im;
reg [13:0] scale_sch;

// Outputs
wire rfd;
wire busy;
wire edone;
wire done;
wire dv;
wire [6:0] xn_index;
wire [6:0] xk_index;
wire [31:0] xk_re;
wire [31:0] xk_im;

parameter clk_period = 4;
integer i;

// Instantiate the Unit Under Test (UUT)
FFt_128point_radix2_main uut (
    .clk(clk),
    .start(start),
    .unload(unload),
    .fwd_inv(fwd_inv),
    .fwd_inv_we(fwd_inv_we),
    .scale_sch_we(scale_sch_we),
    .rfd(rfd),
    .busy(busy),
    .edone(edone),
    .done(done),
    .dv(dv),
    .xn_re(xn_re),
    .xn_im(xn_im),
    .scale_sch(scale_sch),
    .xn_index(xn_index),
    .xk_index(xk_index),
    .xk_re(xk_re),
    .xk_im(xk_im)
);
always begin
 clk = 1'b1;
#2; // high for 20 * timescale = 20 ns

clk = 1'b0;
#2; // low for 20 * timescale = 20 ns
end


initial begin
    // Initialize Inputs
    clk = 0;
    start = 0;
    unload = 0;
    fwd_inv = 0;
    fwd_inv_we = 0;
    scale_sch_we = 0;
    xn_re = 0;
    xn_im = 0;
    scale_sch = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here
    #((clk_period));
    scale_sch_we = 1;
  fwd_inv_we = 1;
   start = 1;
```

```
        #clk_period;
        scale_sch_we = 0;
        fwd_inv_we = 0;
        start = 0;
        #clk_period;
        for (i=1 ;i<128;i=i+1)begin
            xn_re = i;
            #clk_period;
        end

        xn_re = 0;
        xn_im = 0;
        #2700;
        unload = 1;
        #clk_period;
        unload = 0;

    end

endmodule
```

13 CHANNEL ESTIMATION

13.1 INTRODUCTION

In all communication the signal goes through a medium (called channel) and the signal gets distorted due to various noise is added to the signal while the signal goes through the channel. In needing to decode the received signal without much errors by removing the distortion and noise applied by the channel from the received signal. To do this, the first step is to figure out the characteristics of the channel that the signal has gone through. The technique to characterize the channel is called 'channel estimation'. Channel estimation is a vital part of receivers' designs used in mobile communication. Where it is required to compensate the distortion introduced at the symbols, As they travel through channel.

13.2 CHANNEL ESTIMATION OVERVIEW

The pilot-based channel estimation methods are based on the pilot symbols which are inserted among transmitted data subcarriers in fixed positions as shown in figure(12.1). Due to channel the received pilot symbols or sequence positions will be changed. The receiver has perfect knowledge of pilot sequence and the fixed pilot positions, so that the time-varying channel responses are tracked by interpolating a set of estimated obtained through periodically transmitted pilot symbols.

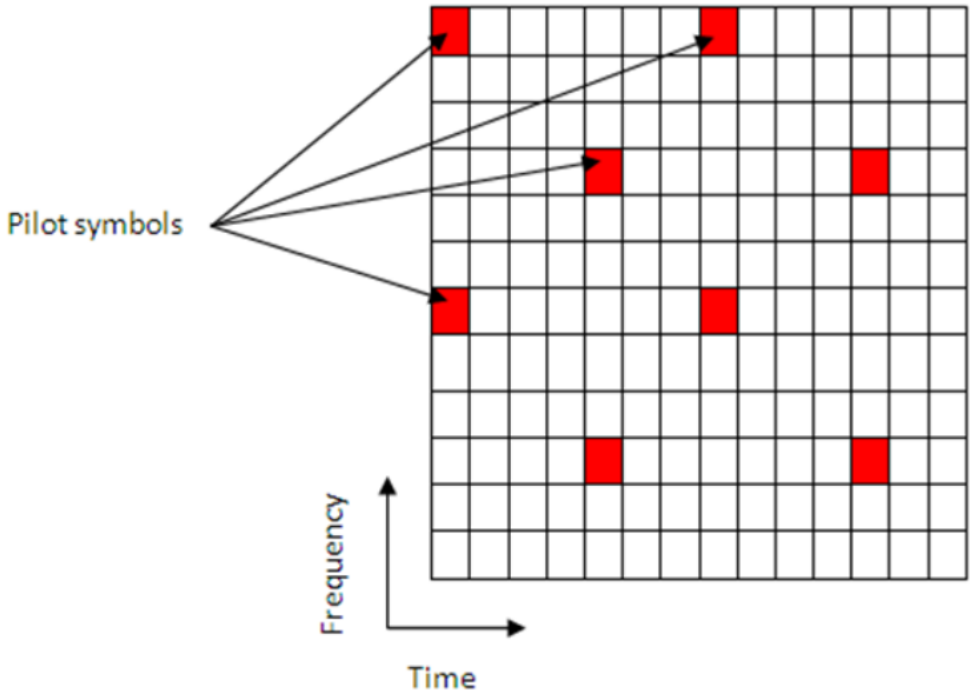
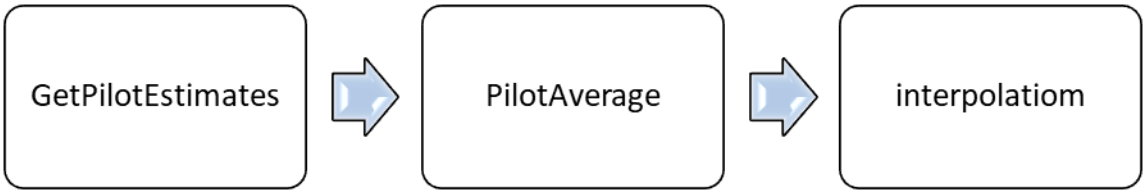


Figure 13-1 Reference signal allocation

13.3 PROCESS OF CHANNEL ESTIMATION BLOCK DIAGRAM



13.4 PILOT ESTIMATION TECHNIQUES

13.4.1 Least Square

The least square (LS) channel estimation is a simple estimation technique with very low complexity. It does not require any prior knowledge of the channel statistics. It is widely used because of its simplicity. However, it suffers from a high mean square error. We can estimate the channel response at the subcarriers by dividing the received version at the subcarrier by the known transmitted value.

$$\boldsymbol{Y} = \boldsymbol{XH} + \boldsymbol{V}$$

$$\begin{bmatrix} Y[0] \\ Y[1] \\ \vdots \\ Y[N-1] \end{bmatrix} = \begin{bmatrix} X[0] & 0 & \dots & 0 \\ 0 & X[1] & \dots & \vdots \\ \vdots & \dots & \ddots & 0 \\ 0 & \dots & 0 & X[N-1] \end{bmatrix} \begin{bmatrix} H[0] \\ H[1] \\ \vdots \\ H[N-1] \end{bmatrix} + \begin{bmatrix} V[0] \\ V[1] \\ \vdots \\ V[N-1] \end{bmatrix}$$

Where:

Y is the output vector.
X is diagonal matrix containing reference symbols.
H is the unknown channel frequency response coefficient vector to be estimated.
V is the noise vector.

The LS estimate of such system is obtained by minimizing the square distance between the received signal Y and the original signal X which gives the solution to the LS channel estimation as:

$$H_{LS} = X^{-1}Y$$

$$\begin{bmatrix} H[0] \\ H[1] \\ \vdots \\ H[N-1] \end{bmatrix} = \begin{bmatrix} \frac{1}{X[0]} & 0 & \dots & 0 \\ 0 & \frac{1}{X[1]} & \dots & \vdots \\ \vdots & \dots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{X[N-1]} \end{bmatrix} \begin{bmatrix} Y[0] \\ Y[1] \\ \vdots \\ Y[N-1] \end{bmatrix}$$

13.4.2 MMSE

LSE channel estimation method has been described which is computationally simple, but its performance is not good. Another method to estimate the CIR is minimum mean square estimator (MMSE) which has better performance than LS, but it is computationally complex. This method intends at the minimization of the mean square error between the exact and estimated CIRs.The CIR can be calculated using LMMSE in the following way:

$$\hat{H}_{MMSE} = R_{HH}(R_{HH} + \sigma^2 N(XX^H)^{-1})^{-1}\hat{H}_{LS}$$

Where:

X: The original reference signals in frequency domain (diagonal matrix)

H_{LS}: The estimated channel using Least Square method

H_{MMSE}: The estimated channel using Minimum Mean Square method

R_{HH}: the auto covariance matrix of H

σ^2_N: The noise variance

To get R_{HH} and σ^2_N:

$$R_{HH} = E\{(h - \mu_h)(h - \mu_h)\}$$

$$\sigma^2_N = va(n) = E\{(n - \mu_n)^2\}$$

13.5 PILOT AVERAGE SUBSYSTEM

It is used to minimize the effects of noise on the channel estimates, the least square estimates is averaged using an averaging window. This simple method produces a substantial reduction in the level of noise found on the pilot REs. The following two pilot symbol averaging methods are available.

- **'TestEVM'** — follows the method described in TS 36.141 [1], Annex F.3.4.
- **'UserDefined'** — allows you to define the window size and direction of averaging used on the pilot symbols plus other settings used for the interpolation.

We will focus here on 'UserDefined' method.

'UserDefined', allows the user to define the size of the averaging window, which direction averaging will be done in (time, frequency, or both) as shown in figure(12.2).

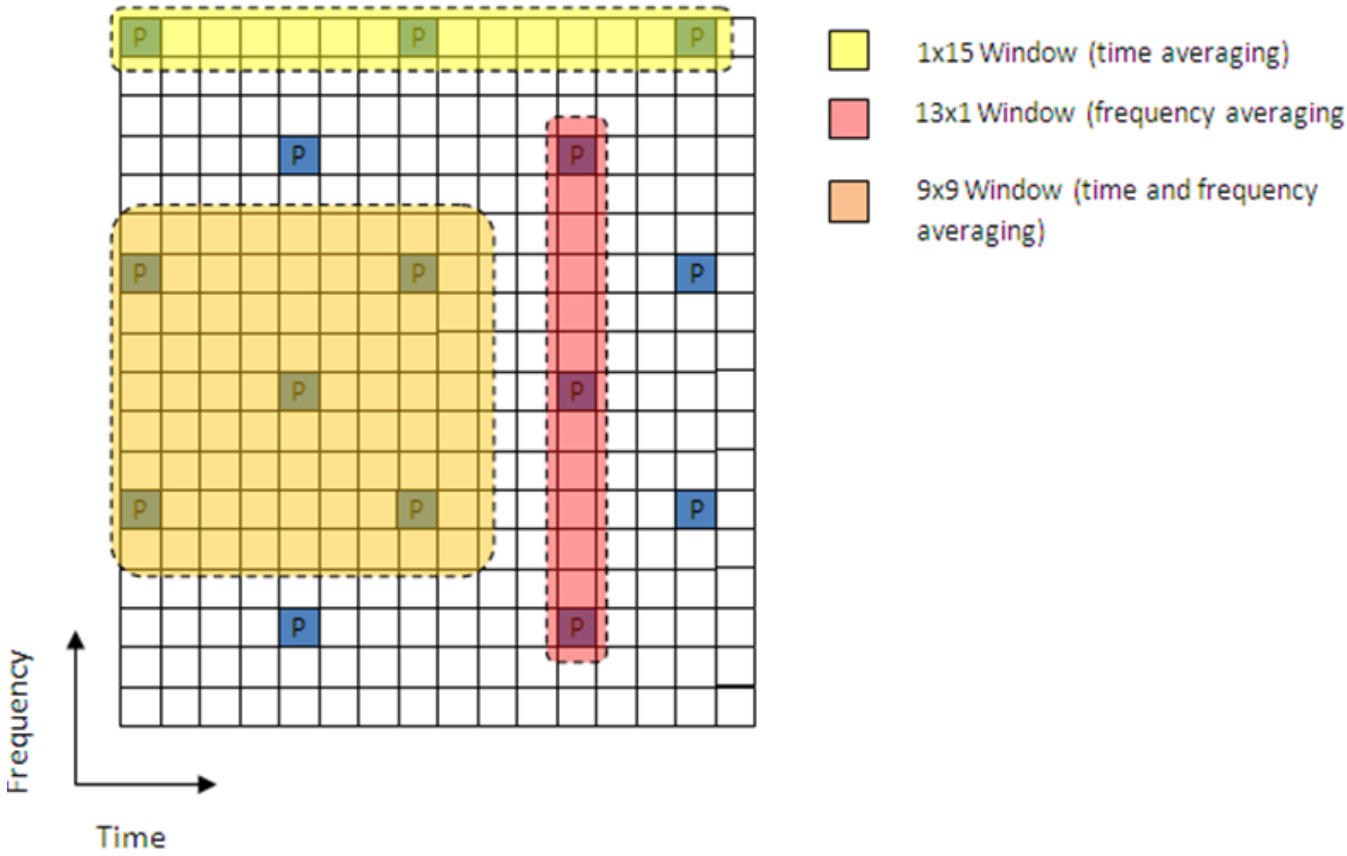


Figure 13-2 Averaging methods

13.6 INTERPOLATION

Interpolation is the process of deriving a simple function from a set of discrete data points so that the function passes through all the given data points (i.e. reproduces the data points exactly) and can be used to estimate data points in-between the given ones. So, after estimating the unknown channel frequency response coefficient at the position of the RS and doing time averaging for example as shown in figure (12.3), we use these values to get the other positions (that do not carry RS) by interpolation.

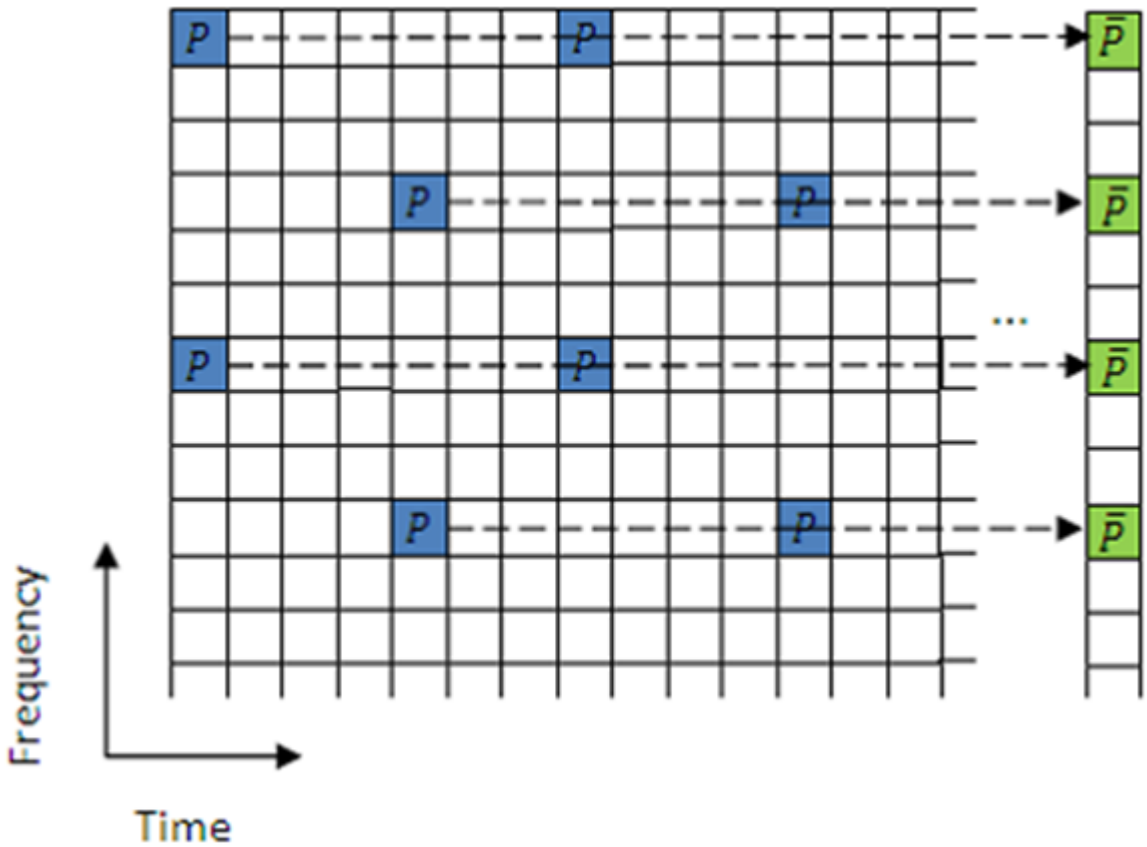


Figure 13-3 grid after doing time average

13.6.1 Linear interpolation method:

Linear interpolation technique makes use of the two successive pilot subcarriers in order to determine the channel response at the data subcarriers between these pilot subcarriers.

$$y = y_0 + (x - x_0)\frac{(y_1 - y_0)}{(x_1 - x_0)} \qquad (8.12)$$

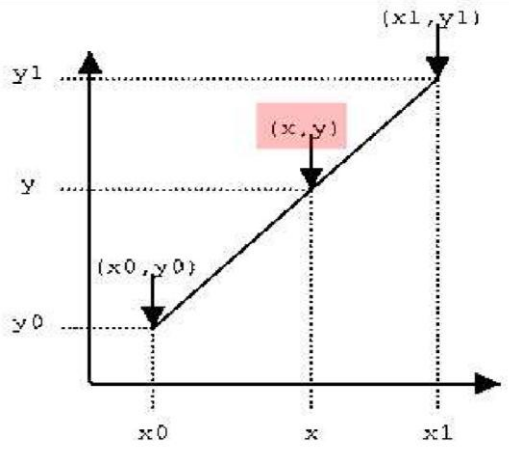


Figure 13-4 Linear interpolation example

It is given by

$$He(k) = He(mL + l) \qquad 0 \le l < L$$
$$= \left(Hp(m+1) - Hp(m)\right)\frac{l}{L} + Hp(m)$$

Where:

Hc(k) is channel response at data between two successive pilots

Hp(m) is channel response at pilot

Hp(m+1) is channel response at second pilot

m is refers to polit carrier number

L is frequency interval of inserted pilot

l is frequency interval between data resource block and pilot

13.7 THE DIGITAL PERSPECTIVE:

In this section we will illustrate how to implement each step in channel estimation and equalization in digital perspective as we use a Verilog “IEEE 1364-2001” HDL to describe the estimation block. Before going into the details of these steps we must find out the answers for some extra details that’s really matter in digital which are: What‘s the incoming input we deal with? Its pattern? Its data format?

And the most important question here is it supported (synthesizable) in Verilog or not? And if it not how to make it synthesis?

13.7.1 The Rx grid (input):

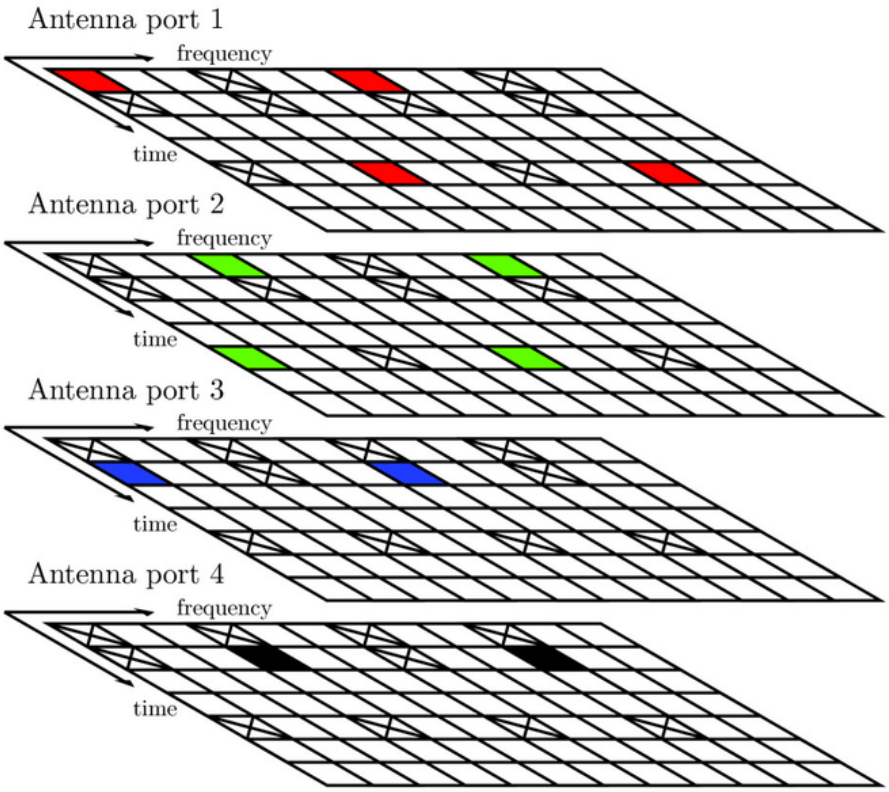


Figure 13-5 Rx Grid

The input that supposed to receive from previous blocks (OFDM demodulation) in LTE system is this 3D grid. The number of rows represents the number of subcarriers. The number of columns equals the number of OFDM symbols in a subframe. The number of subcarriers and symbols is the same for the returned grid from OFDM demodulation as the grid passed into OFDM Modulation in Tx. The number of planes (3rd dimension) in the grid corresponds to the number of receive antennas in MIMO case. And that's answer the first two questions what's exactly the input, and what's its pattern?
The next one is what's the data format in each resource element in OFDM grid? Data format here means how many bits in each resource element in grid and the data type these bits represent. To know about that we really need to have a good understanding of OFDM modulation and demodulation in Tx and Rx respectively and to make sure of that by checking the matlab and Simulink tool box of them. As at Tx the modulation is done by mapping a group of data bits to complex data symbols by using the modulation types supported by LTE then applies an OFDM mechanism to converts the resource grid

input samples to an equivalent time-domain signal output. At Rx the time domain waveform undergoes OFDM demodulation to transform it to the frequency domain and recreate a resource grid. Backing to our question that generated data is complex in digital that means the real and imaginary parts represents the In-phase and quadrature signals which in return represents the corresponding values of 1 or 0 in different modulation techniques (Qpsk,8psk,16psk). So, this bits either in in-phase or quadrature represents real values in range of (-1:1) and the number of bits depends on the ADC used in Rx which is 16 bits for our case in c700 SDR.

13.7.2 The limitations in Verilog HDL:

After knowing about data format and pattern of the incoming input remains for us to know is that kind of inputs is acceptable for Verilog. Actually, the Verilog HDL has two limitations associated to this input:

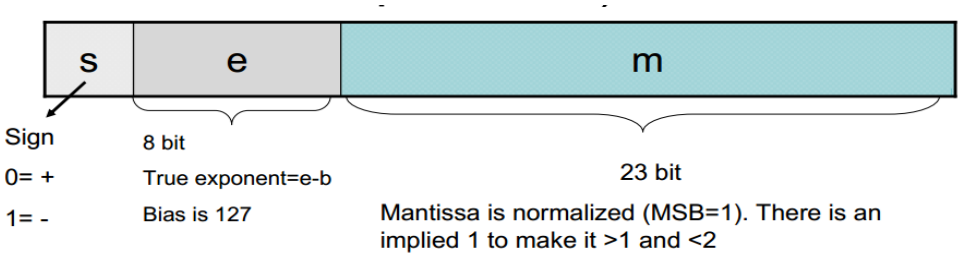
-Verilog accept only a vectors as input or output which means we can't pass multidimensional array (grid) from blocks (OFDM demodulation) to (Channel Estimation& Equalization).AS to pass it we should make it flatten as a one stream of bits with number of (subcarriers X OFDM symbols) words each word is 16 bit for In-phase (I) and quadrature (Q) which could really disturb the whole operation of extracting pilot symbols from grid later.

- The second limitation is pretty expecting which is verilog deals with any input bits as integer data type in other words it doesn't support the real values. Although It has a built-in function ($real) to be used in simulation only as a signals inside the module not even declared for inputs or outputs.

So we need to find out how overcome these limitations before starting in actual steps.

13.7.3 Real values:

Real numbers are the numbers which include both rational and irrational numbers. Rational numbers such as integers (-2, 0, 1), fractions (1/2, 2.5) and irrational numbers such as √3, π(22/7), etc., are all real numbers. To represent it as binary there's a two categories of formats: Fixed point and floating point. Real numbers are the numbers which include both rational and irrational numbers. Rational numbers such as integers (-2, 0, 1), fractions (1/2, 2.5) and irrational numbers such as √3, π(22/7), etc., are all real numbers. To represent it as binary there's a two categories of formats: Fixed point and floating point.



$$x = (-1)^s \times 1.m \times 2^{e-127}$$

Figure 13-6 (32 bit) single floating point format

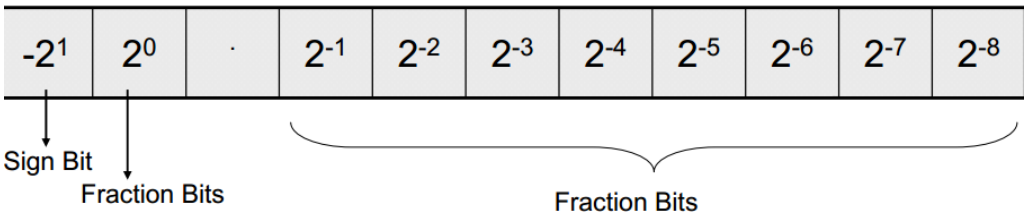


Figure 13-7 Q2.8 fixed point format

13.7.4 Arithmetic operations of Fixed point:

The main reason makes us interesting in Fixed point is to handle the previous mentioned limitation of Verilog in dealing with real data. Once we do any operations in the real data bits it will be corrupted. Besides being simpler to deal with it's not all fully unsupported in Verilog. Actually we can do the most popular operations like: addition, subtraction, shifting even the unsigned multiplication, so we just need to know how to convert it to a fully supported by defining the other desired operations in our block like: signed multiplication and signed division (Details in appendix).

Fixed point Multiplication steps:

- Check the sign of the two inputs.
- Convert any negative input into positive using 2's complement
- Do the usual integer multiplication in output with double numbers of word of the inputs.
- Then, if just one of two inputs was negative, do 2's complement again on the result as (+ * −) will be –ve .

- The last step is truncating the next word bits after the number of fraction bits from the doubled output word.

We tested this technique before using in our block with random inputs has the same range of expected received data in Rx grid where get a deviation on matlab calculation by (3.1999e-05).

Converting between Q format to another:

- To convert from higher fraction bit numbers to lower one just do signed shift left by:

$$F - (M - D)$$

  Where:

  F: number fraction bits in higher Q format.
  M: max possible fraction bits (word -1).
  D: number of fraction bits of the desired Q format

- To convert from lower fraction bit numbers to higher one just fix the MSB in the word and apply normal shift left on the rest of bits in the word.

Fixed point division steps:

- First, we need to extend the the integer range to accept values <1024 for the two inputs so convert the current Q format into Q(M-10).
- Then multiply the dividend only by 1000 which almost equals shifting left by 10 but.
- Round the dividend by any rounding methods then truncate the bits of integer part from the word.
- Apply any integer division technique on the rounded dividend and the converted divisor from first step.
- Finally, back the result to the very first Q format of inputs.

This technique has an acceptable accuracy with average deviation respecting to matlab calculation around (0.0516) for 16 bit word can be improve by bigger words. Anyway, the main problem is the overflow that’s happened if the integer bits range smaller than the result.

13.8 IMPLEMENTATION OF ESTIMATION:

13.8.1 Extract the Pilot from Rx Grid:

In Estimation we interest only in pilot or reference symbols (RS) so we need to detect its locations accurately to extract them from Rx grid then apply the estimation technique on them. These position depends on some important parameter and properties:

- The type of CP is extended or normal.
- Port number of Rx antenna (MIMO or SISO).
- The Cell ID
- There's 8 pilots in each resource block.
- Each 2 pilots exists in specific OFDM symbols and Subcarriers.
- The 2 Pilots in Frequency axis (Subcarriers) repeated each 6 resource element.
- RS is generated as QPSK (very useful in next Estimation step).

In case of (SISO) the pilot always exists in the first and middle OFDM symbols in each slot time in subframe (time axis) doesn't matter is it extended or normal CP so the locations at The RSs are existed in OFDM symbols 1, 5, 8 and 12. In case of extended cyclic prefix, The RSs are existed in OFDM symbols 1, 4, 7 and 10. While In the frequency axis the first position is detected by calculating the modulo 6 for the Cell ID.

Position = <Cell_ID>6 = Cell_ID mod 6

In case of 2 antennas (Port0 ,port1) the locations of RS is the same for time axis but in frequency axis is shifted by 3.

After detect which antenna port we received grid from we still face the first mentioned limitation in Verilog of flatting the (2D) grid into (1D) vector by applying this equation on The input I & Q stream to extract the desired word of RS:

$$LocationofExtractedRS = (Word * F) + S$$

Where:

Word: the number of bits represent the element in grid (16 bits).

F: subcarrier number where RS exists in resource block (0:11).

S: OFDM symbol number where RS exists in resource block (0:11 or 13) depends on CP.

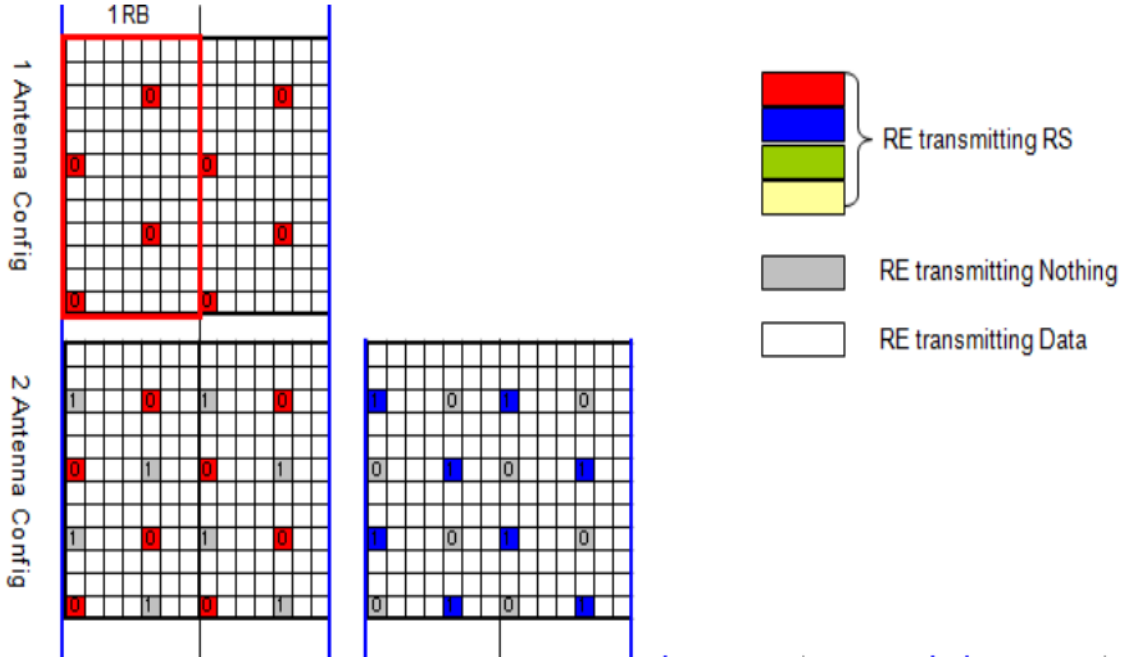


Figure 13-8 Locations of Reference signal

13.8.2 Applying the LSE algorithm:

To do Estimation for the error or changes in RX grid due to channel effect on the transmitting data we use the LSE algorithm which estimate the error by dividing the noisy extracted version of RS from the grid by its supposed value that known in Rx.

Channel response = Noisy RS / Expected RS

Anyway, it's a complex division so it will be like that for I & Q grids:

$$Iresponse = (ReNoisy * ReExp) + (ImNoisy * ImEXP)$$

$$Qresponse = (ImNoisy * ReExp) + (ReNoisy * ImEXP)$$

As we realized in the above 2 equations there's a missed **(Re_Exp^2 +Im_Exp^2)** divisor but it's not as the RS generated as QPSK so its real and imaginary values always equal (±1/√2) which always equal **one** in the divisor so no need to dividing and that's really a valuable for two reasons: first is the fact of being division is the most complicated and expensive operation among the others in FPGA and the other is the accuracy of division operation that we developed for fixed point data has an average error equals (correct calculation± 0·05) so we want to avoid using it as possible as we can.

13.8.2 Averaging and Interpolation:

As mentioned before the averaging is used to minimize the effects of noise on the channel estimates as If the estimated error itself is dc, and the noise component is random, then with each successive sample averaged the standard deviation of the original signal (DC error) divided by the square root of N (the number of successive samples).In our block we receive one resource block a time and applying time averaging then frequency averaging in each one we use 2 samples only so the final expected standard deviation of the estimated error decreases by factor equals 2.

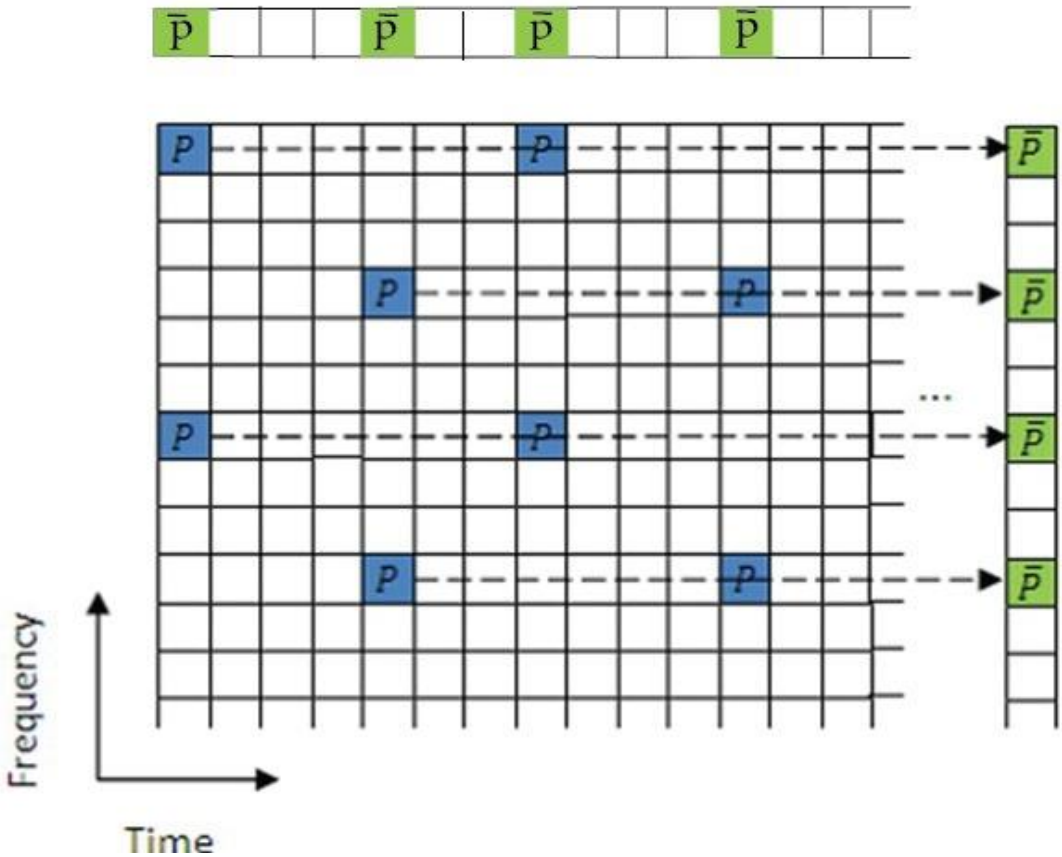


Figure 13-9 Time averaging and Frequency averaging

After the first averaging (time averaging) in one column vector we overwrite the new averaged values and replace the old estimated values with the new averaged then do the second averaging (frequency averaging) in one row vector then apply the linear interpolation to get the estimated errors for elements between the averaged pilots elements and the resulting vector is then replicated and used as the channel estimate for the entire received resource block grid.

13.9 EQUALIZATION:

After estimating the channel response by LSE technique by comparing the noisy and the known version of RS, now we need to recover the unknown data itself so we use the same concept in LSE technique we remove the effect of channel on data by dividing the noisy data by estimated channel response this time.

$$Recovereddata = Noisydata / Channelresponse$$

It's a complex division so for I & Q but Here we can't avoid using dividing.

$$Idata = ((ReNoisy * ReH) + (ImNoisy * ImH))/(ReH^2 + ImH^2)$$

$$Qdata = ((ImNoisy * ReH) - (ReNoisy * ImH))/(ReH^2 + ImH^2)$$

It's clear that's not a practical method to be executed in our block as we need here to do a division (subcarriers*OFDM symbols) times so we just settle for doing simulation it this step.

13.10 SIMULATION RESULTS:

13.10.1 Communication results:

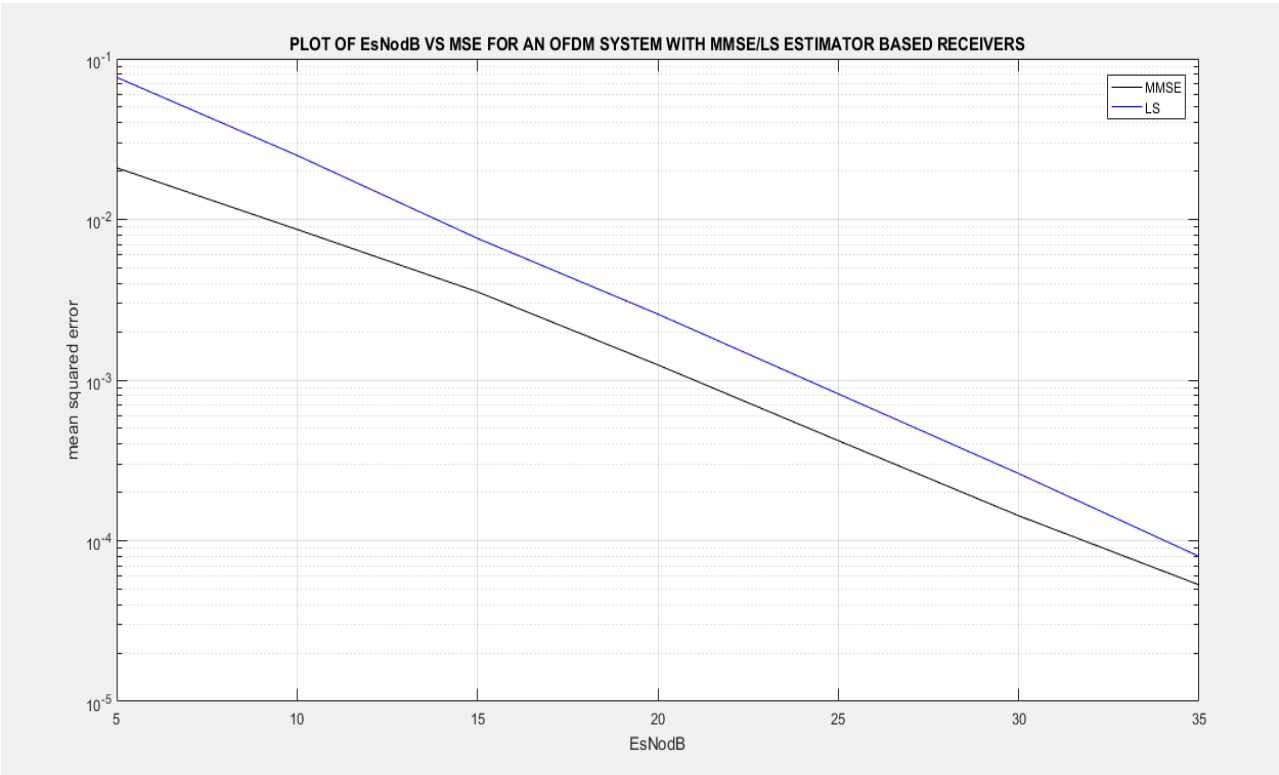


Figure 13-10 Comparison between LS and MMSE vs MSE

From the simulation: the MMSE has less mean error than LS, which means that the MMSE

Is better than LS. And it is expected, because in MMSE we take the noise power in our consideration, and when the EsNodB increase the two methods is simple and adequate.

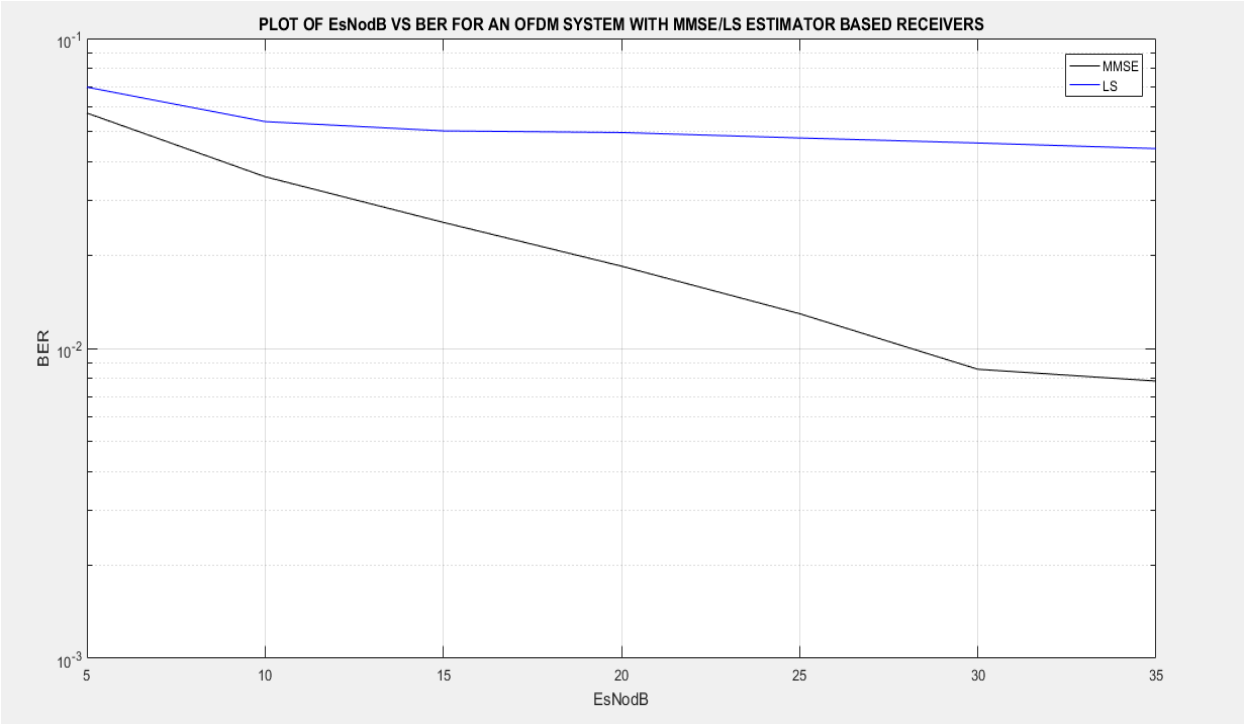
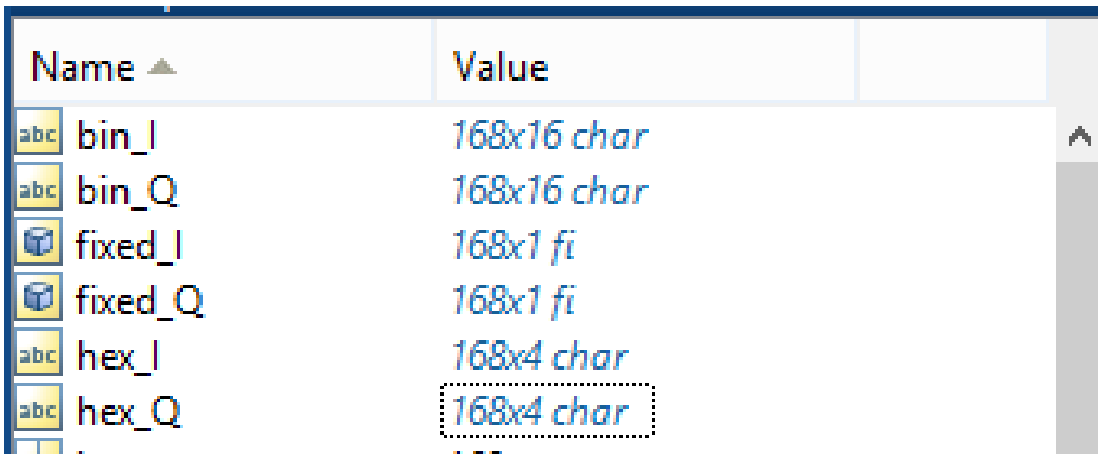


Figure 13-11 Comparison between LS and MMSE vs BER

From the simulation: As the value of EsNodB increases the difference between the BER by LS & MMSE methods also increases.

13.10.2 Digital results:

We tested the different blocks in our design with input random 16 bits Q14 format inputs generated by matlab in each step In the implementation as binary text file then pass it by function $readmemb using Xilinx ISE 14.7 to the test module.



Name	Value
bin_I	168x16 char
bin_Q	168x16 char
fixed_I	168x1 fi
fixed_Q	168x1 fi
hex_I	168x4 char
hex_Q	168x4 char

Figure 13-12 The generated random binary data from matlab

We start testing for the fixed point multiplier by comparing the output result from our blocks calculation and matlab calculation as shown below:

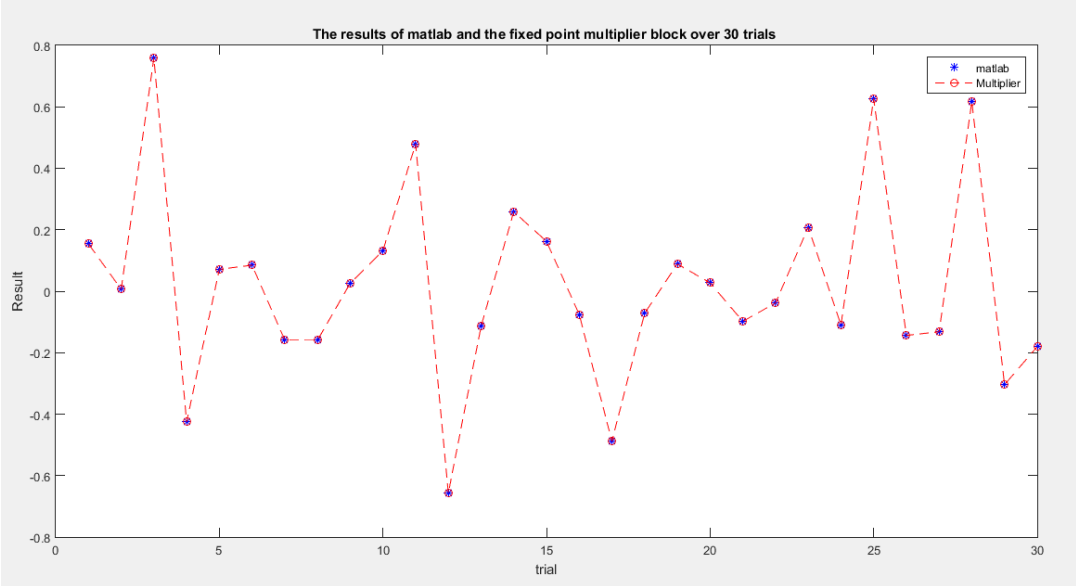


Figure 13-13 Multiplier Block calculation vs matlab calculation over 30 trials

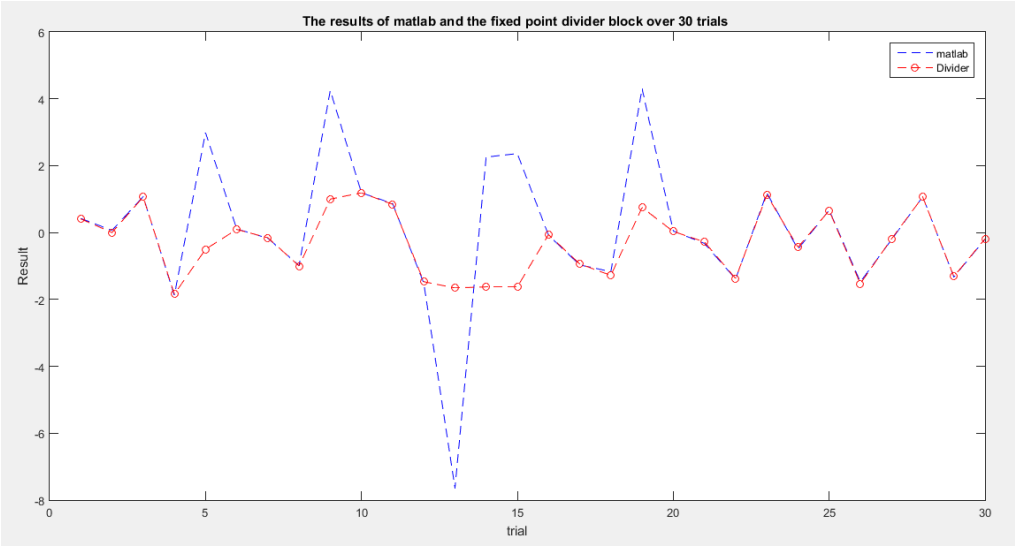


Figure 13-14 Divider block calculation vs matlab calculation over 30 trials

As mentioned before the deviation between the calculation in multiplying reaches to (3.1999e-05) and in dividing around (0.0516) without considering the overflow.

The next is the whole schematic of Estimation block generated by Quartus13.

Figure 13-15 Rtl Schematic of LSE Estimation block

The next is the primary timing and utilization reports for the estimation block (still working on enhancing it).

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 4.949ns (Maximum Frequency: 202.050MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 4.283ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Figure 13-16 Summary of timing report

```
                                  Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDR:C->Q              2   0.591   0.622  w_no_5 (w_no_5)
     LUT4_D:I0->O          2   0.704   0.482  w_no_cmp_eq000018 (w_no_cmp_eq000018)
     LUT3:I2->O          124   0.704   1.291  _old_busy_71 (_old_busy_7)
     FDE:CE                    0.555          Avg_I_5_0
    ----------------------------------------
    Total                      4.949ns (2.554ns logic, 2.395ns route)
                                       (51.6% logic, 48.4% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 124 / 124
-------------------------------------------------------------------------
Offset:              4.283ns (Levels of Logic = 1)
  Source:            Avg_Q_13_15 (FF)
  Destination:       Q_out<223> (PAD)
  Source Clock:      clk rising

  Data Path: Avg_Q_13_15 to Q_out<223>
                                  Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q              1   0.591   0.420  Avg_Q_13_15 (Avg_Q_13_15)
     OBUF:I->O                 3.272          Q_out_223_OBUF (Q_out<223>)
    ----------------------------------------
    Total                      4.283ns (3.863ns logic, 0.420ns route)
```

Figure 13-17 Detailed delay in gates in timing report

The next report is the utlization report:

```
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 3s500efg320-4
 Number of Slices:                       71  out of   4656     1%
 Number of Slice Flip Flops:            133  out of   9312     1%
 Number of 4 input LUTs:                133  out of   9312     1%
 Number of IOs:                         481
 Number of bonded IOBs:                 449  out of    232   193% (*)
 Number of MULT18X18SIOs:                 2  out of     20    10%
 Number of GCLKs:                         1  out of     24     4%
WARNING:Xst:1336 -  (*) More than 100% of Device resources are used
---------------------------
Partition Resource Summary:
---------------------------
  No Partitions were found in this design.
---------------------------
=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.
```

Figure 13-18 the summary of utilization report

The last simulation results for Equalization shows the (12*14) I &Q recovered data grid in ISE simulation.

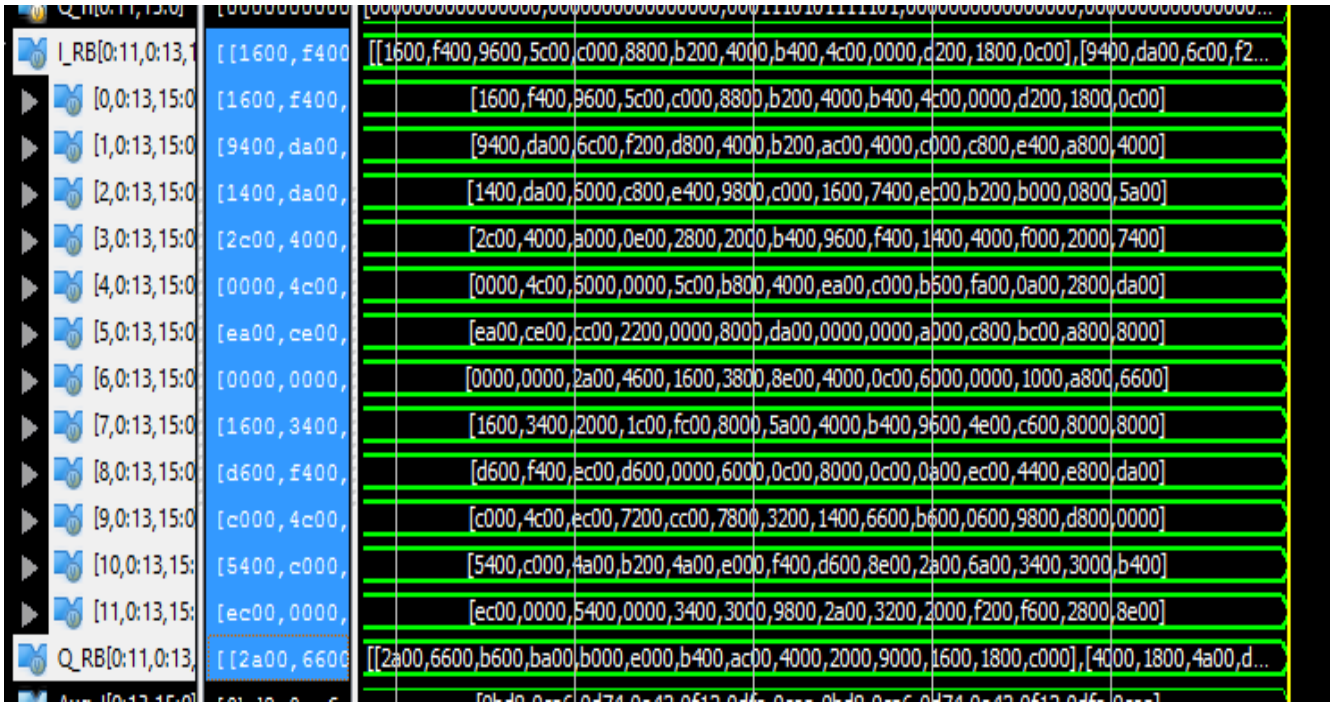


Figure 13-19 the simulation of the recovered data in Equalization block

14 CONCLUSION AND FUTURE WORK

14.1 COMMUNICATION TEAM

This project consists on analysing and working with a MATLAB code which simulates an LTE Scanner in order to extract the Master Information Block (MIB).

We are described a brief overview of the LTE network architecture and investigated in LTE Physical Layer to know all parameters that's we used in LTE system.

We have discussed the procedure of extract MIB that consists of Frequency synchronizations, Cell Search and Time Synchronization, OFDM Demodulation and Channel Estimation.

We investigated in each procedure individual in order to study the main parameter and the effect of the noise in a simulation of MATLAB on the BER. As we have

Frequency synchronizations:

Carrier frequency synchronization, Sampling frequency synchronization, effect of CFO, simulation results showed that CFO violates the orthogonality principle, and hence introduce ICI, decreases the SNR of the signal and increases the bit error rate. We used The CP based estimation technique to improve the bit error rate, and compare the result with The Symbol based (P.H. Moose) estimation technique in order to ger the best performance.

Cell Search and Time Synchronization:

Primary synchronization signals (PSS), Secondary synchronization signal (SSS), Timing synchronization (time slot synchronization, Frame synchronization) and Differences between FDD & TDD.

OFDM:

Architecture of OFDM, LTE OFDMA in the downlink, LTE SC-FDMA in the uplink, OFDM Advantages & Disadvantages and simulation results of BER with changing the size of FFT.

Channel Estimation:

Channel Estimation Overview, comparison of Pilot Estimation Techniques (Least Square, minimum mean square estimator (MMSE))

Our future work is to decode System Information Blocks (SIBs) and We plan to implement more of estimation techniques for CFO, STO and channel estimation to get improvement of the signal in receiver in order to get which technique has the best performance among the other techniques.

14.2 SDR TEAM

THEN we want to build LTE Scanner using hardware based on SDR. We have many tracks to get that target. At first we work on C700 to act as standalone receiver and connect it to PC at which the signal can be analyzed and decoded; so in this part we generate a specific signal using LTE waveform generator in MATLAB and pass it through all the sequence above then let the RX module of C700 to receive it , finally we decode the acquired samples to extract MIB.

We could connect an omnidirectional antenna to C700 to receive LTE downlink signal and then pass it through the same sequence to extract MIB for the available cells.

We work on other hardware device; RTL-SDR connected to Evrytania on Linux to act as a scanner; we can select a specific range of frequency for search or let the device scan the all available cells.
Here we use commands to know cell ID for each detected cell, Number of antenna ports, power of each port, Number of resource blocks and more.

14.1 DIGITAL TEAM

As a digital team, we implemented different blocks inside the scanner including correlators needed for synchronization, FFT block needed in the OFDM demodulation, build fixed point multiplier and divider, and channel estimation and equalization blocks needed in error detection and correction.

Our future work is to create an actual prototype of the scanner by completing the missing blocks and integrating the whole scanner blocks and trying it on The FPGA with real lte signals.

REFERENCES

[1] Abdul Gani Abshir, M. M. Abdullahi, Md. Abdus Samad, "A Comparative Study of Carrier Frequency Offset (CFO) Estimation Techniques for OFDM Systems" IOSR Journal of Electronics and Communication Engineering (IOSR-JECE), Volume 9, Issue 4, Ver. IV (Jul - Aug. 2014), PP 01-06.

[2] P.H. Moose, A Technique for Orthogonal Frequency Division Multiplexing Frequency Offset Correction, IEEE Trans. On communications, Vol. 42, No. 10, pp 2908-2914, October 1994.

[3] Classen, F. and Myer, H. Frequency Synchronization Algorithm for OFDM Systems Suitable for Communication Over Frequency Selective Fading Channels, IEEE VTC'94, pp. 1655 1659, June 1994.

[4] B. Sathish Kumar, K.R. Shankar Kumar, and R.Radhakrishnan, An Efficient Inter Carrier Interference Cancellation Schemes for OFDM Systems, IJCSIS, Vol. 6, No. 3, 2009 LTE; Evolved Universal Terrestrial Radio Access (E-UTRA);Physical channels and modulation (3GPP TS 36.211 version10.0.0 Release 10), ETSI TS 136 211 V10.0.0 (2011-01),Technical Specification .

[5] https://www.allaboutcircuits.com/technical-articles/xilinx-fft-ip-core-fast-fourier-transform-software-walkthrough/

[6] https://www.allaboutcircuits.com/technical-articles/xilinx-fft-ip-core-fast-fourier-transform-software-walkthrough/

[7] https://www.youtube.com/watch?v=B0HB9lSn4uU

[8] https://www.researchgate.net/publication/333029661_50_Years_of_FFT_Algorithms_and_Applications

[9] 3GPP TR 25.913 - v7.3.0, Requirements for EUTRA and EUTRAN, http://www.3gpp.org/ftp/Specs/archive/25%5Fseries/25.913/

[10] Channel Estimation (Mathworks, LTE Toolbox)

[11] https://www.sharetechnote.com/html/Communication_ChannelEstimation.html?fbclid=IwAR2cawzJoUVQ d2_JLSHF4uRPCWOtqg6uD9QCgSDZGAFs-KbeNhmAFzNzB-0

[12] https://stackoverflow.com/questions/16369698/how-to-pass-array-structure-between-two-verilog-modules

[13] https://timetoexplore.net/blog/fixed-point-numbers-in-verilog

C700 refrences:

[14] Data sheet: http://spectratronix.com/wp-content/uploads/2018/07/Spectratronix-C700-Datasheet_2015_V4.1.pdf

[15] Data sheet: http://spectratronix.com/wp-content/uploads/2018/07/Spectratronix-C700-Datasheet_2015_V4.1.pdf

[16] http://www.techplayon.com/master-information-block-mib/

[17] AN EXPERIMENTAL STUDY ON MOBILITY PERFORMANCE IN REAL LTE SCENARIOS USING SOFTWARE DEFINED RADIO

USRP N210 refrences:

[18] Manual: https://files.ettus.com/manual/page_usrp2.html

[19] Datasheet: https://www.ettus.com/all-products/un210-kit/

RTL_SDR references:

[20] https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr

[21] https://www.aliexpress.com/item/32756711496.html

Evrytania references:

[22] http://www.evrytania.com/lte-tools